UNIVERSITY OF WISCONSIN
DATA SCIENCE

DS 740

Data Mining

Iteratively Weighted Least Squares
Grouped Data and Robust Regression

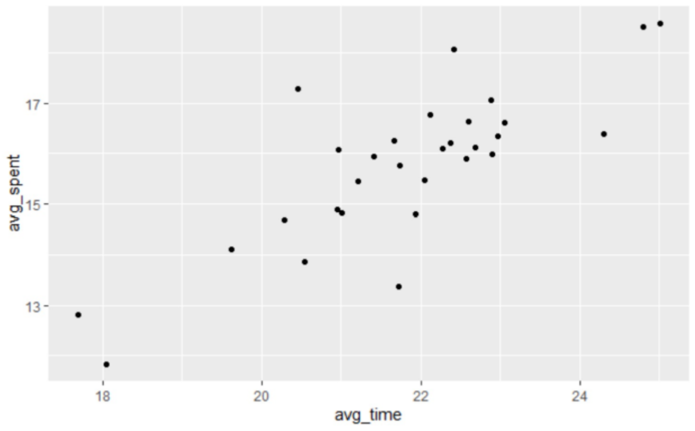
Important note: Transcripts are **not** substitutes for textbook assignments.

Learning Objectives

By the end of this lesson, you will be able to:

- Perform weighted least- squares regression for grouped data.
- Write a `while` loop to perform iteratively weighted least squares regression.
- Explain the differences between Huber’s weights and Tukey’s bisquare weights for robust regression.
- Perform robust regression in R using the `rlm()` function.

Are All Data Points Equally Reliable?



When performing linear regression, some data points may be more reliable than others. For example, suppose we want to investigate the relationship between the amount of time that customers spend shopping and the amount of money that they spend. To investigate this, we could ask 30 different stores to report the average amount of time that their customers spent in a store on a particular day and the average amount of money that the customers spent.

Notes:

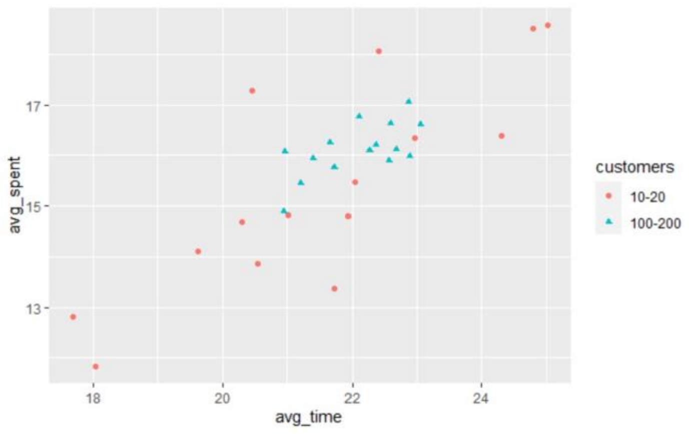
```
set.seed(30)
n_cust = c(sample(10:20, 15, replace=T), sample(100:200,
15, replace=T)) # n_cust = number of customers per week at
diff stores
avg_time = numeric(length = 30)
avg_spent = numeric(length = 30)
for(store in 1:30){
time = runif(n_cust[store], 5, 40)
spent = .5*time + 5 + rnorm(n_cust[store], 0, 5)
avg_time[store] = mean(time)
avg_spent[store] = mean(spent)
}

gf_point(avg_spent ~ avg_time)
```

Some Data Points May Be More Reliable Than Others

▲ Larger Stores

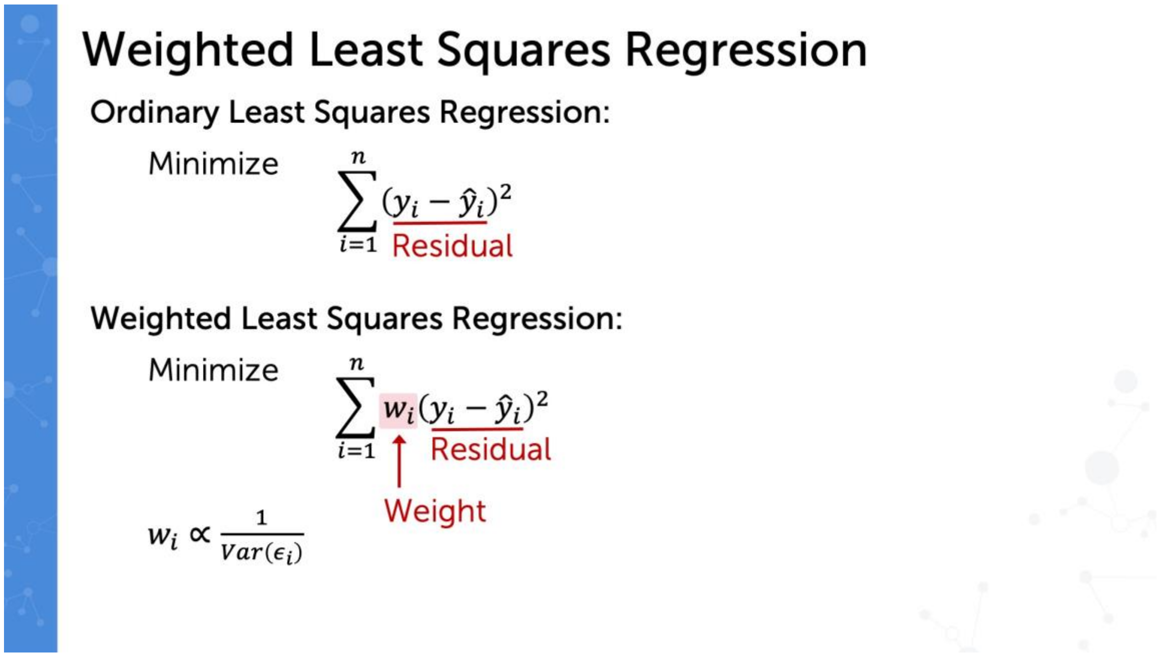
should be more influential than

● Smaller Stores



But what if some of those stores are small and some are large? We know that the variance of an average depends on the sample size, so we might want to place greater weight or importance on the averages from the large stores as we're fitting our linear regression model.

Notes:
```
customers = ifelse(n_cust < 21, "10-20", "100-200")
gf_point(avg_spent ~ avg_time, color =~ customers, shape =~
customers)
```



Weighted Least Squares Regression

Ordinary Least Squares Regression:

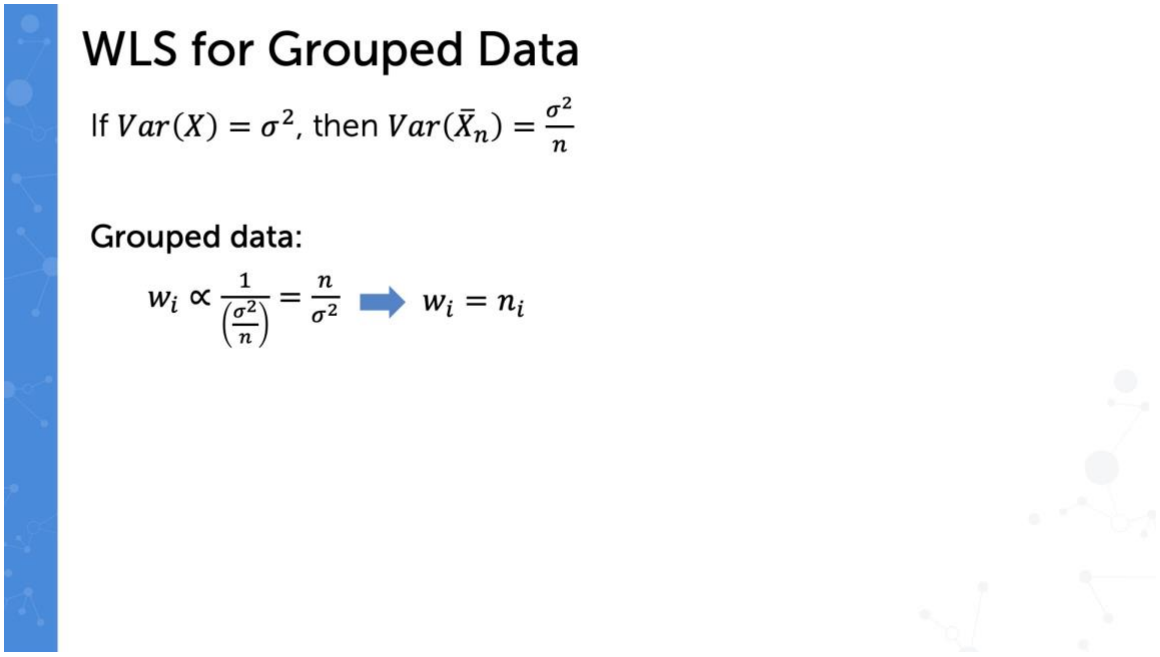
Minimize $$\sum_{i=1}^{n} \underbrace{(y_i - \hat{y}_i)^2}_{\text{Residual}}$$

Weighted Least Squares Regression:

Minimize $$\sum_{i=1}^{n} \underbrace{w_i}_{\text{Weight}} \underbrace{(y_i - \hat{y}_i)^2}_{\text{Residual}}$$

$$w_i \propto \frac{1}{Var(\epsilon_i)}$$

We can do this using weighted least squares regression where, instead of minimizing the sum of squared residuals as in ordinary least squares regression, we instead minimize the sum of squared residuals times a set of weights-- w sub i-- where the weight of the i-th data point is proportional to 1 over the variance of the noise term for that data point, epsilon sub i.



WLS for Grouped Data

If $Var(X) = \sigma^2$, then $Var(\bar{X}_n) = \frac{\sigma^2}{n}$

Grouped data:

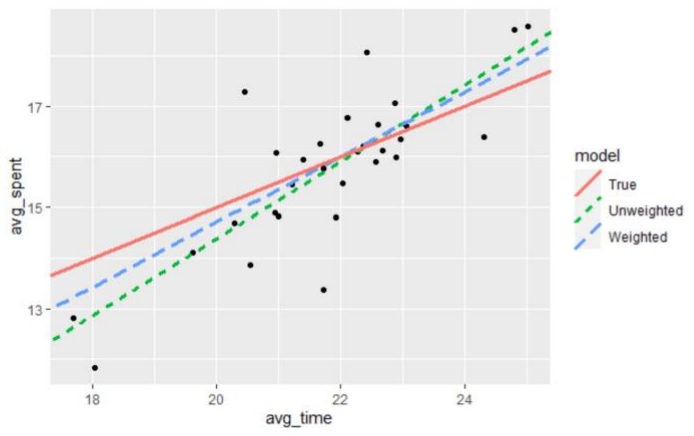
$$w_i \propto \frac{1}{\left(\frac{\sigma^2}{n}\right)} = \frac{n}{\sigma^2} \Rightarrow w_i = n_i$$

From statistics, we know that, if random variables are independent and have the same variance-- say, sigma squared-- then the variance of the mean of n of those random variables will be sigma squared divided by n. That means that, if our data consist of the averages of groups, the weights for weighted least squares will be proportional to 1 over this value or n over sigma squared. Here, sigma squared is the variance of the noise terms, epsilon sub i.

And in practice, we don't know this value. But that's OK because we only need the weights to be proportional to n over sigma squared. That means we can treat sigma squared as an unknown constant and simply use w sub i equal to n sub i, the number of observations that were averaged in the group to create the i-th data point.

