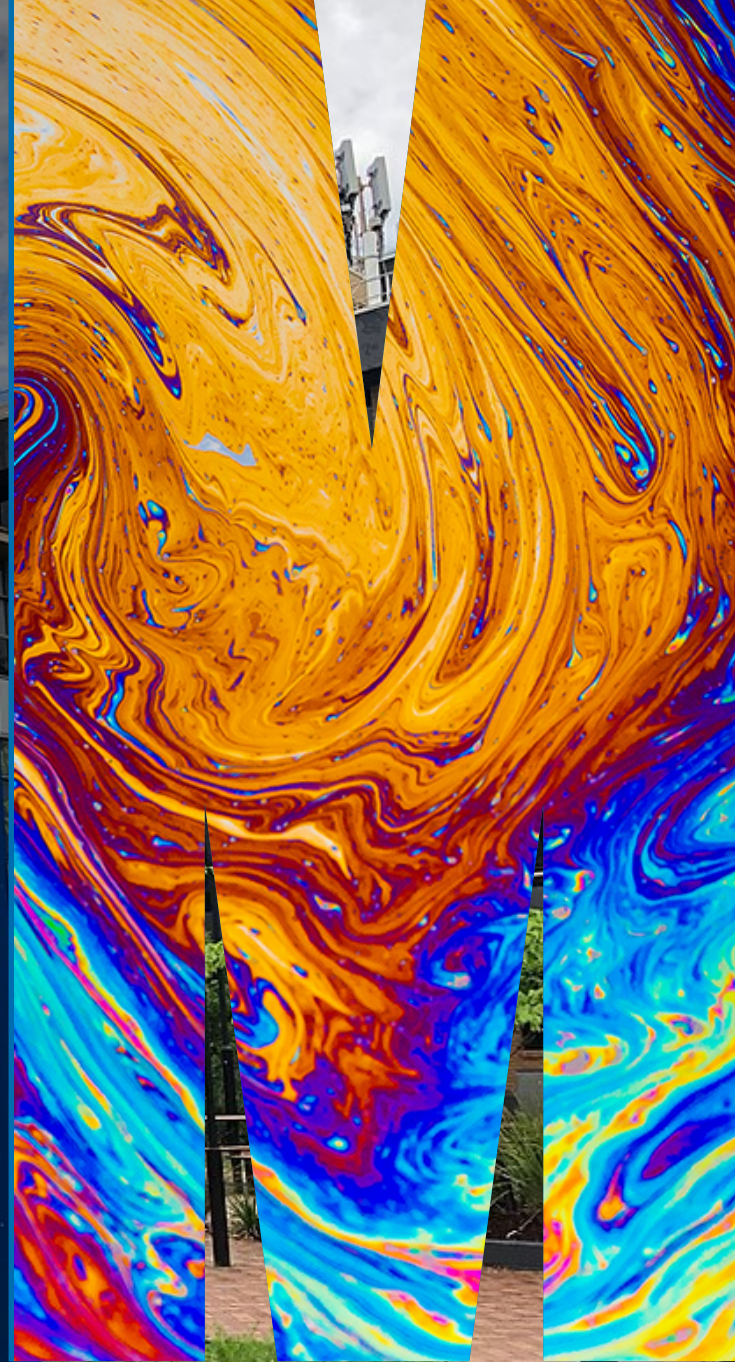
ETC5510: Introduction to Data Analysis

Week 10, part A

Regression and Decision Trees

Lecturer: *Nicholas Tierney & Stuart Lee*
Department of Econometrics and Business Statistics
nicholas.tierney@monash.edu
May 2020



recap

- networks

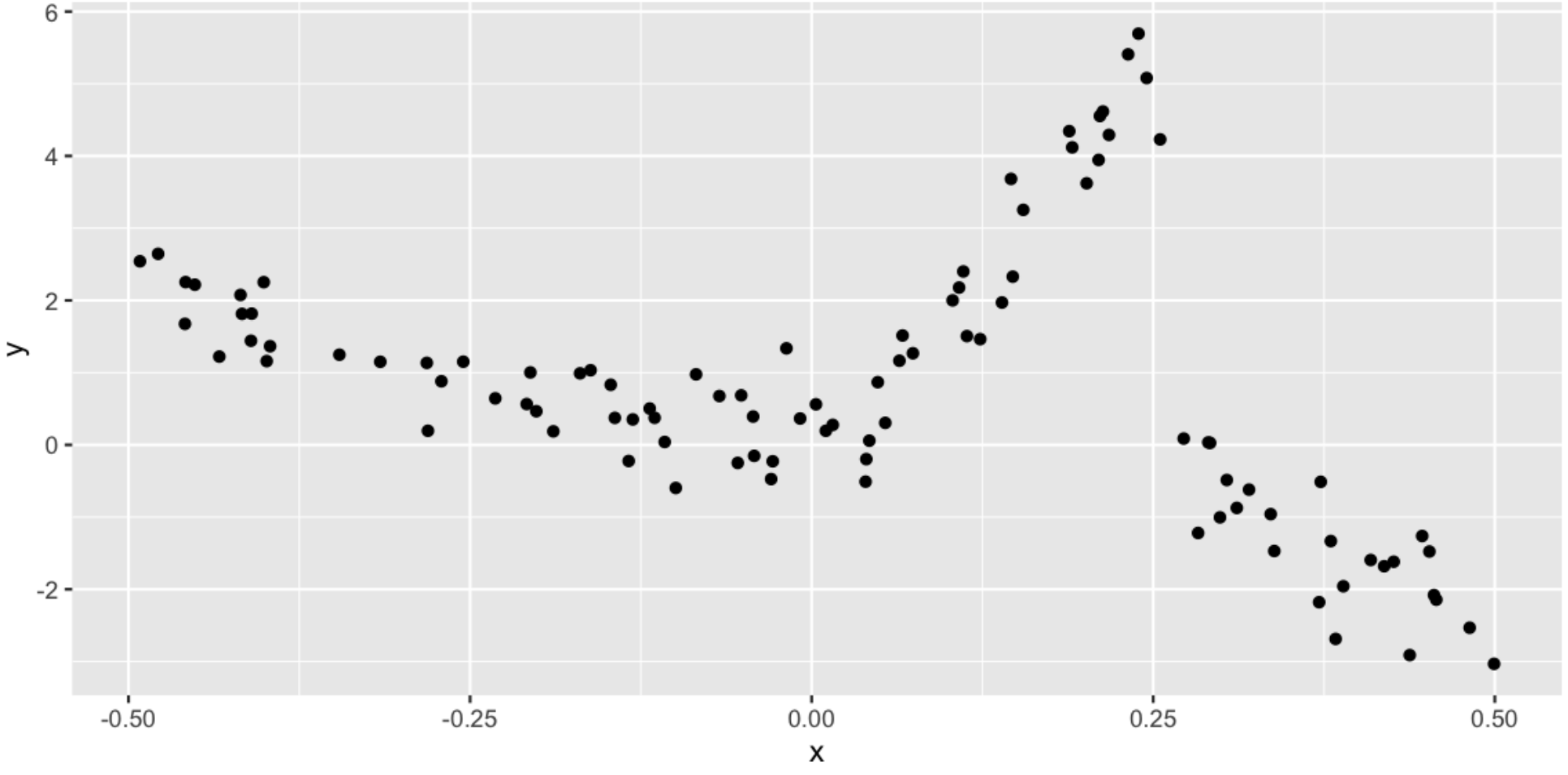
Upcoming

- Peer evaluation of assignment #2 on Wednesday
- Project milestone #3 due this Friday, if you need guidance attend consultation