Weighted Least Squares Regression in R

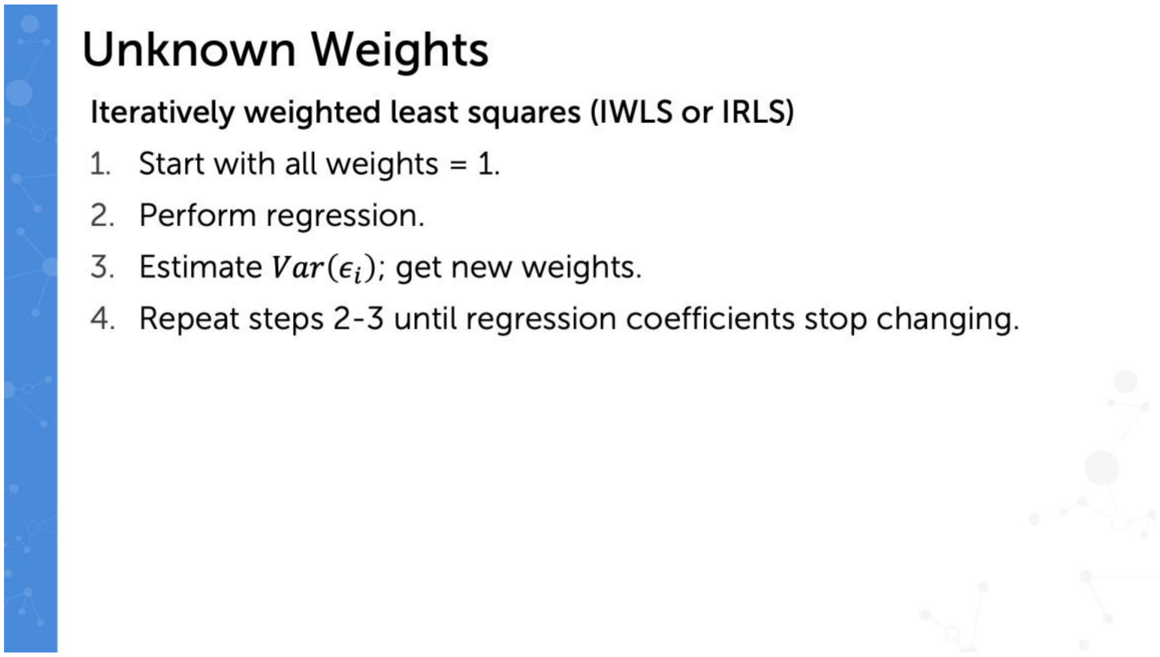
```
fit2 = lm(avgspent ~ avgtime, weights = n_cust)
```



We can perform weighted least squares regression in R using the LM function simply by setting the weights argument equal to a vector of the weights. If we do this for the sales example, the slope and y-intercept of our regression line change to become closer to the true slope and y-intercept that we used to simulate the data.

Notes:

```
fit1 = lm(avg_spent ~ avg_time)
fit2 = lm(avg_spent ~ avg_time, weights = n_cust)

lin_mod = data.frame(model = c("Unweighted", "Weighted",
"True"), intercepts = c(fit1$coef[1], fit2$coef[1], 5),
slopes = c(fit1$coef[2], fit2$coef[2], 0.5))

gf_point(avg_spent ~ avg_time) %>%
  gf_abline(slope = ~slopes, intercept = ~intercepts,
            color =~ model, lty =~ model, lwd = 1.1,
            data = lin_mod)
```



Unknown Weights

Iteratively weighted least squares (IWLS or IRLS)

1. Start with all weights = 1.
2. Perform regression.
3. Estimate $Var(\epsilon_i)$; get new weights.
4. Repeat steps 2-3 until regression coefficients stop changing.

Frequently, we won't know what weights to use. In that case, we can use iteratively weighted least squares-- sometimes, also called "iteratively reweighted least squares." This is actually the process that r uses under the hood when you use the glm function to perform logistic regression. In this process, we start with all of the weights equal to 1 and perform the regression. Then we use the results of that regression to estimate the variance of the noise terms and use those to get the new weights, and we repeat this process of performing a regression and estimating the variances until the regression coefficients stop changing.

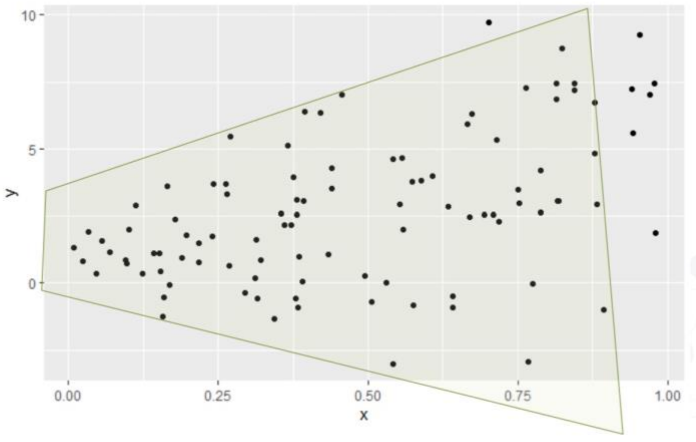
Using a Model to Estimate the Variance of Noise Terms

Common model:

$sd(\epsilon_i) \propto \hat{y}_i$

$Var(\epsilon_i) \propto \hat{y}_i^2$

$w_i = \frac{1}{\hat{y}_i^2}$



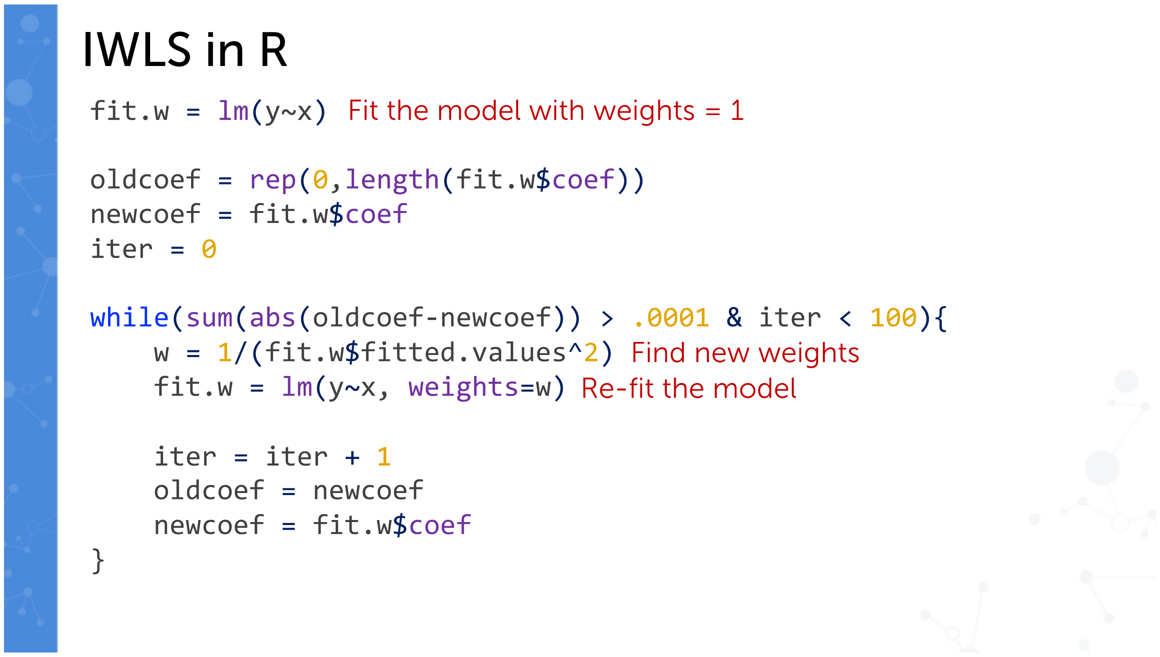
One common way to estimate the variance of the noise terms is to model the standard deviation of the noise terms as being proportional to the fitted or predicted values, y sub i hat. This is a good model to use if you see the vertical spread of the data points increasing as the average height of the data points increases-- producing a fan shape, which is a common form of heteroscedasticity. In this case, the variance of the noise terms will be proportional to the square of the fitted values.

So the weights would be equal to 1 over the square of the fitted values. Of course, we don't know the fitted values until we fit the regression line. So we need to iterate the process of fitting a regression line, choosing new weights, and then fitting a new regression line based on those weights.

Notes:

R Code:

```
set.seed(12)
x = runif(100)
y = 1 + 3*x + rnorm(100, 0, 1 + 3*x)
gf_point(y ~ x)
```

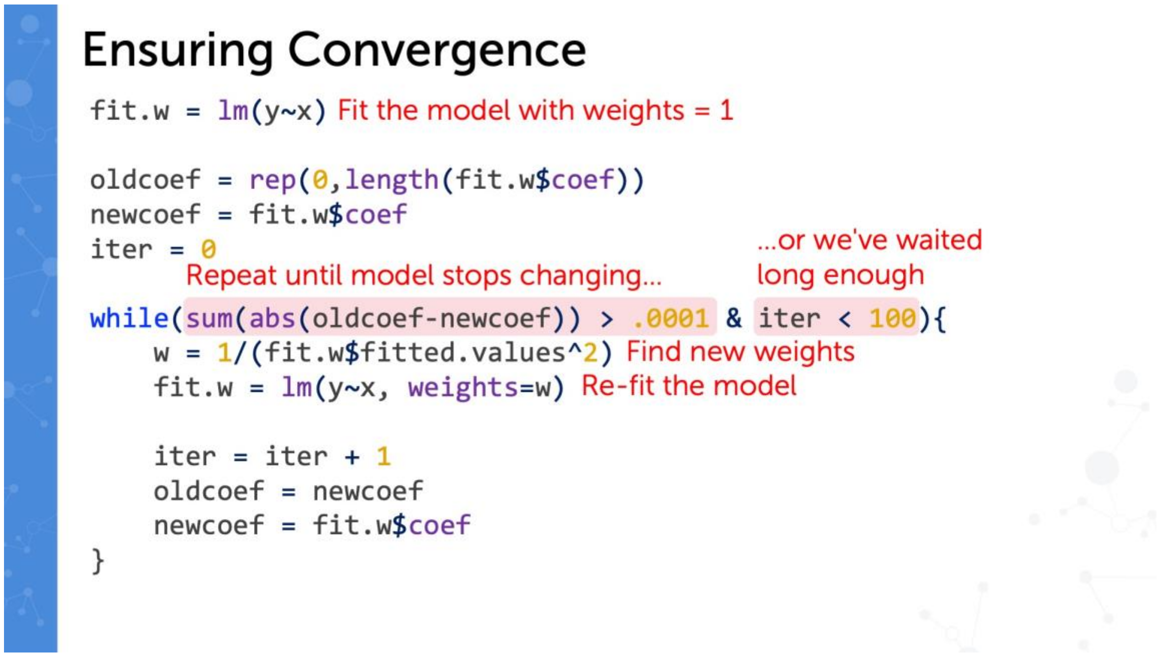


IWLS in R

```
fit.w = lm(y~x)  Fit the model with weights = 1

oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0

while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    w = 1/(fit.w$fitted.values^2)  Find new weights
    fit.w = lm(y~x, weights=w)  Re-fit the model

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

To do this in r, we could start by using the lm function to fit the model with weights equal to 1. Then we could use a while loop to iterate the process of finding new weights and refitting the model with those weights.

Notes:
```
fit.w = lm(y~x)

oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0

while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    w = 1/(fit.w$fitted.values^2)
    fit.w = lm(y~x, weights=w)

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

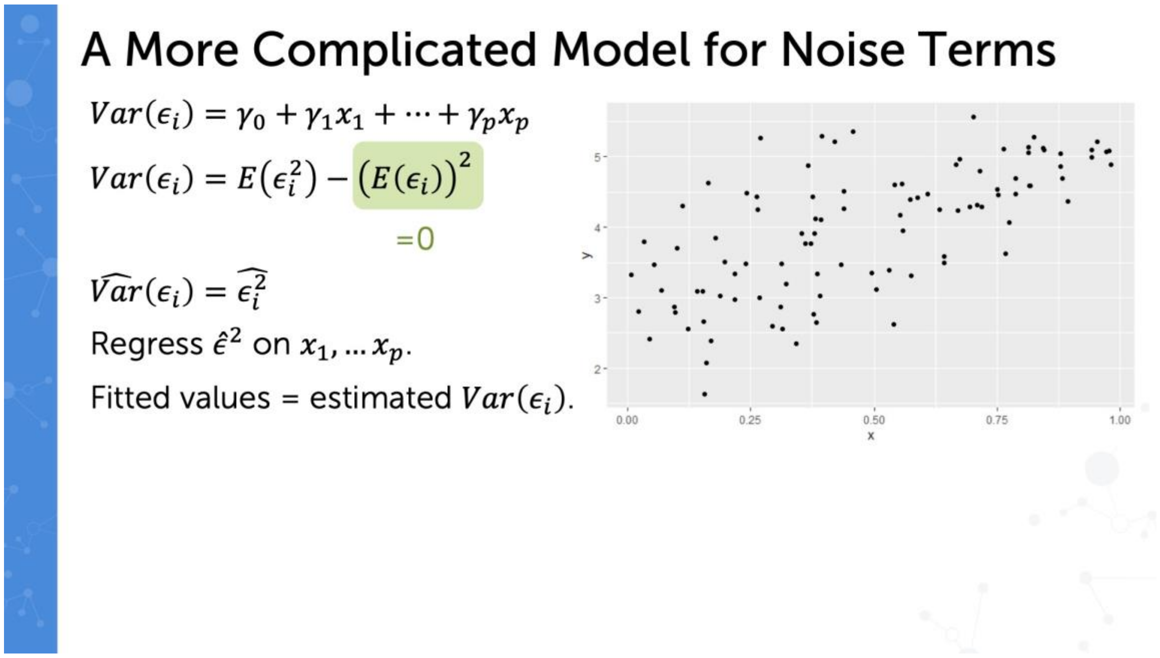


Ensuring Convergence

```
fit.w = lm(y~x) Fit the model with weights = 1

oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0                                         ...or we've waited
        Repeat until model stops changing...     long enough
while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    w = 1/(fit.w$fitted.values^2) Find new weights
    fit.w = lm(y~x, weights=w) Re-fit the model

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

We iterate this process until the old coefficients from the previous iterations regression model and the new coefficients from the current regression model are sufficiently close together or until the number of iterations reaches a preset threshold. This prevents your while loop from running for an excessively long time if your model is poorly specified or there is a bug in your code. It's a good idea to check the value of iter after the while loop has run. If it's equal to your preset threshold, it likely means that your model didn't converge. You may need to respecify your model or increase your preset threshold in order to be able to trust the results of your ultimate regression.

Notes:
```
fit.w = lm(y~x)

oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0

while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    w = 1/(fit.w$fitted.values^2)
    fit.w = lm(y~x, weights=w)

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```



A More Complicated Model for Noise Terms

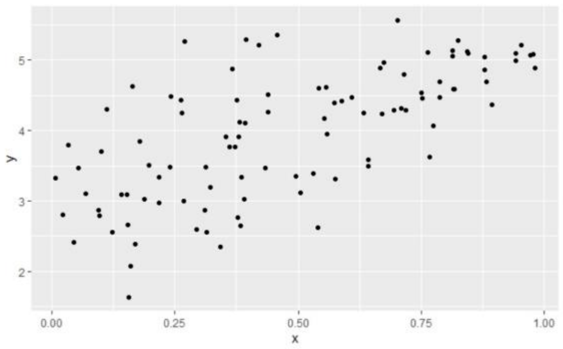
$Var(\epsilon_i) = \gamma_0 + \gamma_1 x_1 + \cdots + \gamma_p x_p$

$Var(\epsilon_i) = E(\epsilon_i^2) - \underbrace{\left(E(\epsilon_i)\right)^2}_{=0}$

$\widehat{Var}(\epsilon_i) = \widehat{\epsilon_i^2}$

Regress $\hat{\epsilon}^2$ on $x_1, \ldots x_p$.

Fitted values = estimated $Var(\epsilon_i)$.



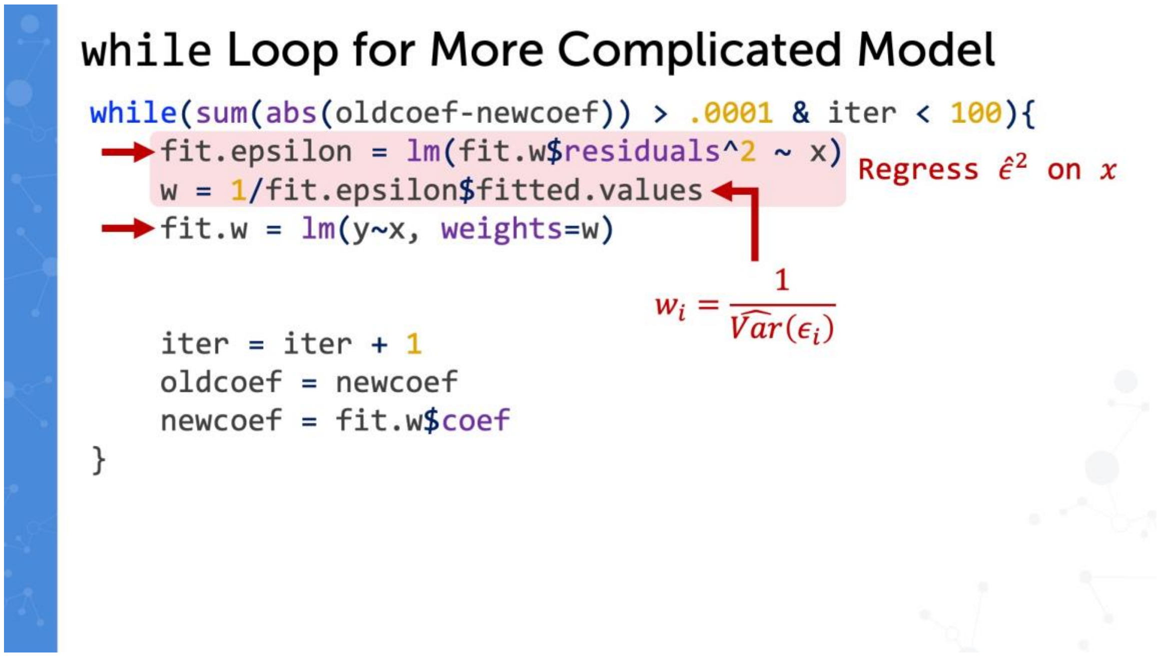
A more complicated model for the variance of the noise terms would be that the variance is a linear function of the predictor variables with unknown coefficients, gamma. These are different from the beta coefficients that control the relationship between the response variable, y sub i, and the predictor variables. To compute the weights in this situation, recall from statistics that the variance of a random variable is the expected value of its square minus the square of its expected value.

But the mean of the noise terms is 0, that means we can estimate the variance by estimating the square of the noise terms. So in our iteratively weighted least squares, we want to regress the squares of the residuals on the predictor variables. The fitted values from that regression will be our estimated values of the variance of the noise terms.

Notes:

R Code to generate graph:

```
set.seed(12)
x = runif(100)
y = 3 + 2*x + rnorm(100, 0, sqrt(1-x)) # rnorm uses standard
deviation instead of variance
gf_point(y ~ x)
```



while Loop for More Complicated Model

```
while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    fit.epsilon = lm(fit.w$residuals^2 ~ x)
    w = 1/fit.epsilon$fitted.values
    fit.w = lm(y~x, weights=w)


    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

Regress $\hat{\epsilon}^2$ on x
$w_i = \frac{1}{\widehat{Var}(\epsilon_i)}$

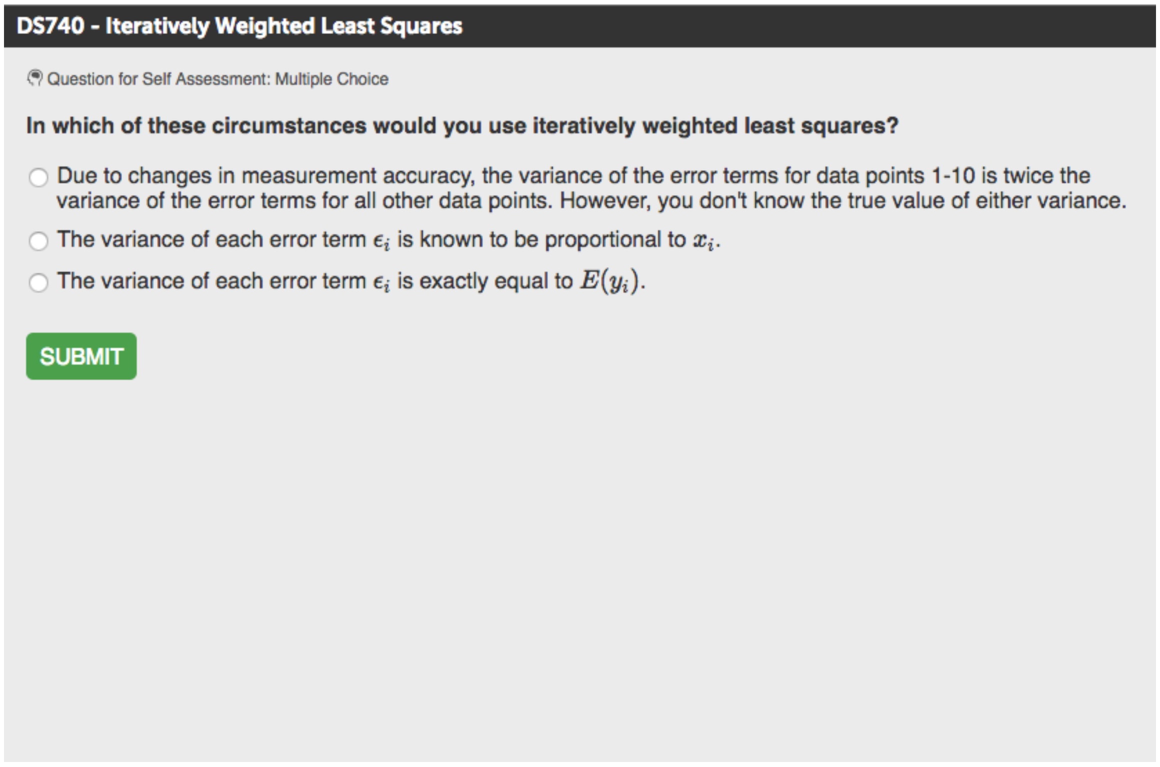
To fit this model in r, we use a while loop with the same structure as we saw before. The only difference is how we calculate the weights-- first, by regressing the squared residuals on x and then by setting the weights equal to 1 over the fitted values. Notice that, in this while loop, we use the lm function twice for each iteration-- one time to regress the squared residuals on x and a second time to regress the response variable, y on x.

Notes:

The code that would come before the while loop is identical to the code for the simpler model:

```
fit.w = lm(y~x)

oldcoef = rep(0, length(fit.w$coef))
newcoef = fit.w$coef
iter = 0
```

Question 1


DS740 - Iteratively Weighted Least Squares

Question for Self Assessment: Multiple Choice

In which of these circumstances would you use iteratively weighted least squares?

- ◯ Due to changes in measurement accuracy, the variance of the error terms for data points 1-10 is twice the variance of the error terms for all other data points. However, you don't know the true value of either variance.
- ◯ The variance of each error term ϵ_i is known to be proportional to x_i.
- ◯ The variance of each error term ϵ_i is exactly equal to $E(y_i)$.

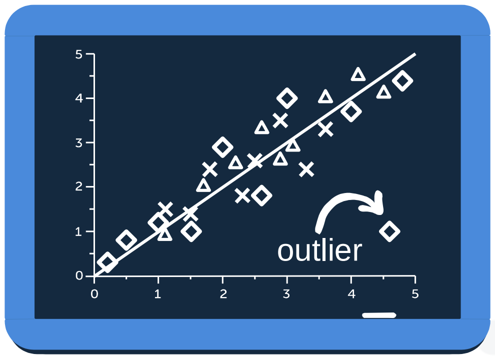
Answer is at the end of the transcript

Robust Regression

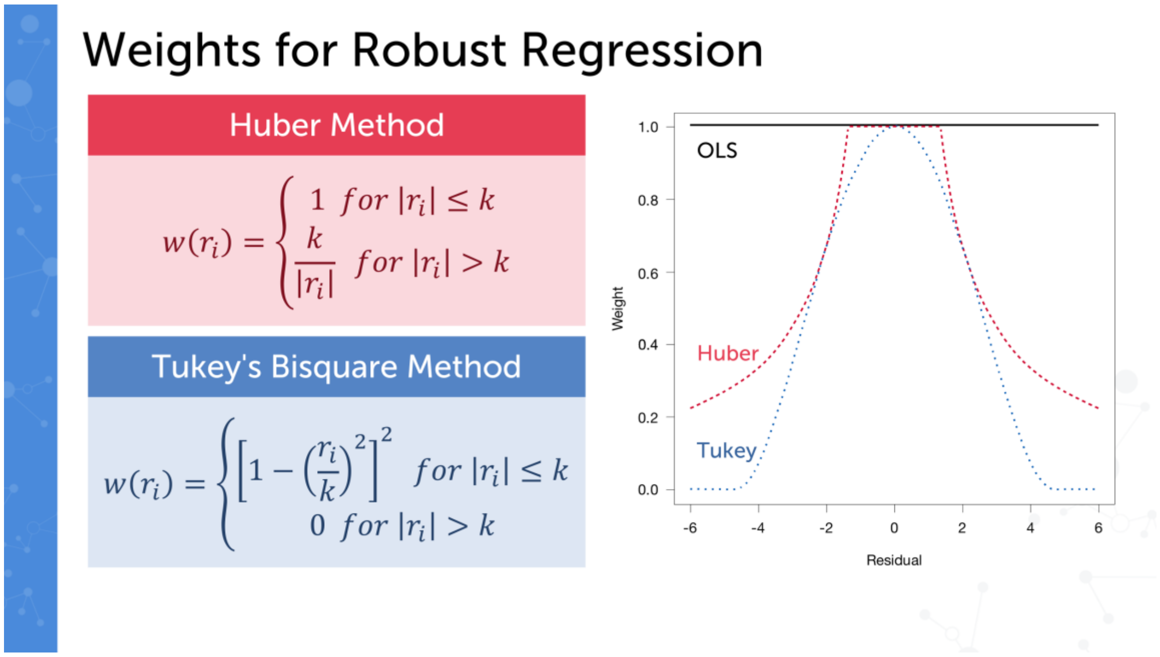
Put less weight on outliers or influential points…
..without excluding them.

Use IWLS

Weights decrease as the… ..absolute value of residuals increase.



Robust regression, sometimes called "resistant regression," refers to putting less weight on outliers or influential points. This allows the outliers to have less influence on the regression model without excluding the outliers altogether, which you generally don't want to do if the outliers didn't come from a data collection or data entry error. We can perform robust regression by using iteratively weighted least squares with weights that decrease as the absolute value of the residuals increase.



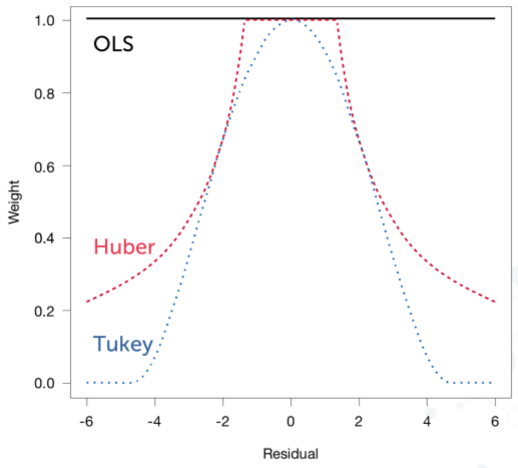
Weights for Robust Regression

Huber Method

$$w(r_i) = \begin{cases} 1 \ \ for \ |r_i| \le k \\ \dfrac{k}{|r_i|} \ \ for \ |r_i| > k \end{cases}$$

Tukey's Bisquare Method

$$w(r_i) = \begin{cases} \left[1 - \left(\frac{r_i}{k}\right)^2\right]^2 \ \ for \ |r_i| \le k \\ 0 \ \ for \ |r_i| > k \end{cases}$$

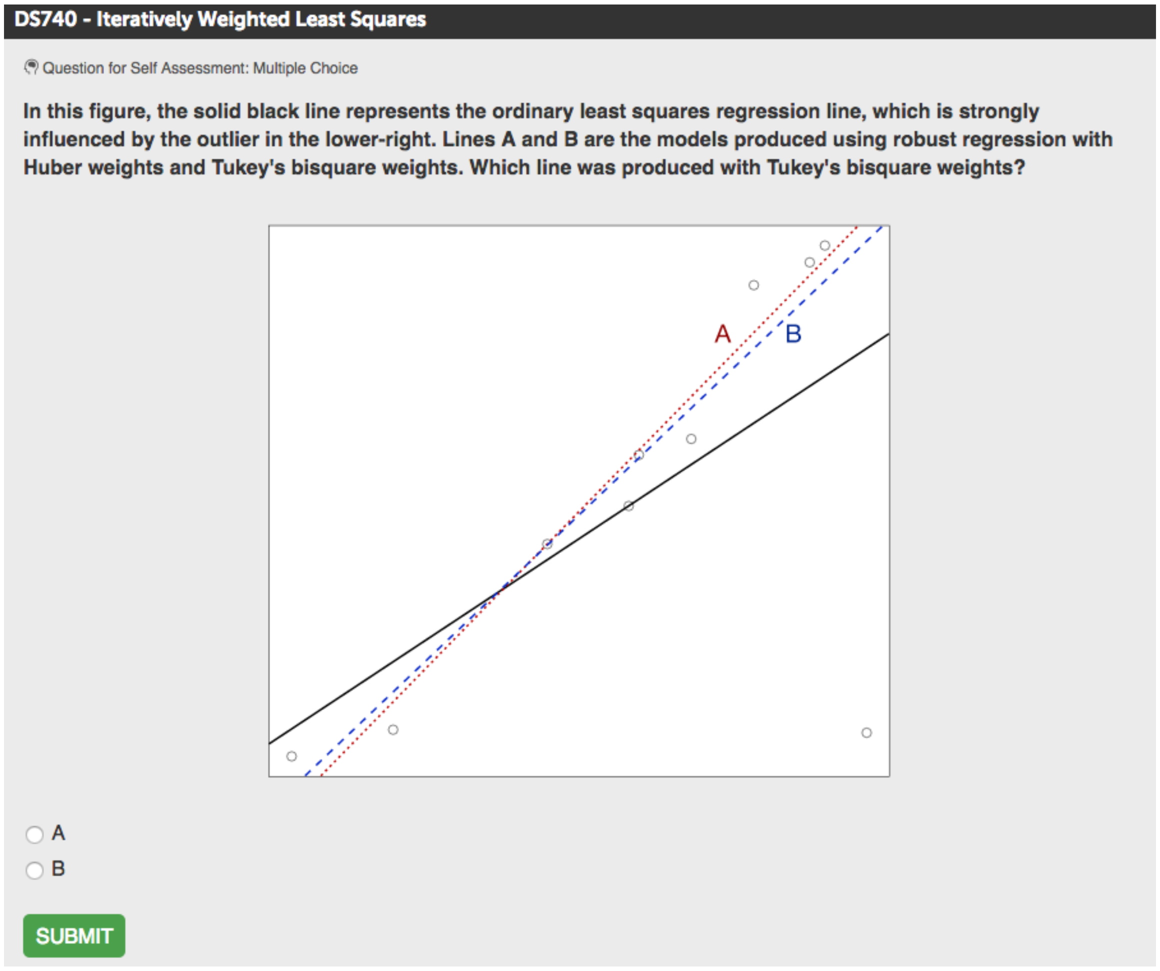


Two common forms of weights for robust regression are the Huber method and Tukey's bisquare or biweight method. In both of these formulas, r sub i represents the residual of the i-th data point and k is a tuning parameter. As you can see in the graph, both of these methods give a weight of 1 to data points with a residual of 0.

That is, points that fall exactly on the regression line. So for these points, robust regression acts just like ordinary least squares. For points with larger residuals, the two-key method has weights that decrease more quickly-- eventually, reaching 0. So the Tukey's method tends to be less influenced by outliers compared to the Huber method.

Notes:
For more about the Huber and Tukey's bisquare methods, see Robust Regression, Appendix to An R and S-PLUS Companion to Applied Regression - John Fox (PDF).

Question 2



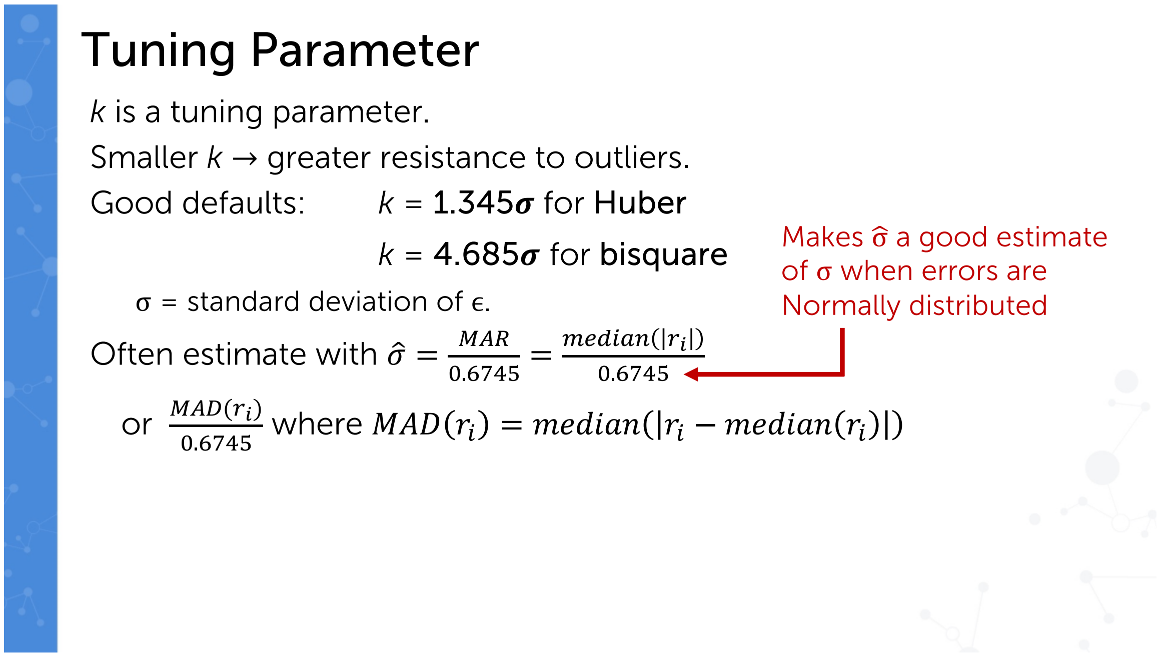
DS740 - Iteratively Weighted Least Squares

Question for Self Assessment: Multiple Choice

In this figure, the solid black line represents the ordinary least squares regression line, which is strongly influenced by the outlier in the lower-right. Lines A and B are the models produced using robust regression with Huber weights and Tukey's bisquare weights. Which line was produced with Tukey's bisquare weights?

○ A
○ B

Answer is at the end of the transcript



Tuning Parameter

k is a tuning parameter.

Smaller $k \rightarrow$ greater resistance to outliers.

Good defaults: $k = 1.345\boldsymbol{\sigma}$ for Huber

$k = 4.685\boldsymbol{\sigma}$ for bisquare

σ = standard deviation of ϵ.

Often estimate with $\hat{\sigma} = \frac{MAR}{0.6745} = \frac{median(|r_i|)}{0.6745}$

Makes $\hat{\sigma}$ a good estimate of σ when errors are Normally distributed

or $\frac{MAD(r_i)}{0.6745}$ where $MAD(r_i) = median(|r_i - median(r_i)|)$

Data points with absolute value residuals larger than the tuning parameter, k, will have weights less than 1 for the Huber method and weights equal to 0 for the bisquare method. So the smaller the value of k you choose, the greater resistance you'll have to outliers. However, if your error terms truly are normally distributed, you'll get better results from using a larger value of k because you'll be downweighting fewer points. Good default values for k are 1.345 times sigma for the Huber method and 4.685 times sigma for the bisquare method where sigma represents the standard deviation of the noise terms.

To estimate sigma, we could just use the standard deviation of the residuals. But we often use a method that's more robust to outliers, such as the median absolute residual divided by 0.6745 where the denominator is chosen to make sigma hat a good estimate of sigma when the errors are normally distributed. If you're concerned that the residuals may not be symmetrically distributed around 0, you could replace the median absolute residual with the median absolute deviation.

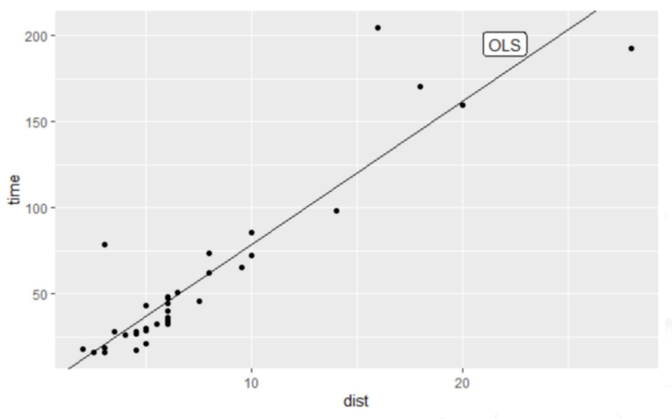
Notes:

When the errors are Normally distributed, \(\hat\sigma\) is an asymptotically unbiased estimator for \(\sigma\).

Example: Hill Racing

Winning times (in minutes) of races in Scotland, as a function of the distance (in miles)