Overview

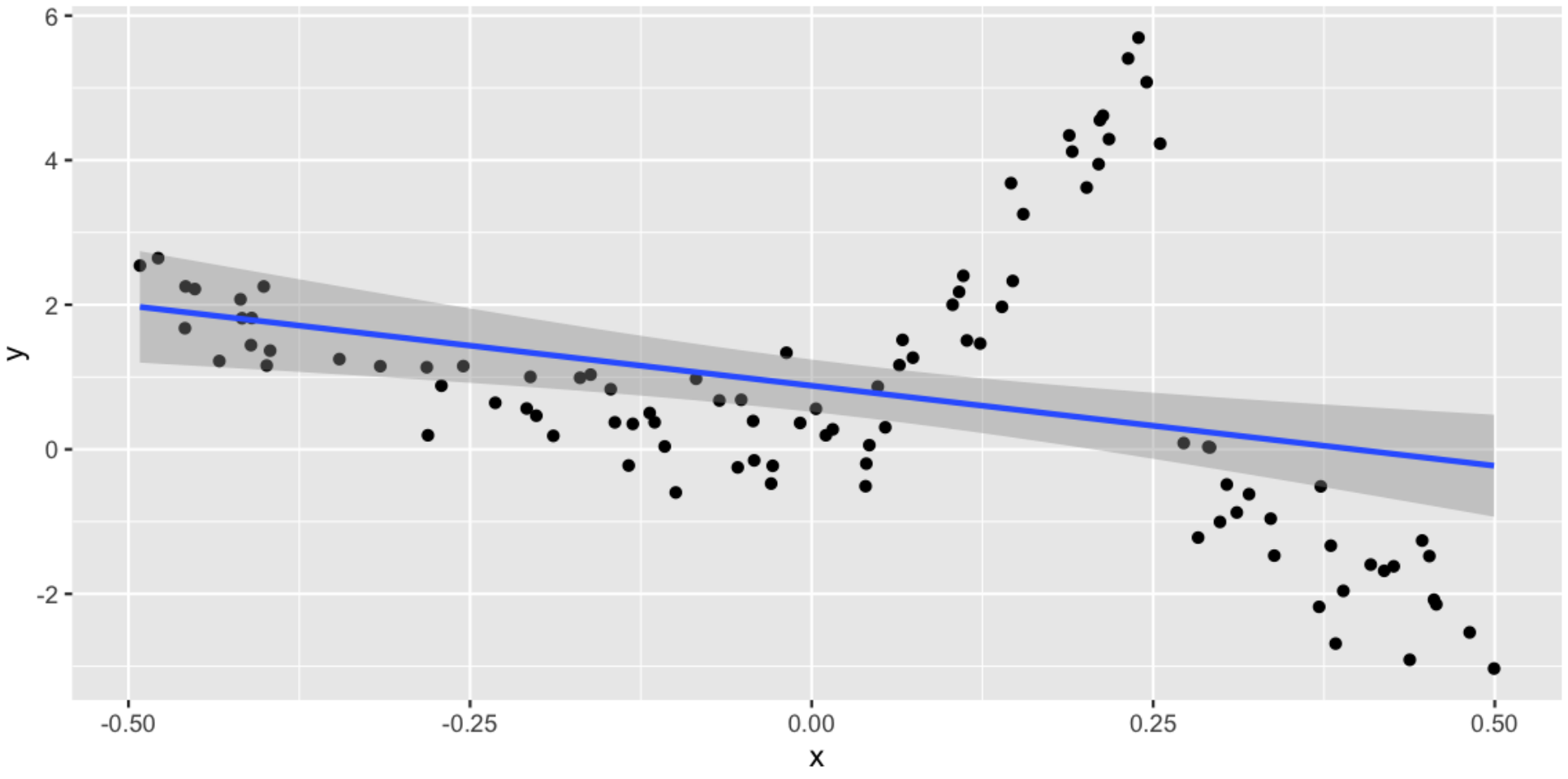
- What is a regression tree?
- How is it computed?
- Deciding when its a good fit
  - rmse
- Comparison with linear models
- Using multiple variables
- Next class:
  - How a classification tree differs from a regression tree?

Example



Let's predict Y using a linear model

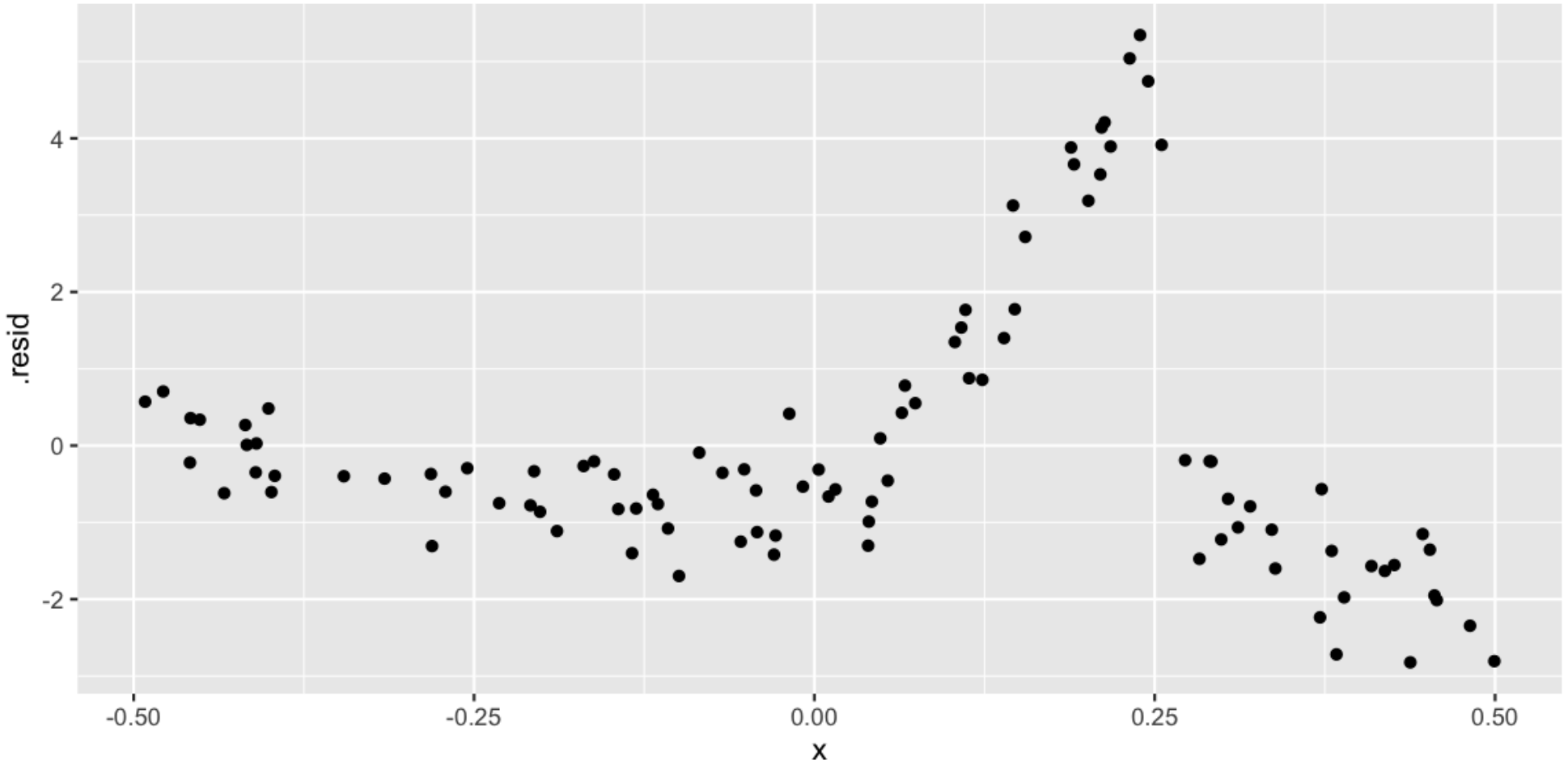
```
df_lm <- lm(y ~ x, df)
```



Assessing model fit

- Look at residuals
- Look at mean square error

Looking at the residuals: this is bad!



It basically looks like the data!

Looking at the Mean square error (MSE)

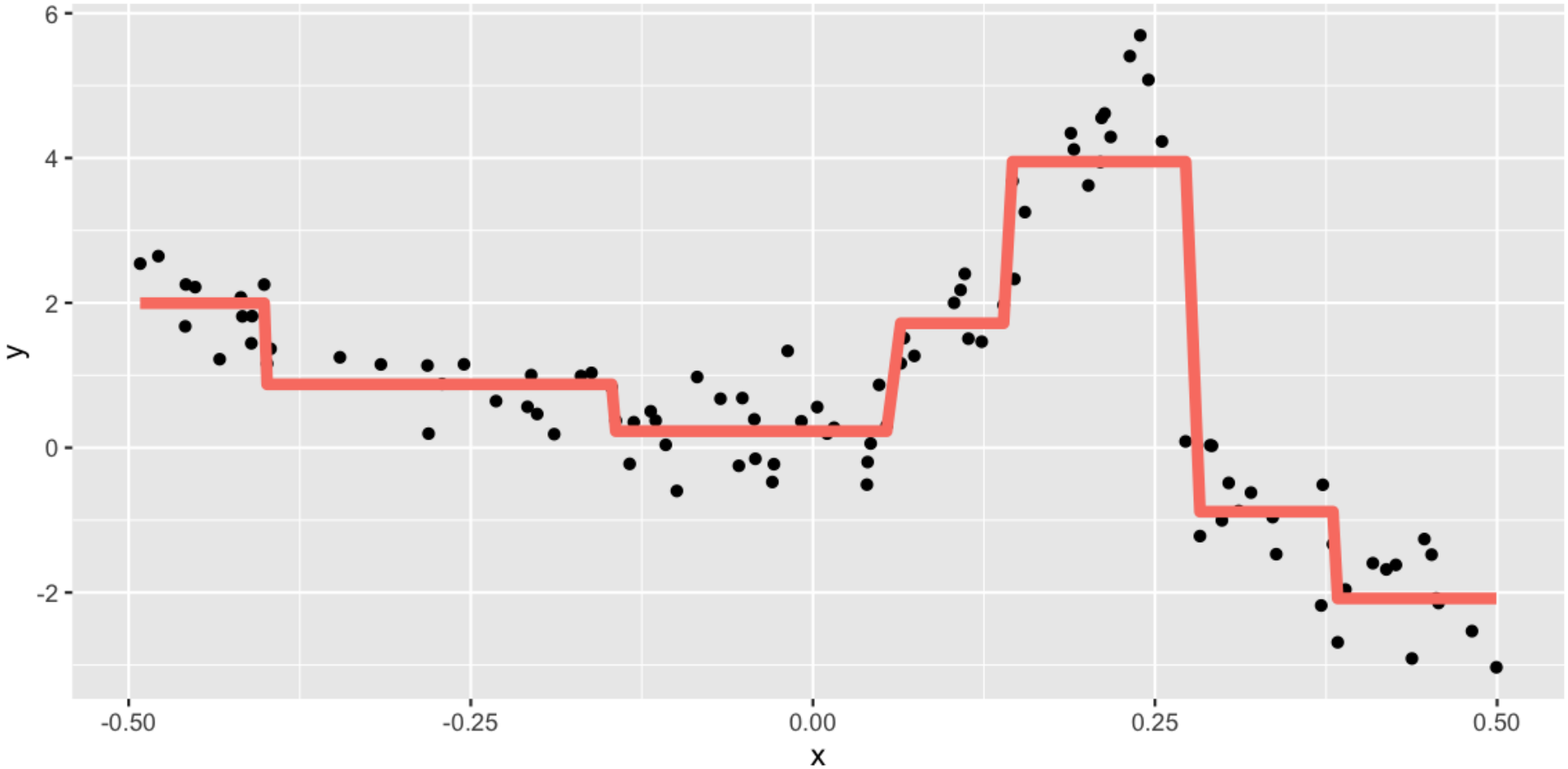
$$MSE(y) = \frac{\sum_{i=1}^{i=N}(y_i - \hat{y}_i)^2}{N}$$

In R code:

```
# calculate example of linear model MSE
```

Let's use a different model: "rpart"

```
library(rpart)
# df_lm <- lm(y~x, data=df) - similar to lm! But rpart.
df_rp <- rpart(y~x, data=df)
```

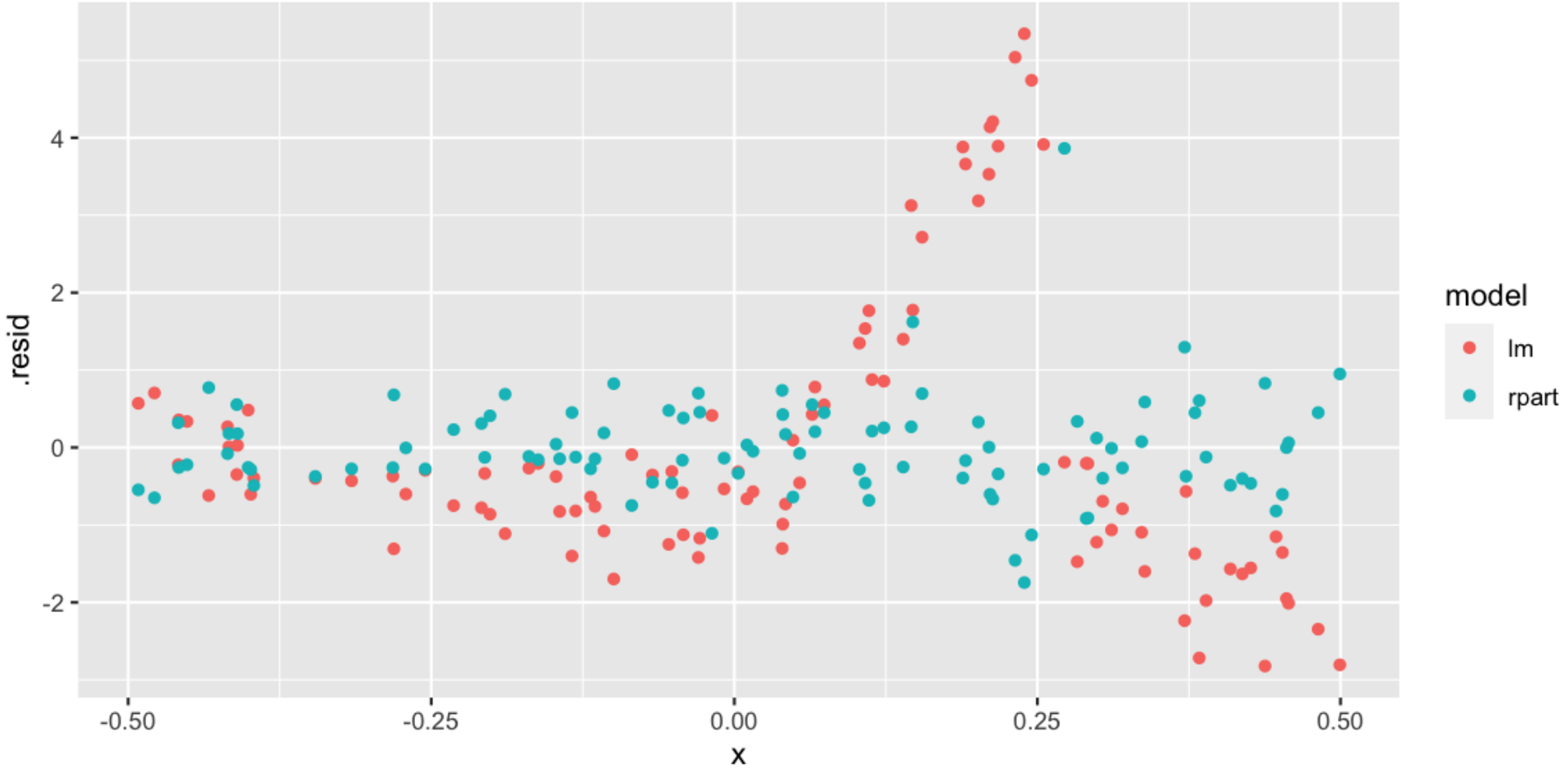


Look at residuals

```
ggplot(df_rp_aug,
       aes(x = x,
           y = y)) +
  geom_point() +
  geom_line(aes(y = .fitted), colour = "salmon", size = 2)
```