```
library(MASS)
hills %>%
  gf_point(time ~ dist)
```

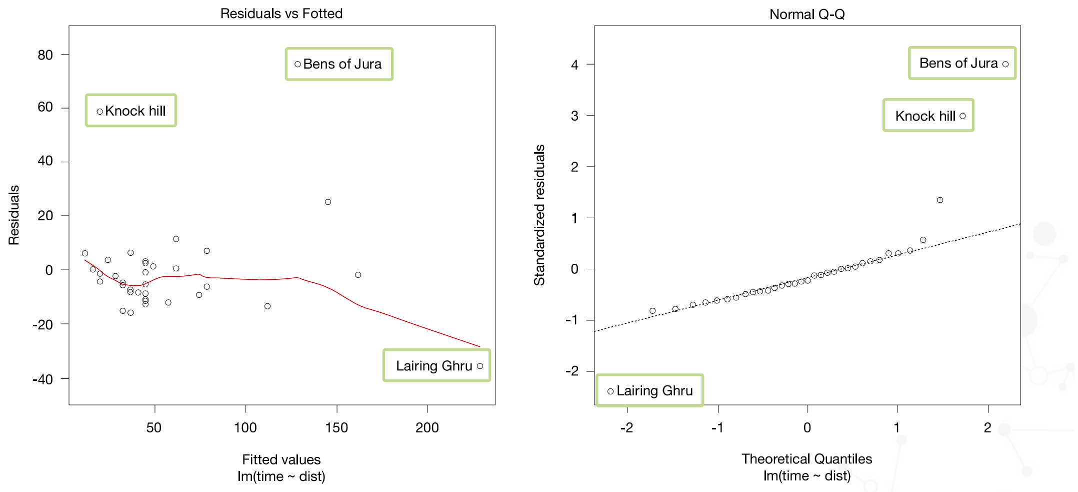


Notes:

```
library(MASS)
fit_hills = lm(time ~ dist, data = hills)

hills %>%
  gf_point(time ~ dist) %>%
  gf_abline(intercept = fit_hills$coef[1], slope =
fit_hills$coef[2]) %>%
  gf_label(195 ~ 22, label = "OLS")
```

Residual Plots

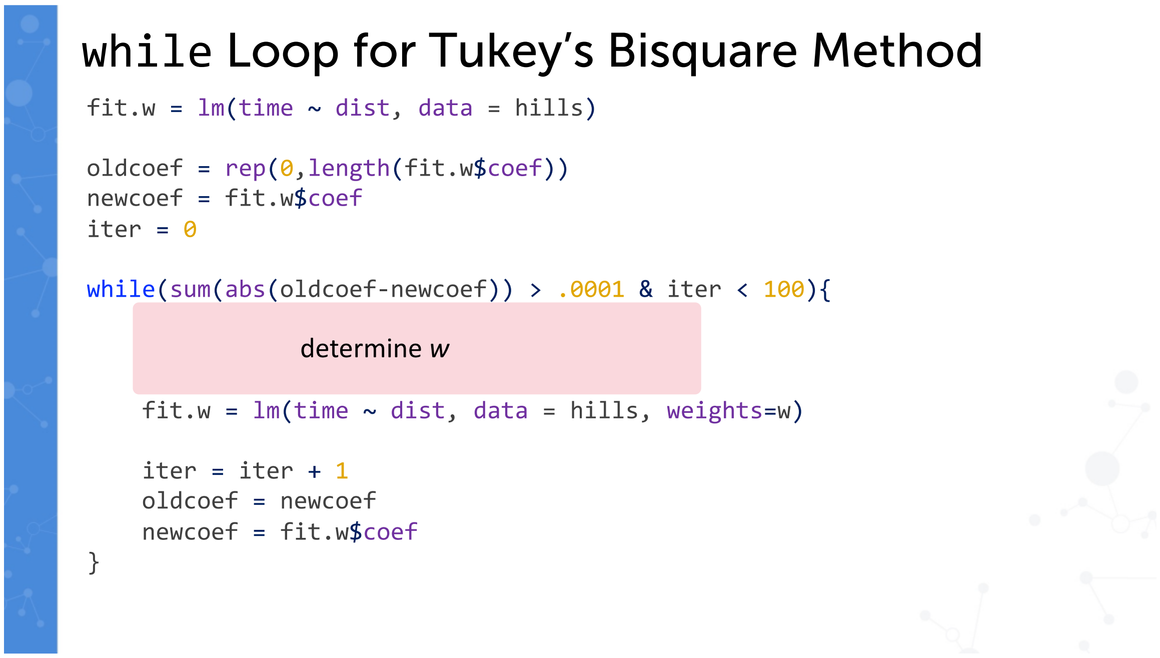
```
fit_hills = lm(time ~ dist, data = hills)
plot(fit_hills)
```



Here, we've applied ordinary least squares regression to the hill racing data, and we're looking at the diagnostic plots. If the assumptions of ordinary least squares regression are appropriate for this dataset, then the residuals versus fitted values graph on the left should show the residuals as a random cloud of points around the line y equal 0 with no discernible trend in the red smoothing spline. The normal q,q graph on the right should show the points falling closely along the line y equals x to indicate that the standardized residuals closely follow the expected normal distribution. For this dataset, both of these graphs look OK for most of the data points, but there are three outliers that r has picked out automatically and labeled with their row names from the dataset. If the dataset didn't have row names, then it would label them with their row numbers.

Notes:

```
fit_hills = lm(time ~ dist, data = hills)
plot(fit_hills)
```



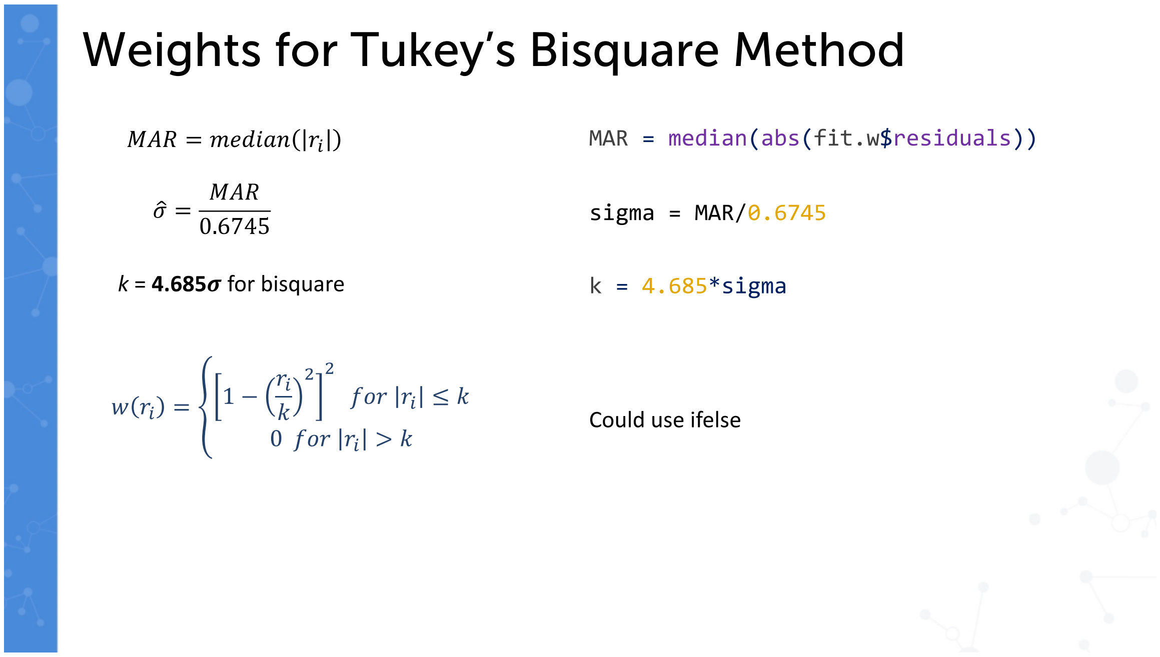
while Loop for Tukey's Bisquare Method

```
fit.w = lm(time ~ dist, data = hills)

oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0

while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
                determine w

    fit.w = lm(time ~ dist, data = hills, weights=w)

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

Let's use a while loop to implement Tukey's bi-square method of robust regression. The structure of this is exactly the same as we've used for other versions of iteratively weighted least squares. The only thing we need to change is how we compute the weights.

Notes:

```
fit.w = lm(time ~ dist, data = hills)
oldcoef = rep(0,length(fit.w$coef))
newcoef = fit.w$coef
iter = 0
while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
### determine w ###
   fit.w = lm(time ~ dist, data = hills, weights=w)
   iter = iter + 1
   oldcoef = newcoef
   newcoef = fit.w$coef
}
```



Weights for Tukey's Bisquare Method

$MAR = median(|r_i|)$

```
MAR = median(abs(fit.w$residuals))
```

$\hat{\sigma} = \frac{MAR}{0.6745}$

```
sigma = MAR/0.6745
```

k = $\mathbf{4.685\sigma}$ for bisquare

```
k = 4.685*sigma
```

$$w(r_i) = \begin{cases} \left[1 - \left(\frac{r_i}{k}\right)^2\right]^2 & for\ |r_i| \le k \\ 0\ for\ |r_i| > k \end{cases}$$

Could use ifelse

To compute the weights for Tukey's method, we need to start by computing the median of the absolute value of the residuals, which we can do using the code shown here. Note that we're extracting the residuals component of the linear regression object, which we called fit.w, using dollar sign notation.

Then we need to compute sigma hat by dividing the median of the absolute value of the residuals by 0.6745, and we need to compute k which, for Tukey's method, is typically done by multiplying sigma by 4.685. Finally, we need to compute the weights. For Tukey's method, this has a pretty complicated piecewise defined function.

We could do this computation using an if else statement to first check whether the absolute value of r was less than or equal to k, and then determine the value of the weight. But there's another method we could use that ends up being pretty clever.