Look at MSE

Comparing lm vs rpart



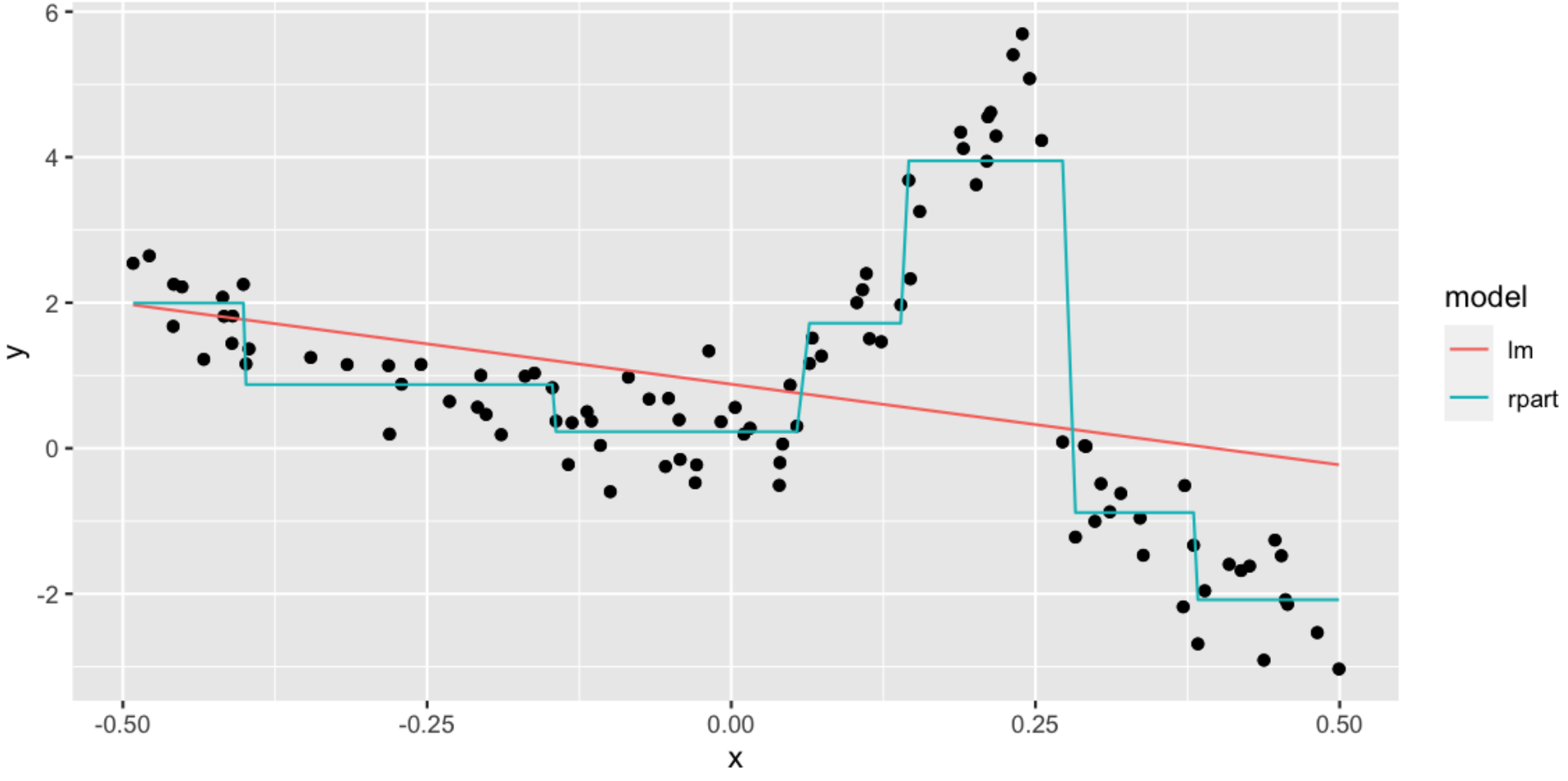
What the output of a linear model looks like

```
##
## Call:
## lm(formula = y ~ x, data = df)
##
## Coefficients:
## (Intercept)            x
##      0.8806      -2.2165
```

What the output of a rpart looks like

```
## n= 100
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 100 359.245100  0.8081071
##    2) x>=0.2775916 24  16.840100 -1.4822830
##      4) x>=0.3817438 12   3.832238 -2.0814410 *
##      5) x< 0.3817438 12   4.392090 -0.8831252 *
##    3) x< 0.2775916 76 176.745400  1.5313880
##      6) x< 0.1426085 61  41.562800  0.9365995
##       12) x>=-0.3999242 50  24.519860  0.7035330
##         24) x< 0.05905847 41  11.729940  0.4807175
##           48) x>=-0.1455513 25   5.653876  0.2281914 *
##           49) x< -0.1455513 16   1.990829  0.8752895 *
##         25) x>=0.05905847 9   1.481498  1.7185820 *
##       13) x< -0.3999242 11   1.981477  1.9959930 *
##      7) x>=0.1426085 15  25.842970  3.9501960 *
```

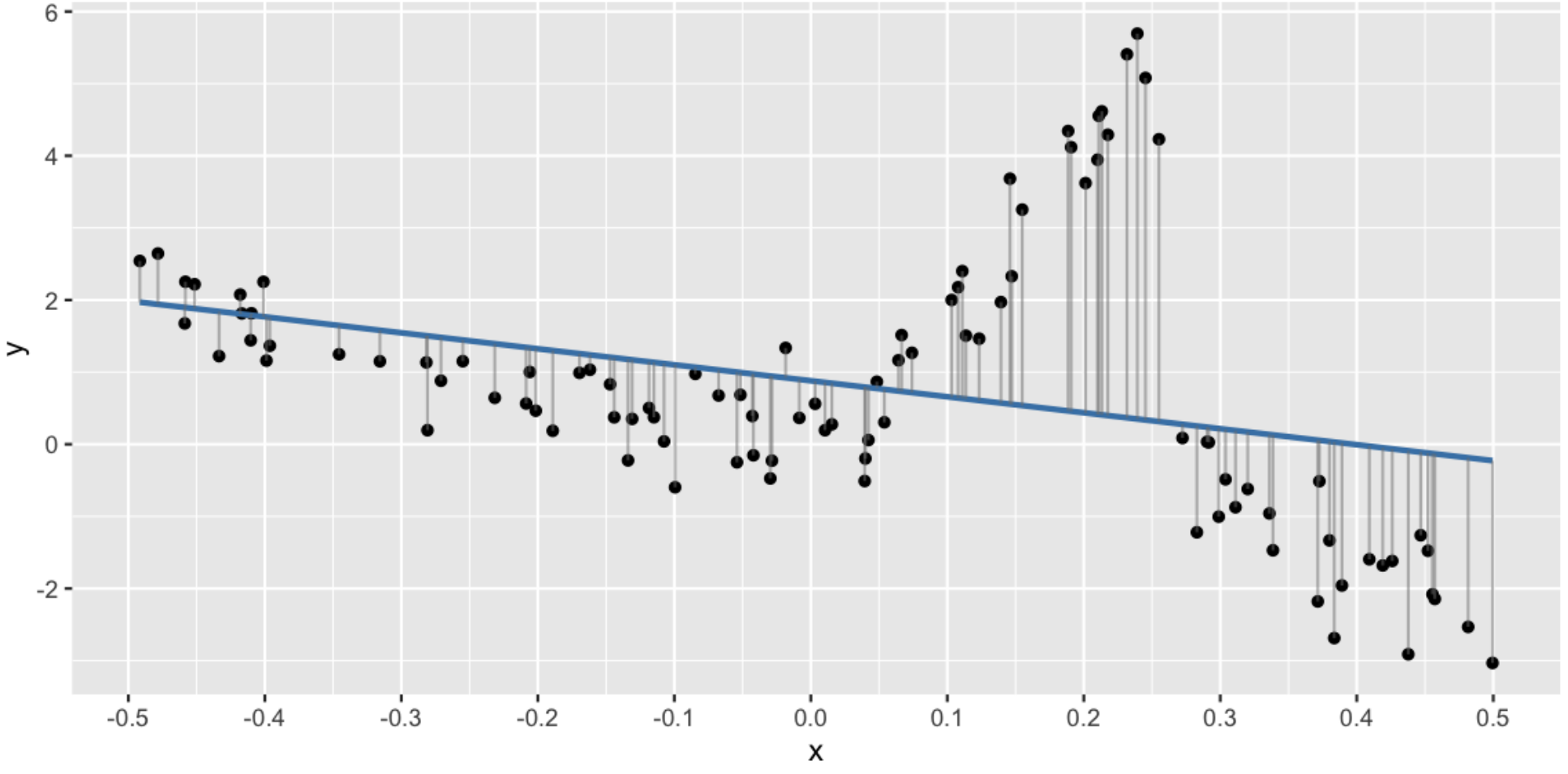
Predictions from linear model vs rpart



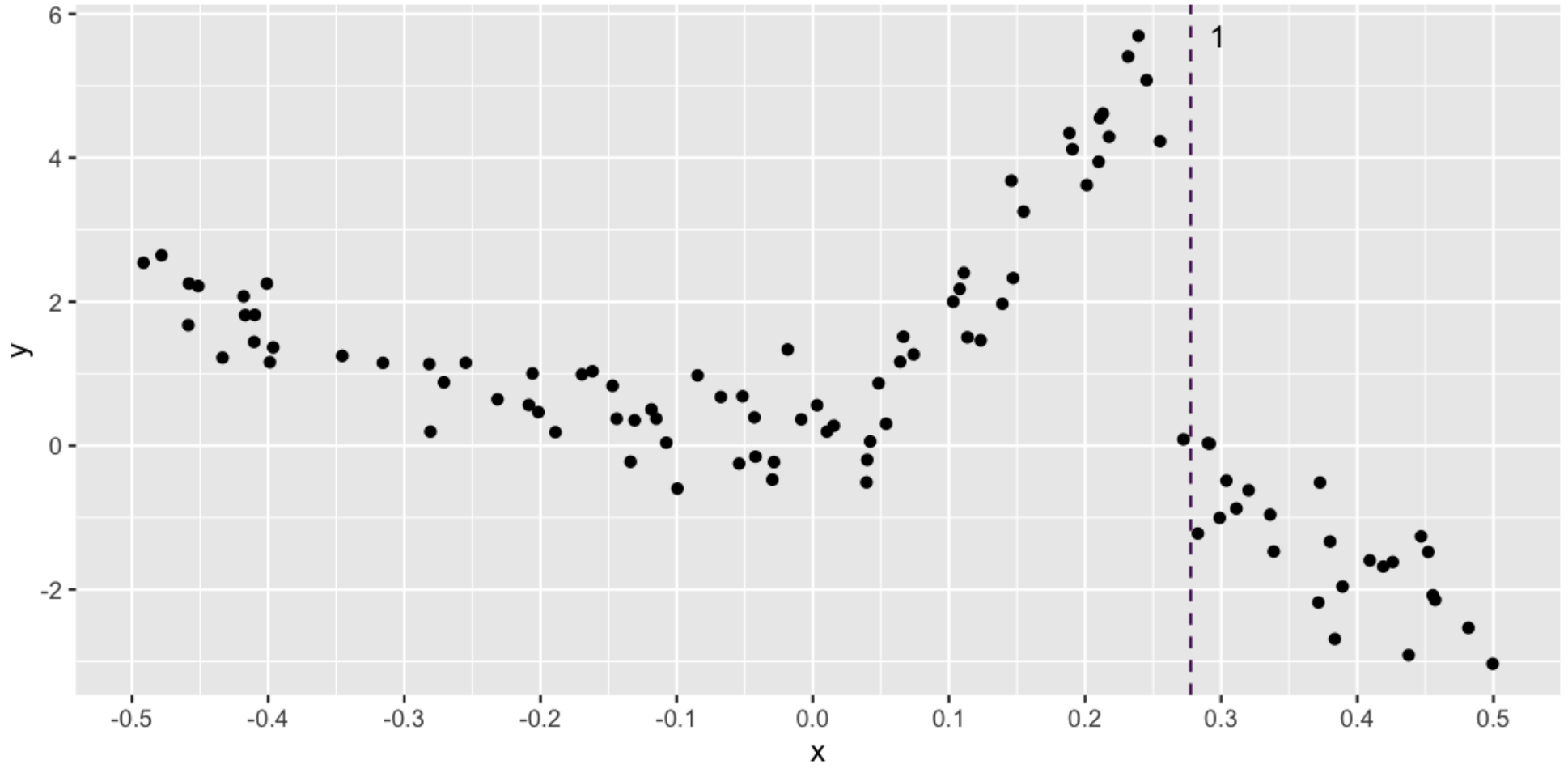
So what is going on?

- A linear model asks "What line fits through these points, to minimise the error"?
- A decision tree model asks "How can I best break the data into segments, to minimize some error?"

A linear model: draws the line of best fit



A regression tree: segments the data to reduce some error



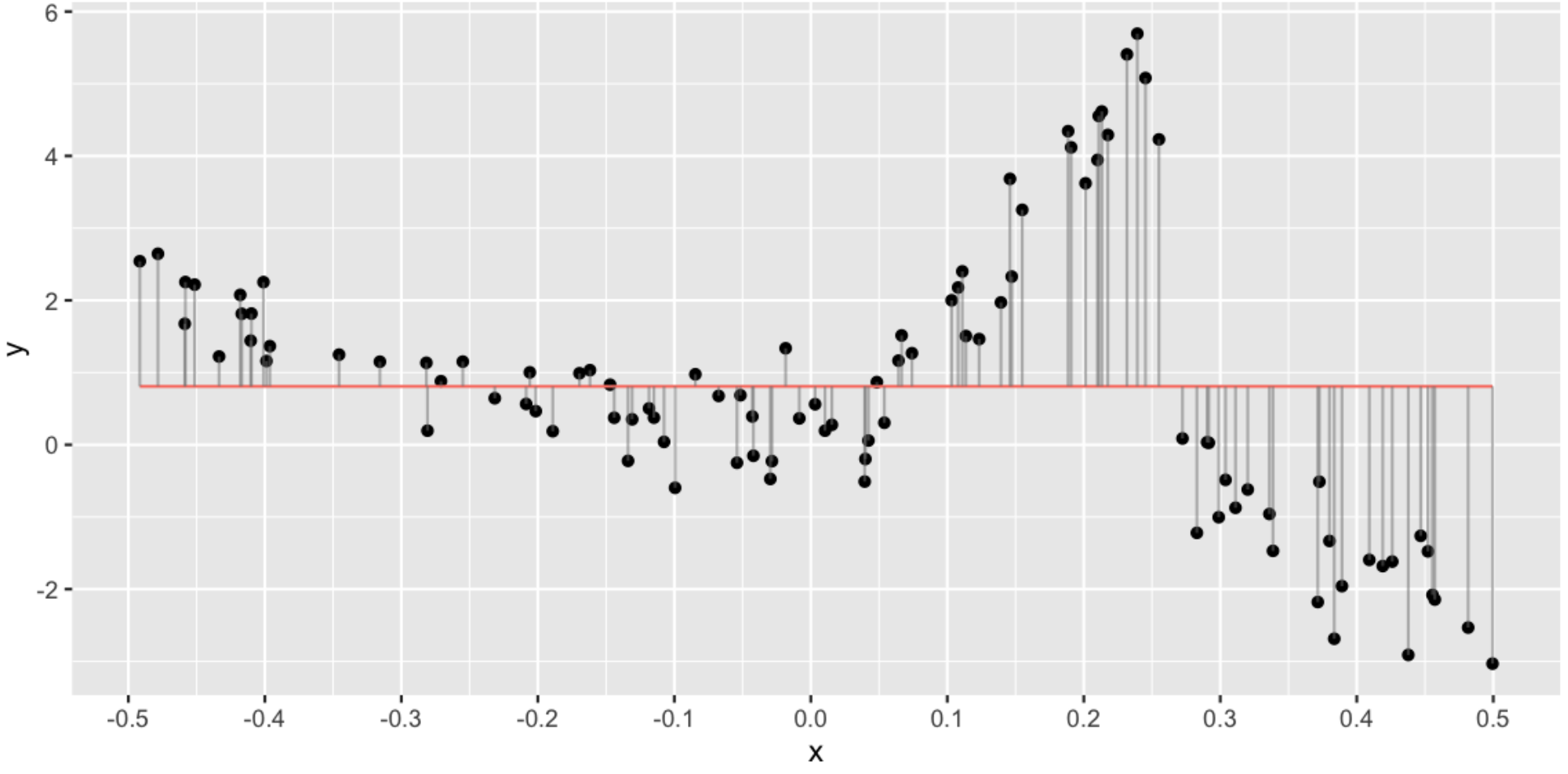
Regression trees

- Regression trees recursively partition the data, and use the average response value of each partition as the model estimate
- It is a computationally intense technique that examines all possible partitions, and choosing the BEST partition by optimizing some criteria
- For regression, with a quantitative response variable, the criteria is called ANOVA:

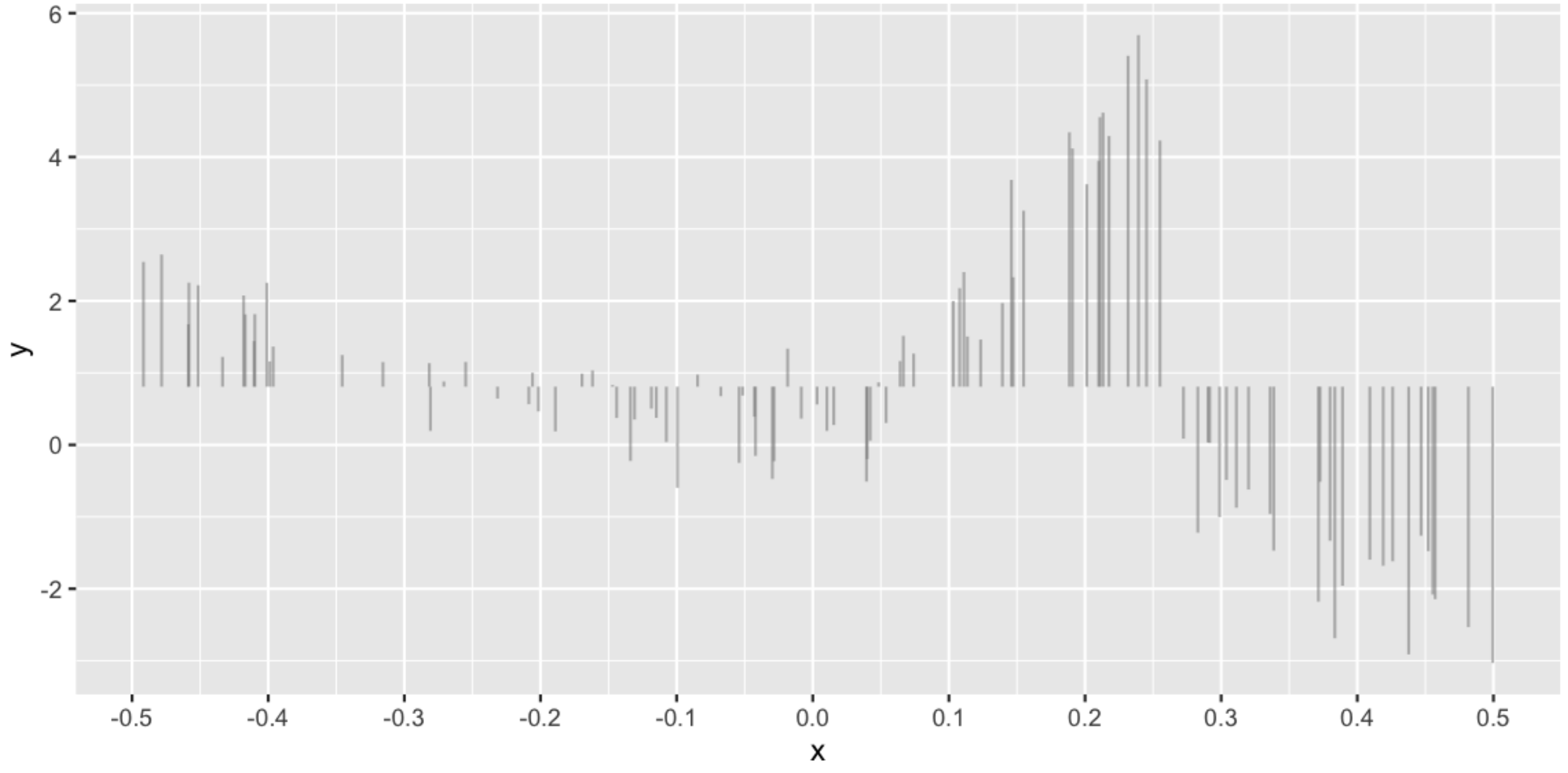
$$SS_T - (SS_L + SS_R)$$

where $SS_T = \sum(y_i - \bar{y})^2$, and SS_L, SS_R are the equivalent values for the two subsets created by partitioning.

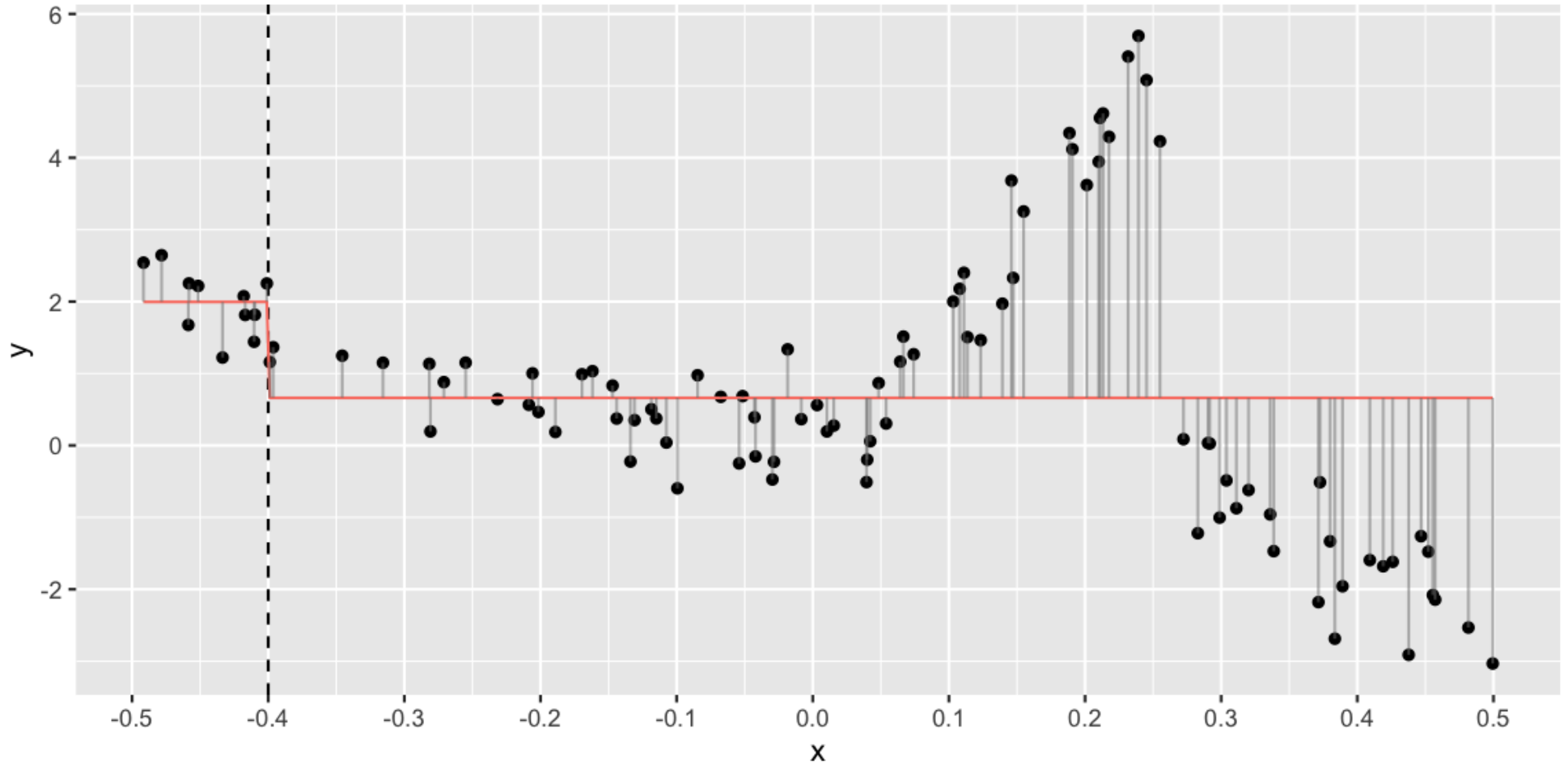
Break down: What is $SS_T = \sum(y_i - \bar{y})^2$?



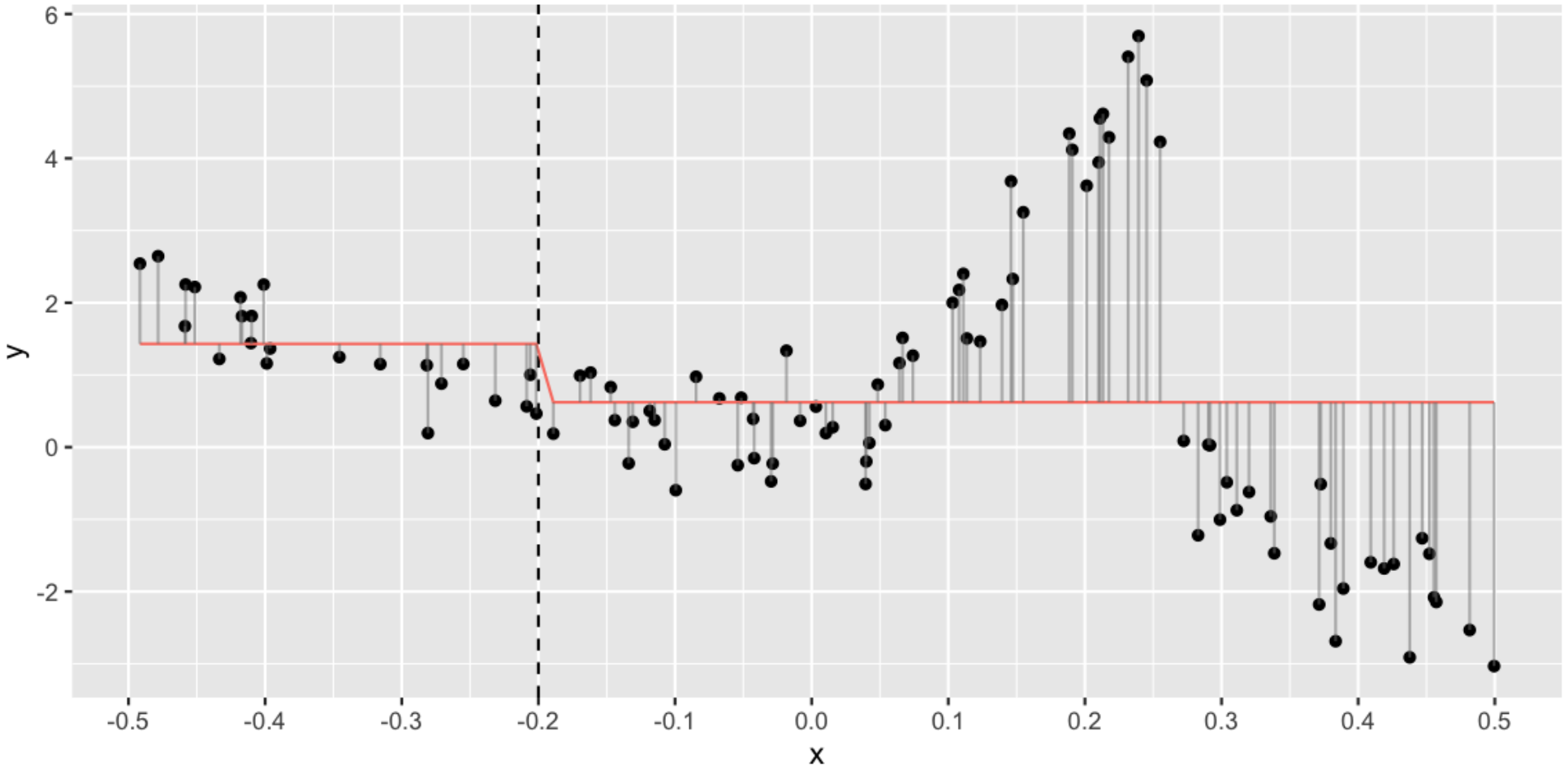
Break down: What is $SS_T = \sum(y_i - \bar{y})^2$?