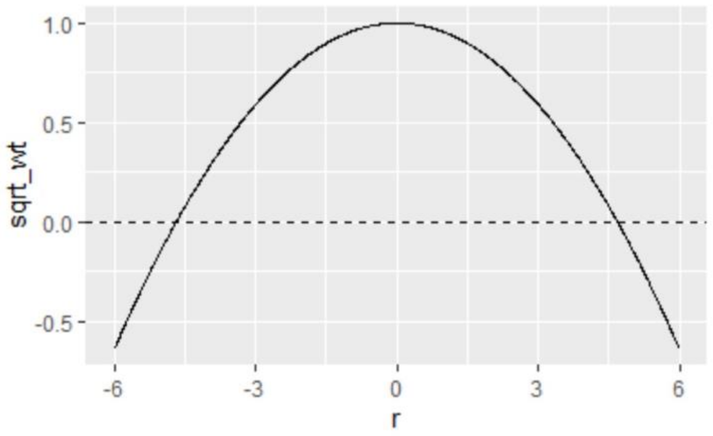
Notes:

```
MAR = median(abs(fit.w$residuals))
sigma = MAR/0.6745
k = 4.685*sigma
```

Computing *w* cleverly

$$w(r_i) = \begin{cases} \left[1 - \left(\frac{r_i}{k}\right)^2\right]^2 & for\ |r_i| \le k \\ 0 & for\ |r_i| > k \end{cases}$$

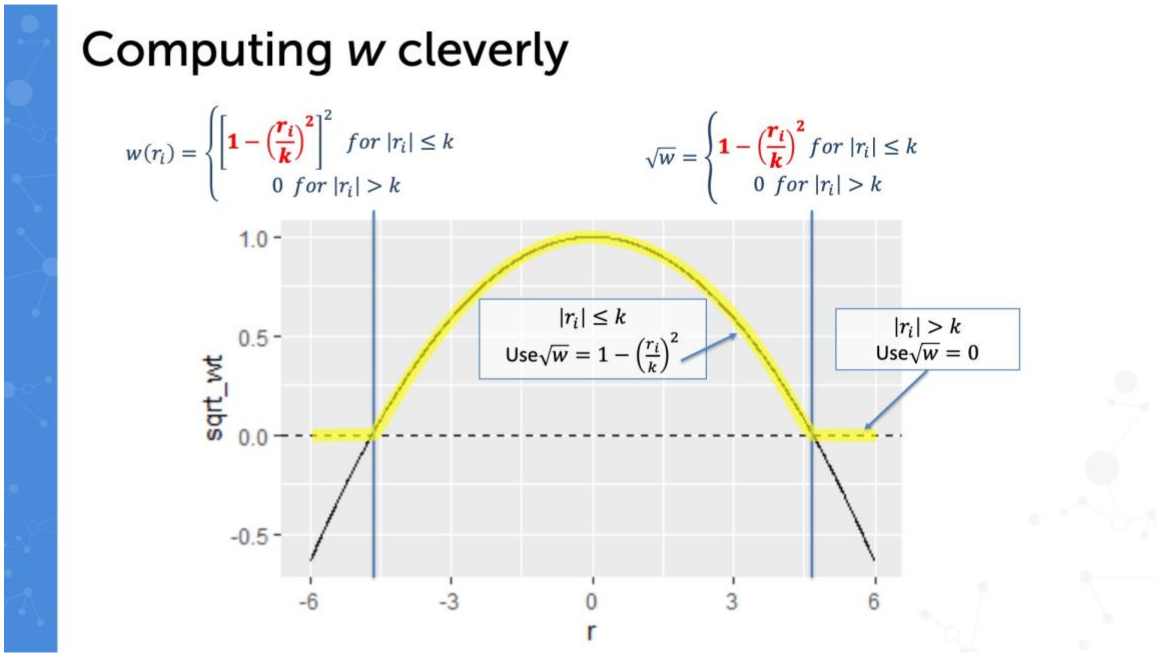
$$\sqrt{w} = \begin{cases} 1 - \left(\frac{r_i}{k}\right)^2 & for\ |r_i| \le k \\ 0 & for\ |r_i| > k \end{cases}$$



To use the clever approach of computing w, let's start by focusing on the portion of the weight function that's inside the square brackets that's being squared. This means that if the absolute value of r is less than or equal to k, then the square root of w is just the portion inside the square brackets. That is 1 minus the square of r over k. And if the residual's absolute value is greater than k, then we just have the square root of 0, which is 0.

Next, let's graph these two pieces of the piecewise defined function for the square root of w. We have a parabola, 1 minus the quantity r over k-squared, and we have a horizontal line at a height of 0.

Notes:
Notice that at each point, we choose the higher of the two lines.

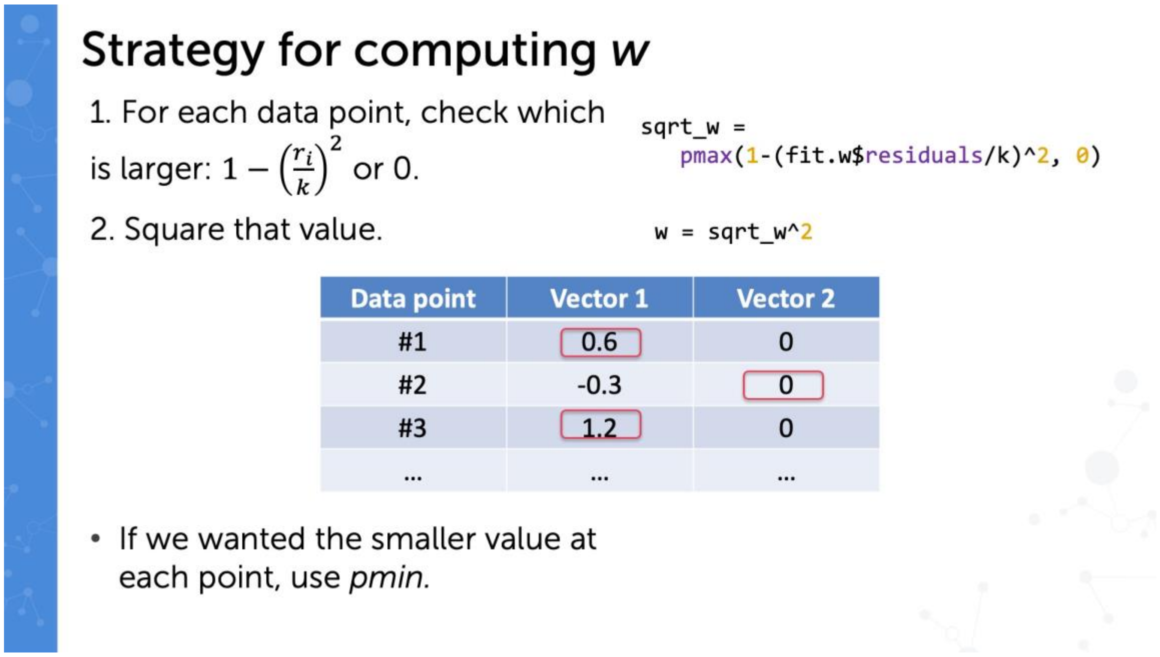


Computing *w* cleverly

$$w(r_i) = \begin{cases} \left[\mathbf{1} - \left(\frac{\mathbf{r_i}}{\mathbf{k}}\right)^{\mathbf{2}}\right]^2 & for\ |r_i| \le k \\ 0\ for\ |r_i| > k & \end{cases}$$

$$\sqrt{w} = \begin{cases} \mathbf{1} - \left(\frac{\mathbf{r_i}}{\mathbf{k}}\right)^{\mathbf{2}} & for\ |r_i| \le k \\ 0\ for\ |r_i| > k & \end{cases}$$

The value of the square root of w is going to be the height of one of these two lines, depending on the value of r, where we are horizontally on the graph. If we're in the middle of the graph where the absolute value of r is less than or equal to k, then the square root of w is the parabola. If we're outside of the lines where the absolute value of r is greater than k, then the square root of w is 0. Notice that in either of these cases, the square root of w is equal to the higher of the two lines.



Strategy for computing *w*

1. For each data point, check which is larger: $1-\left(\frac{r_i}{k}\right)^2$ or 0.
2. Square that value.

```
sqrt_w =
   pmax(1-(fit.w$residuals/k)^2, 0)

w = sqrt_w^2
```

Data point	Vector 1	Vector 2
#1	0.6	0
#2	-0.3	0
#3	1.2	0
…	…	…

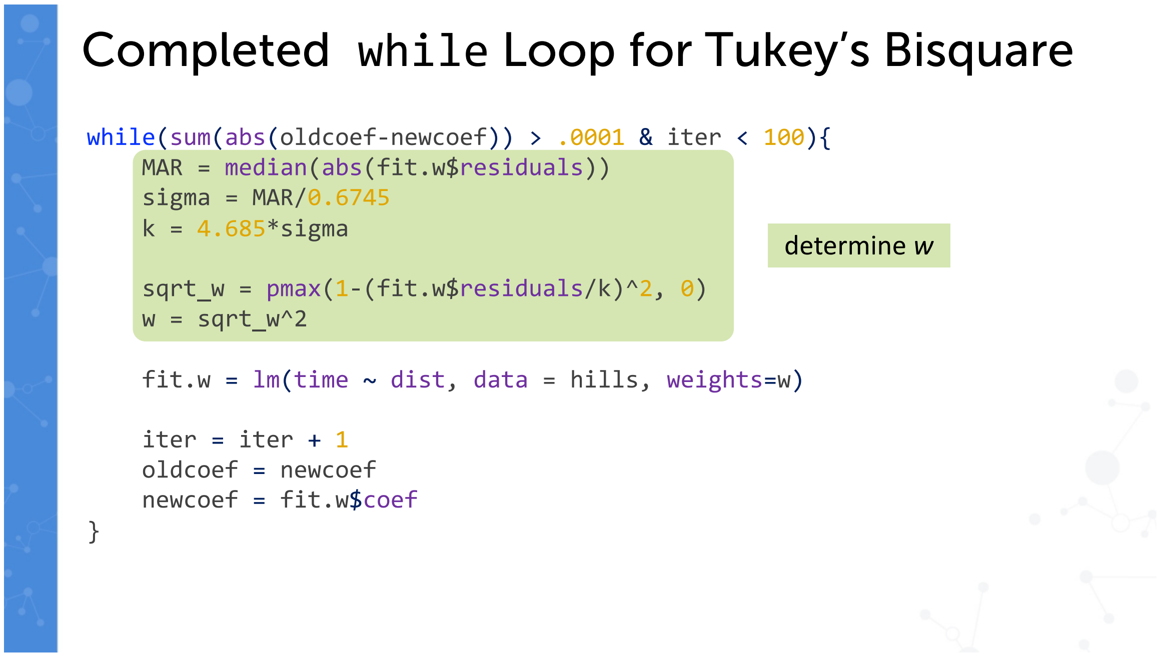
- If we wanted the smaller value at each point, use *pmin.*

So our strategy for computing the weights cleverly will be for each data point, we'll check which value is larger, 1 minus the square of the residuals over k or 0. Then we'll square that value. To do this in R, We'll use the pmax function, which stands for parallel maximum. The way this function works is we give it as arguments to vectors. In this case, the first argument is the vector of heights of the parabola, and the second argument is just the number 0, which will get recycled into a vector as many times as we need to be the same length as the first argument.

Then, the pmax function goes through each position of the two vectors and identifies which of the two numbers is the maximum. If we wanted to use the smaller value at each point, we could use pmin to get the parallel minimum.