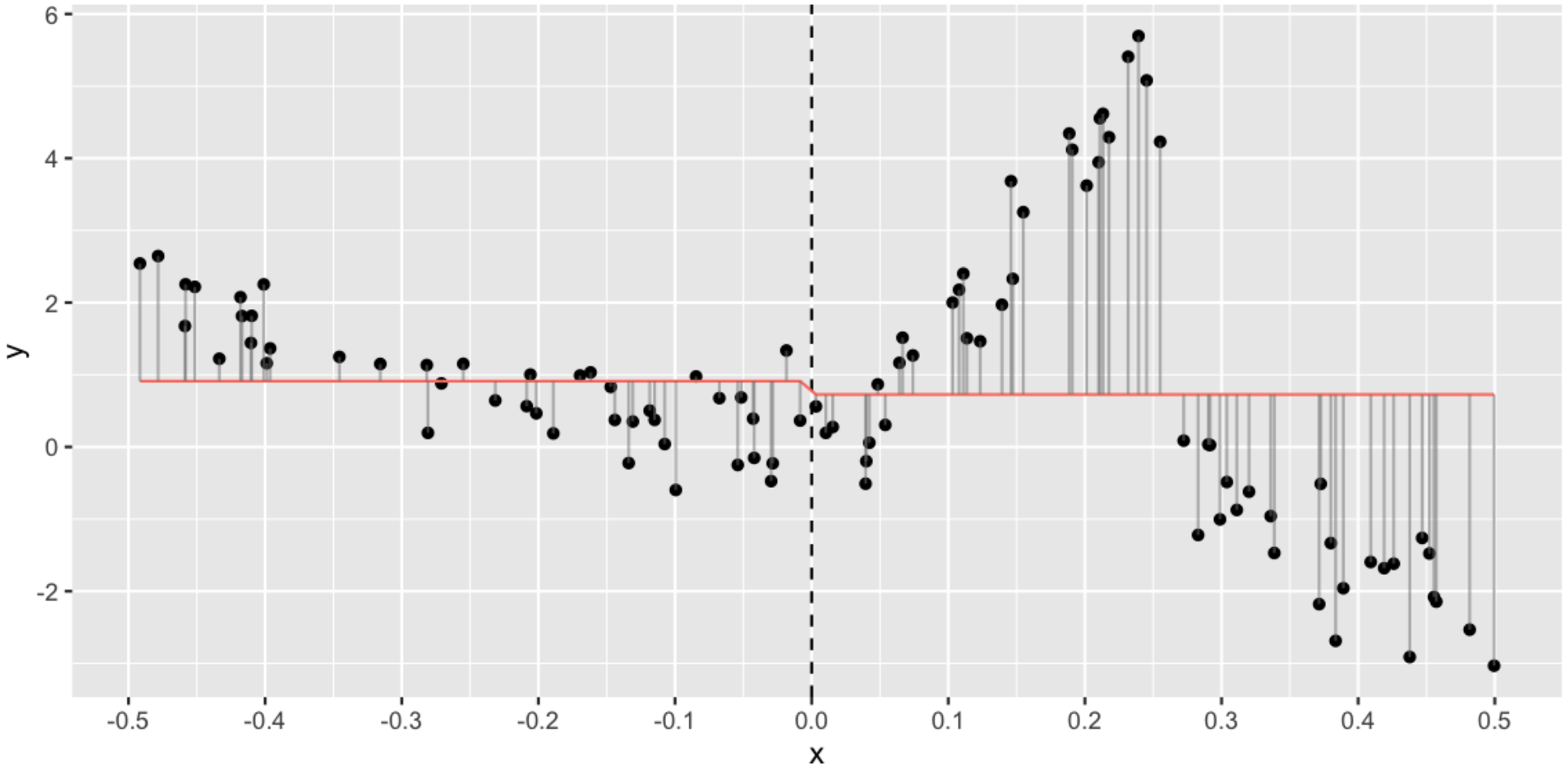
SS_L SS_R? Choose a point, compare the left and right



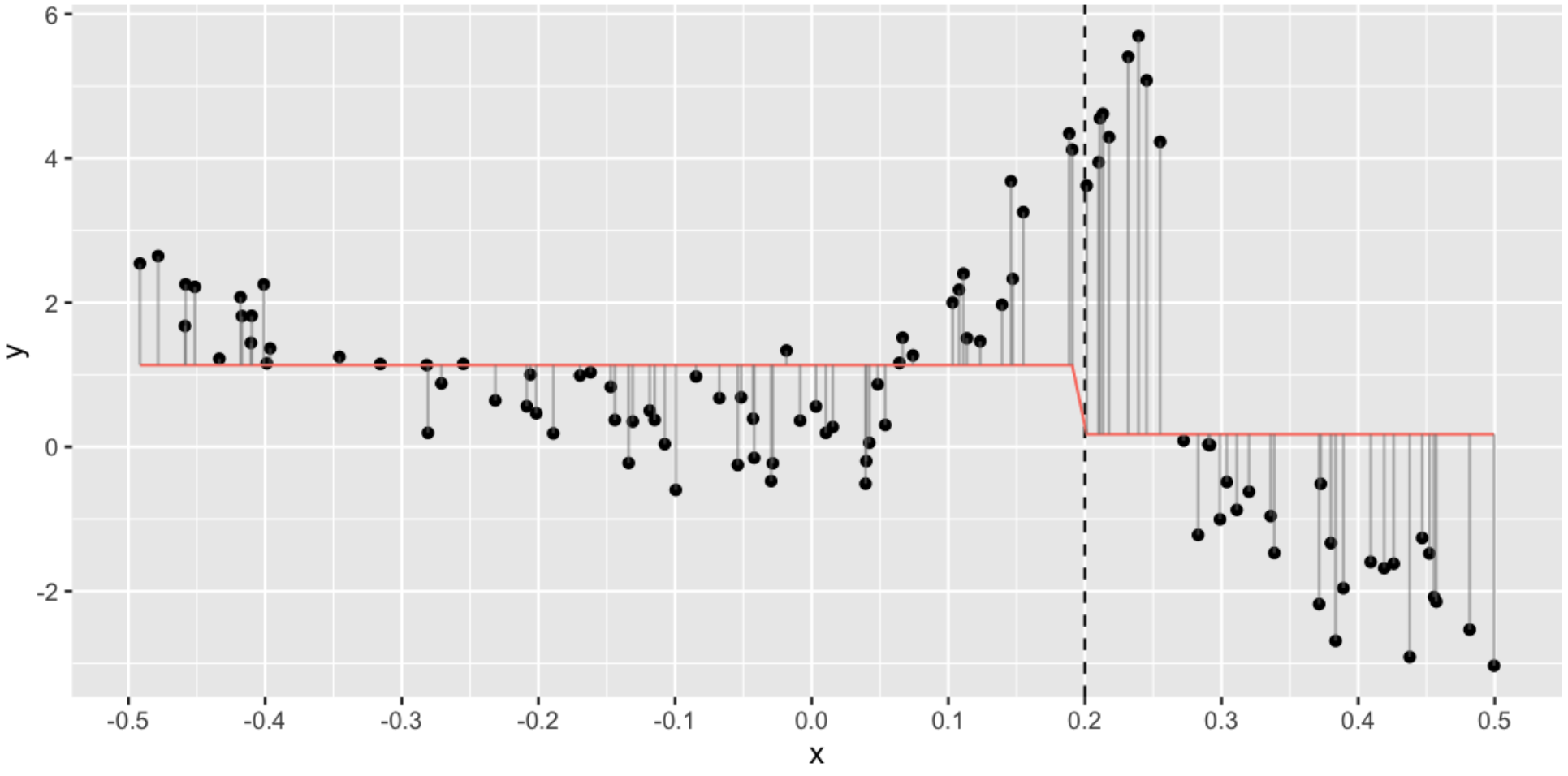
SS_L SS_R? Choose a point, compare the left and right