Notes:

```
sqrt_w = pmax(1-(fit.w$residuals/k)^2, 0)
w = sqrt_w^2
```



Completed `while` Loop for Tukey's Bisquare

```
while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    MAR = median(abs(fit.w$residuals))
    sigma = MAR/0.6745
    k = 4.685*sigma

    sqrt_w = pmax(1-(fit.w$residuals/k)^2, 0)
    w = sqrt_w^2

    fit.w = lm(time ~ dist, data = hills, weights=w)

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

determine *w*

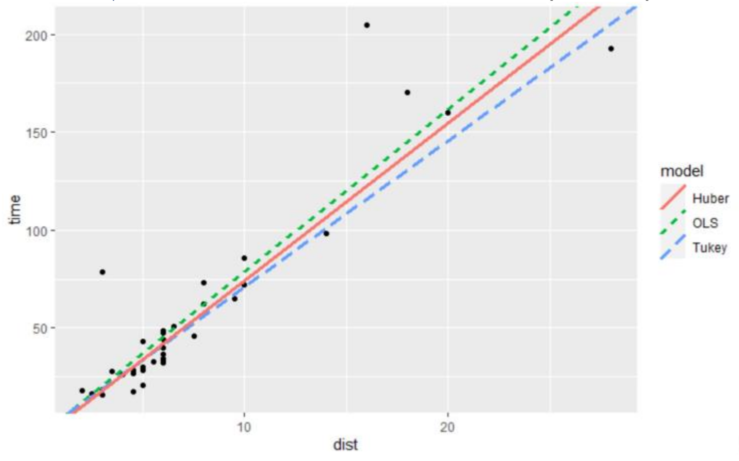
When we put all these pieces together, our completed while loop for the Tukey's bi-square method looks like this.

Notes:

```
while(sum(abs(oldcoef-newcoef)) > .0001 & iter < 100){
    MAR = median(abs(fit.w$residuals))
    sigma = MAR/0.6745
    k = 4.685*sigma
    sqrt_w = pmax(1-(fit.w$residuals/k)^2, 0)
    w = sqrt_w^2
    fit.w = lm(time ~ dist, data = hills, weights=w)

    iter = iter + 1
    oldcoef = newcoef
    newcoef = fit.w$coef
}
```

Robust Regression via `rlm()`

```r
library(MASS)
fit_bisquare = rlm(time ~ dist, data = hills, psi = psi.bisquare)
fit_huber = rlm(time ~ dist, data = hills, psi = psi.huber)
```



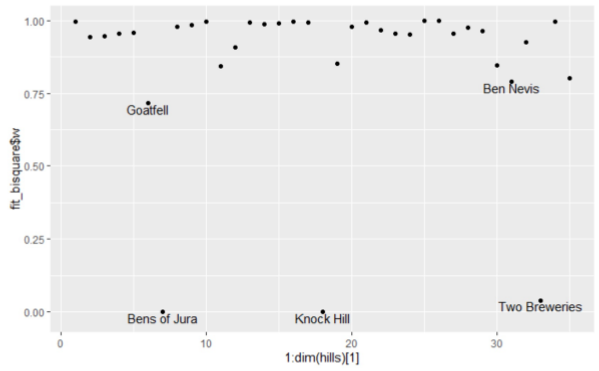
We can also implement robust regression using the rlm function in the MASS library. When we apply this to the hill racing data, we see that both the Huber and the bisquare methods give us a fairly different slope compared to ordinary least squares regression. It's not surprising to see that the bisquare method produces a regression line that's more different from ordinary least squares compared to the Huber method because we know that the bisquare method downweights outliers more.

Notes:

```r
library(MASS)
fit_bisquare = rlm(time ~ dist, data = hills, psi = psi.bisquare)
fit_huber = rlm(time ~ dist, data = hills, psi = psi.huber)

rlm_mod = data.frame(model = c("OLS", "Tukey", "Huber"),
            intercepts = c(fit_hills$coef[1],
fit_bisquare$coef[1], fit_huber$coef[1]),
            slopes = c(fit_hills$coef[2], fit_bisquare$coef[2],
fit_huber$coef[2]))

hills %>%
  gf_point(time ~ dist) %>%
  gf_abline(slope = ~slopes, intercept = ~intercepts,
            color =~ model, lty =~ model, lwd = 1.1,
            data = rlm_mod)
```

```
hills2 <- hills %>% tibble::rownames_to_column("Race")
hills2 <- hills2 %>%
  mutate(my_label = case_when(fit_bisquare$w < .8 ~ Race,
                              TRUE ~ ""))
gf_point(fit_bisquare$w ~ 1:dim(hills)[1]) %>%
  gf_text(fit_bisquare$w - .02 ~ 1:dim(hills)[1],
          label = hills2$my_label)
```

which points to label



Use *w*, not *weights*

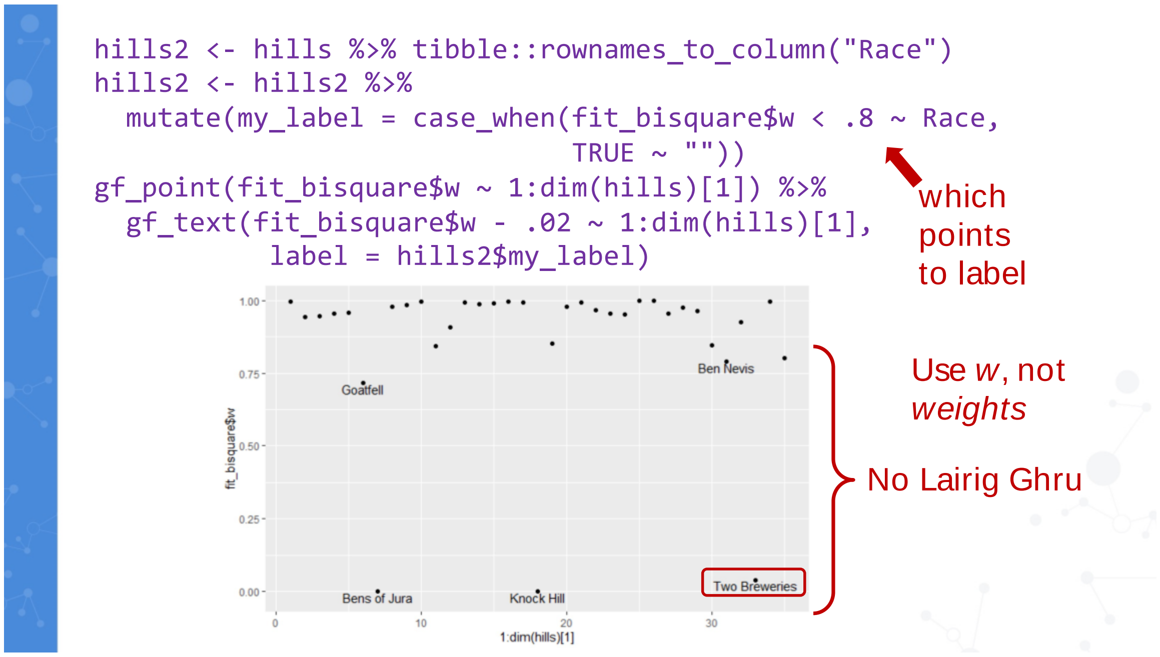
It can be helpful to plot the final weights of the robust regression as a function of the index of each point in the data set. Here, I'm using the row names to column function from the tibble package to convert the names of each row in the data set into a new column called race, so that I can use these as part of my labels in the graph. Next, I'm using the mutate and case when functions to define a new column called My Label. To decide which data points will be labeled, I'm testing the value of the w component from my robust regression object.

If this is less than 0.8, meaning a relatively small weight, then I'll give the point the label of its race name. For all other points, I'll use the label of an empty string, meaning no label. One important point here is to use the w component of the robust regression object to plot the final weights, not the weights component, which plots the initial weights. Those usually aren't very interesting because they're usually all ones.

Finally, we're ready to make our graph. This is just a scatter plot of the final weights with the x-axis being the index of each data point in the data set, and then using the GF text function to add labels to each of the points. Here, I'm adding a small offset to the y-value of each point so that the labels won't be right on top of the dots.

Notes:

```
hills2 <- hills %>%
  tibble::rownames_to_column("Race")

hills2 <- hills2 %>%
  mutate(my_label = case_when(fit_bisquare$w < .8 ~ Race,
                              TRUE ~ ""))

gf_point(fit_bisquare$w ~ 1:dim(hills)[1]) %>%
  gf_text(fit_bisquare$w - .02 ~ 1:dim(hills)[1],
          label = hills2$my_label)
```

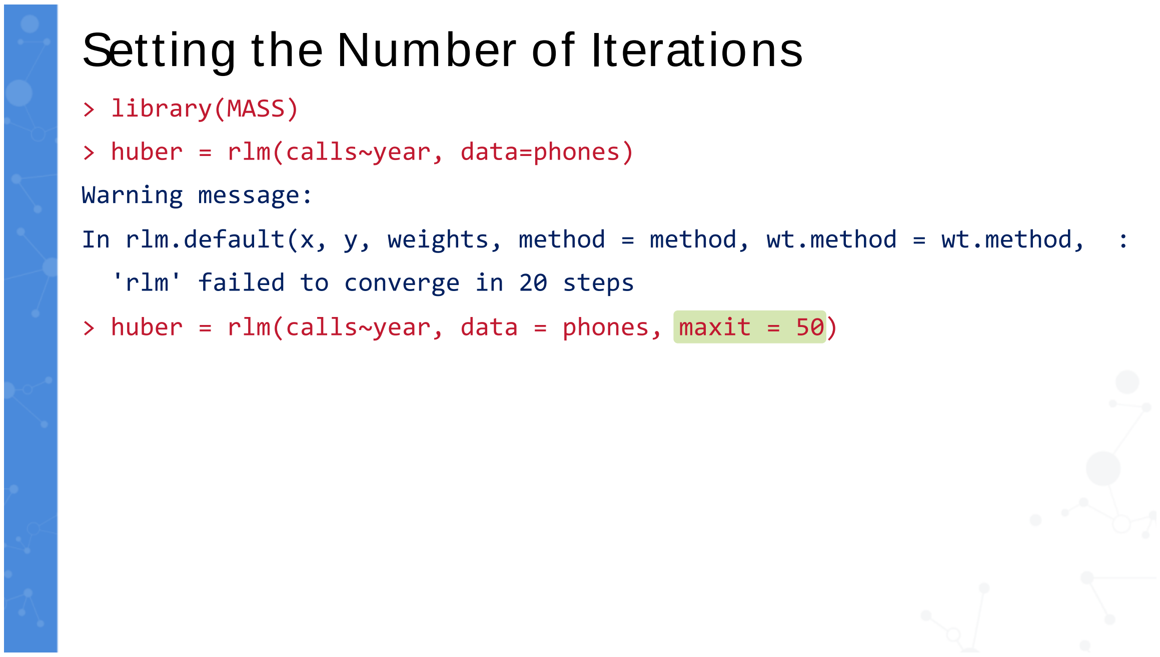


```
hills2 <- hills %>% tibble::rownames_to_column("Race")
hills2 <- hills2 %>%
  mutate(my_label = case_when(fit_bisquare$w < .8 ~ Race,
                              TRUE ~ ""))
gf_point(fit_bisquare$w ~ 1:dim(hills)[1]) %>%
  gf_text(fit_bisquare$w - .02 ~ 1:dim(hills)[1],
          label = hills2$my_label)
```

In this case, we see that the race Larry Grew, one of the points that looked like an outlier on the diagnostic plots from ordinary least squares regression, does not have a low weight in the Tukey's bi-square regression. This means it ended up being close to the regression line, whereas the race Two Breweries, which was not an outlier in ordinary least squares regression, does have a low weight in Tukey's bi-square regression, meaning it ended up being an outlier for this regression line.

Notes:

```
hills2 <- hills %>%
  tibble::rownames_to_column("Race")

hills2 <- hills2 %>%
  mutate(my_label = case_when(fit_bisquare$w < .8 ~ Race,
                              TRUE ~ ""))

gf_point(fit_bisquare$w ~ 1:dim(hills)[1]) %>%
  gf_text(fit_bisquare$w - .02 ~ 1:dim(hills)[1],
           label = hills2$my_label)
```



Setting the Number of Iterations

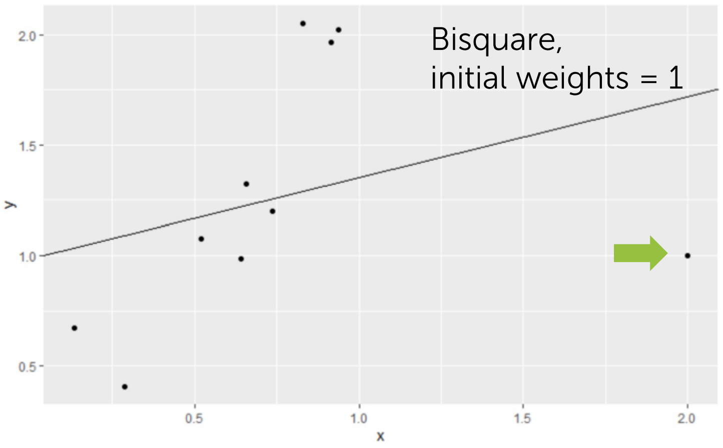
```
> library(MASS)
> huber = rlm(calls~year, data=phones)
Warning message:
In rlm.default(x, y, weights, method = method, wt.method = wt.method,   :
  'rlm' failed to converge in 20 steps
> huber = rlm(calls~year, data = phones, maxit = 50)
```

The Huber method is the default in rlm. So if you leave out the psi-- or p-s-i-- argument, that's what you'll get. Another default of rlm is to perform 20 iterations, but sometimes this isn't enough. In that case, you can set the number of iterations by setting the argument maxit from MaxIterations.

High-Leverage Points

Outliers with respect to *x*.

Check if model is reasonable.



High leverage points are points that are outliers with respect to a predictor variable. They tend to be very influential on the slope of a regression line, so they may not have a large residual with ordinary least squares regression. Both Huber and Tukey's bisquare methods tend to have a hard time with high leverage points, especially in small datasets such as the one shown here. So it's a good idea to plot your data and check if the model seems reasonable.

Notes:

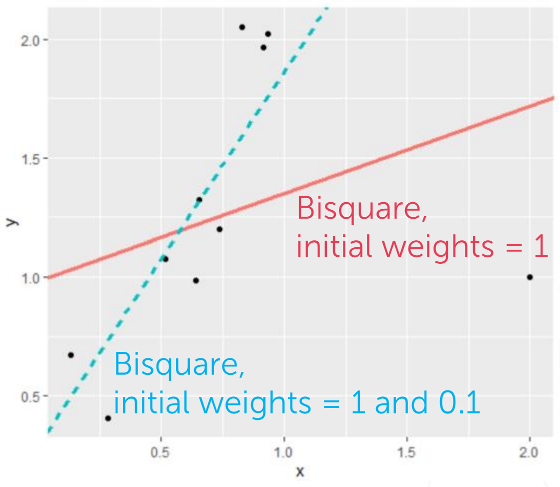
```
set.seed(42)
x=c(runif(9),2)
y=2*x+rnorm(10,0,.25)
y[10]=1

bisquare = rlm(y ~ x, psi = psi.bisquare)
gf_point(y ~ x) %>%
gf_abline(intercept = bisquare$coef[1], slope =
bisquare$coef[2])
```

Dealing with High-Leverage Points

Try different starting weights for Tukey's method:

```
fit = rlm(y~x, psi=psi.bisquare,
  weights=c(1,1,1,1,1,1,1,1,1,.1))
```

Try another method, such as least trimmed squares regression or least median of squares.



Tukey's method can find local minima. So it might be worth trying different starting weights either before the start of the while loop or in the weights argument of the RLM function. For example, in the data set shown here, if we give an initial weight of 0.1 to the high leverage point, we end up with the regression line shown in the dashed line here. This is a much better fit to the overall trend of the data than we got by using Tukey's bi-square method with all initial weights set equal to 1. If this still doesn't work, however, you can try another method of robust regression such as least trimmed squares, or least median of squares.

Notes:

For more about least trimmed squares regression, see
https://www.cs.umd.edu/~mount/Papers/lts-manuscript07.pdf

```
fit = rlm(y~x, psi=psi.bisquare,
  weights=c(1,1,1,1,1,1,1,1,1,.1))
```

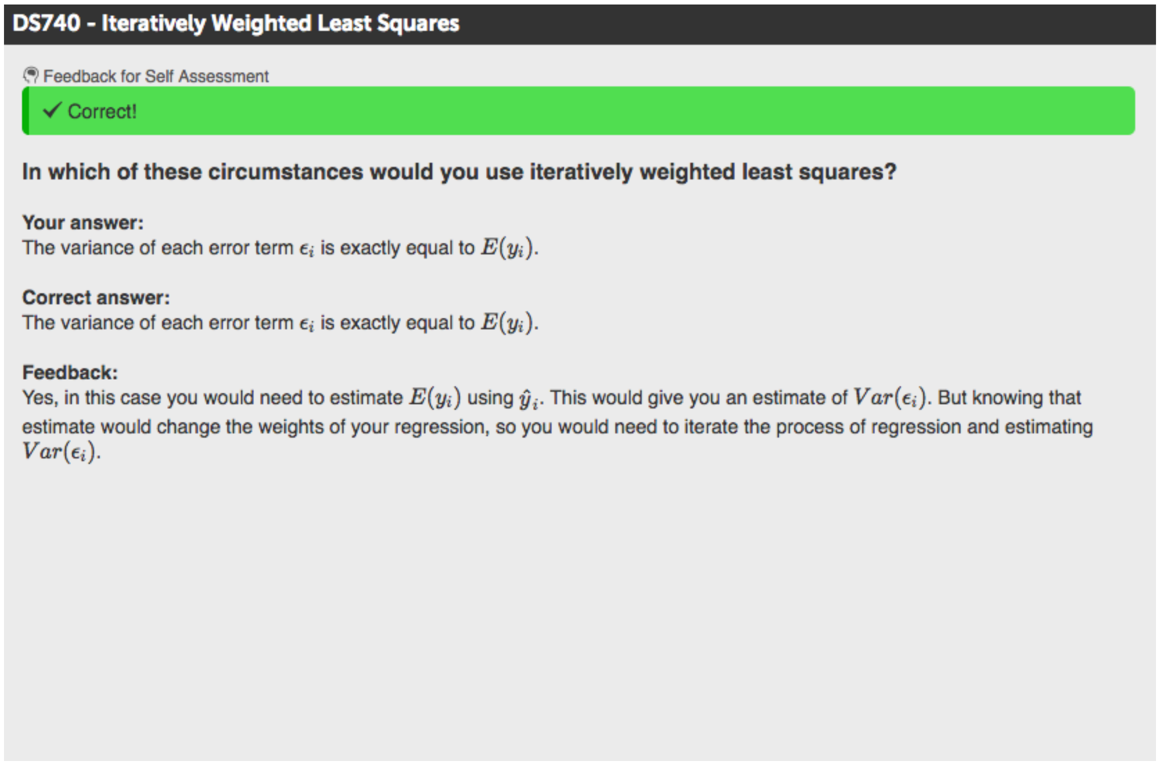
Summary: Weighted Least Squares

- When some data points are more reliable than others, use weighted regression with weights = 1/var(ϵ).
- When the *i*th data point's response value is a mean of n_i observations, use weights = n_i.
- In IWLS, we iterate the process of fitting a weighted regression model, estimating var(ϵ_i), and getting new weights.

Summary: Robust Regression

- Robust regression downweights data points with large absolute residuals, to reduce influence of outliers.
- Implemented via IWLS with Huber or Tukey’s bisquare weights.
- Tukey’s method downweights outliers more than Huber’s method.
- Choice of initial weights can influence final model with Tukey’s method, but not with Huber’s method (assuming convergence is achieved).

Question 1 answer



DS740 - Iteratively Weighted Least Squares

Feedback for Self Assessment

✓ Correct!

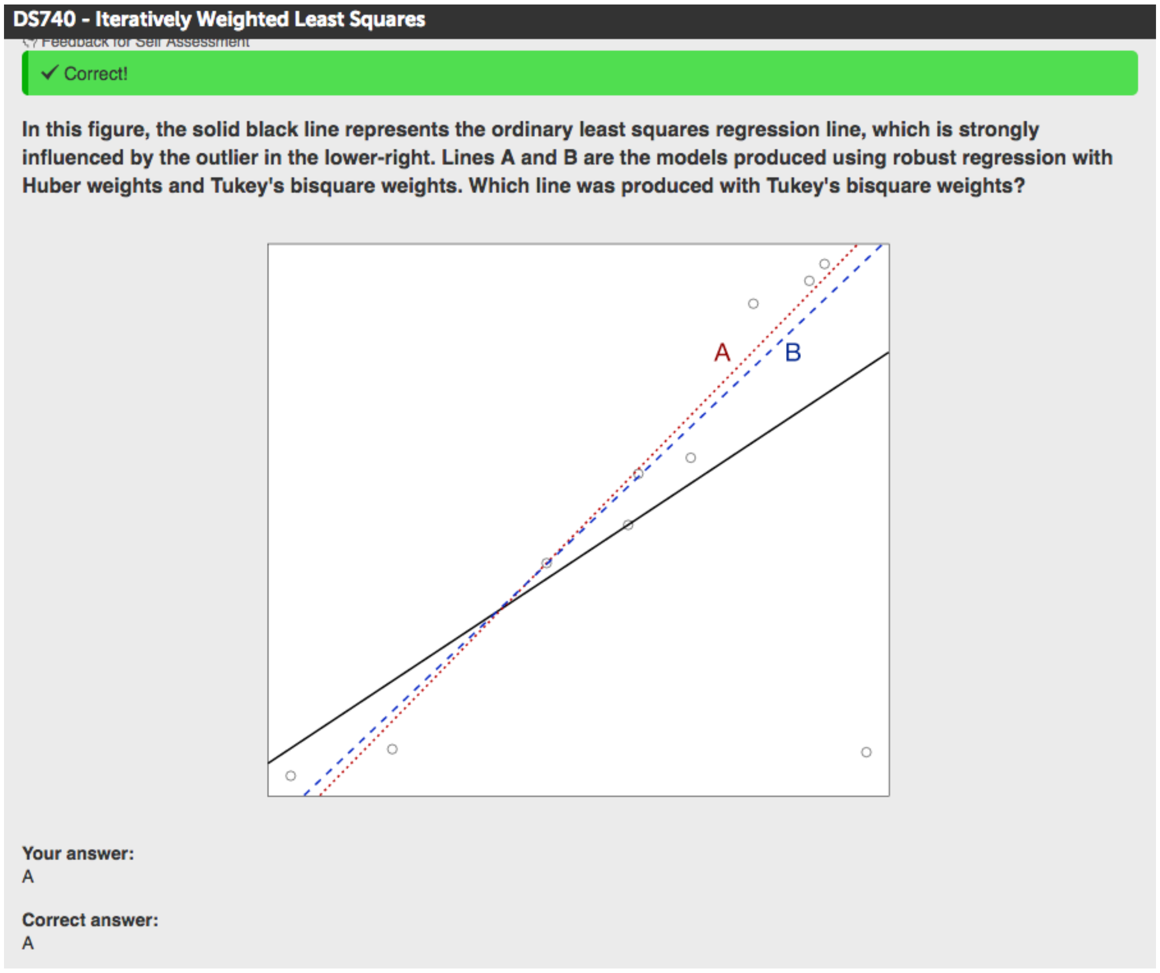
In which of these circumstances would you use iteratively weighted least squares?

Your answer:
The variance of each error term ϵ_i is exactly equal to $E(y_i)$.

Correct answer:
The variance of each error term ϵ_i is exactly equal to $E(y_i)$.

Feedback:
Yes, in this case you would need to estimate $E(y_i)$ using $\hat{y}_i$. This would give you an estimate of $Var(\epsilon_i)$. But knowing that estimate would change the weights of your regression, so you would need to iterate the process of regression and estimating $Var(\epsilon_i)$.

Question 2 answer

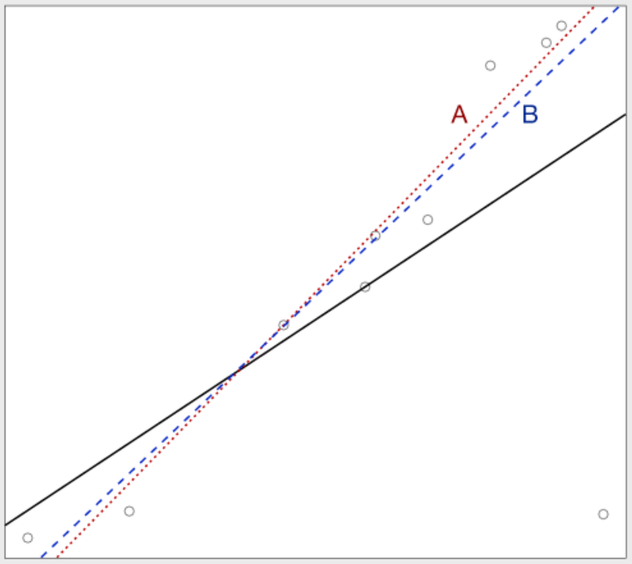


DS740 - Iteratively Weighted Least Squares

Feedback for Self Assessment

✓ Correct!

In this figure, the solid black line represents the ordinary least squares regression line, which is strongly influenced by the outlier in the lower-right. Lines A and B are the models produced using robust regression with Huber weights and Tukey's bisquare weights. Which line was produced with Tukey's bisquare weights?



Your answer:
A

Correct answer:
A