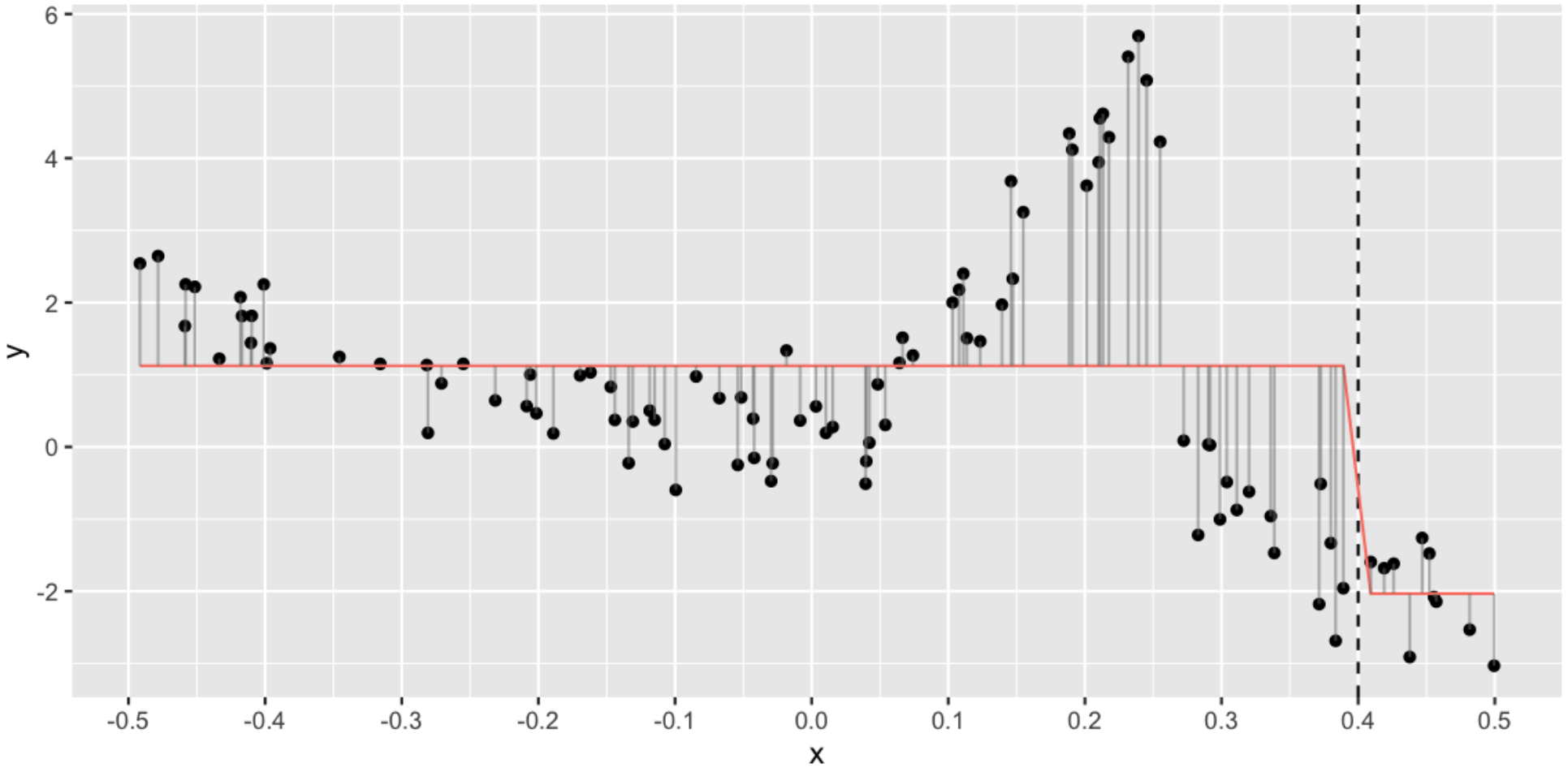
SS_L SS_R? Choose a point, compare the left and right



SS_L SS_R? Choose a point, compare the left and right



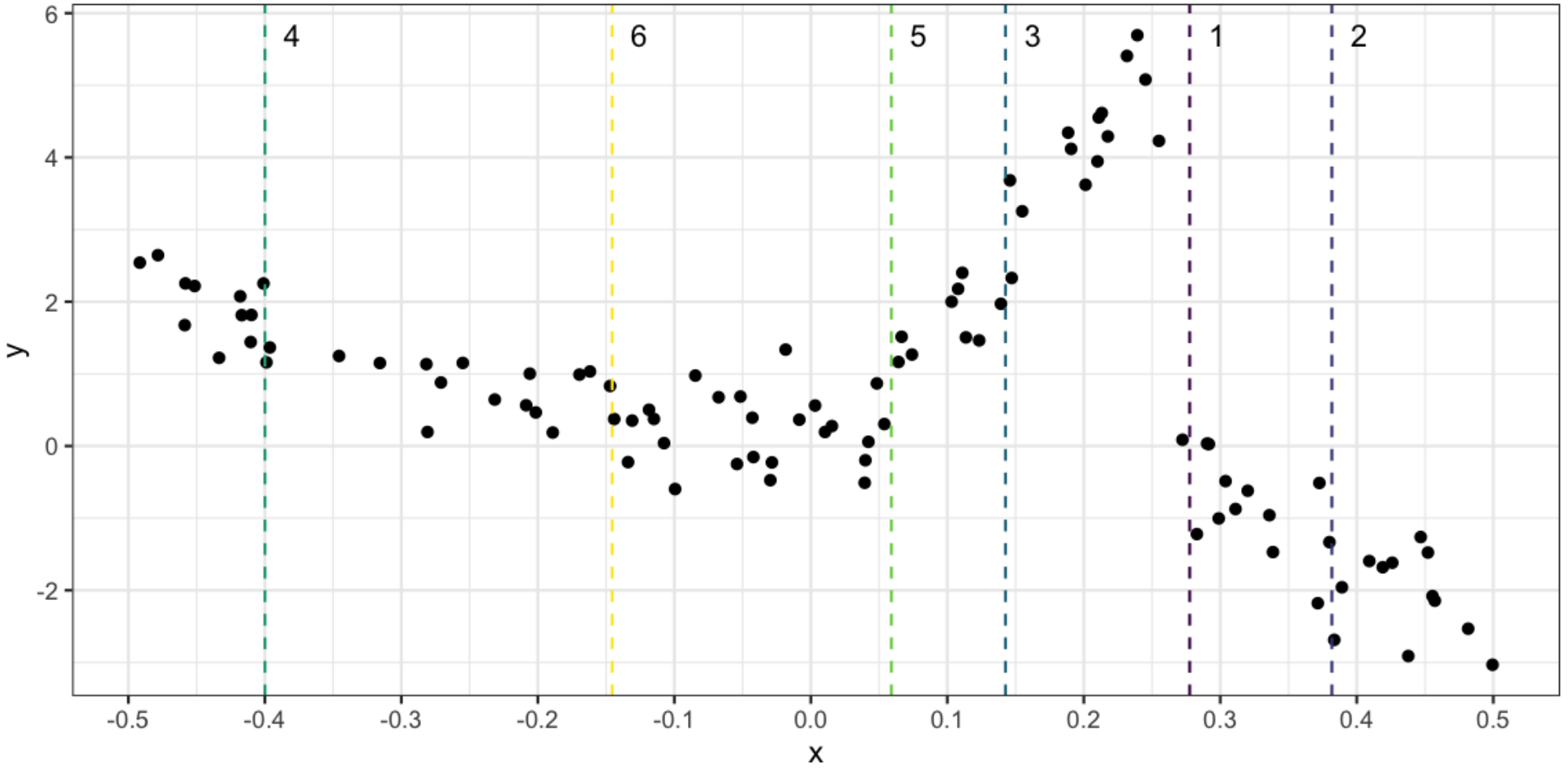
SS_L SS_R? Choose a point, compare the left and right



Across all values of x?

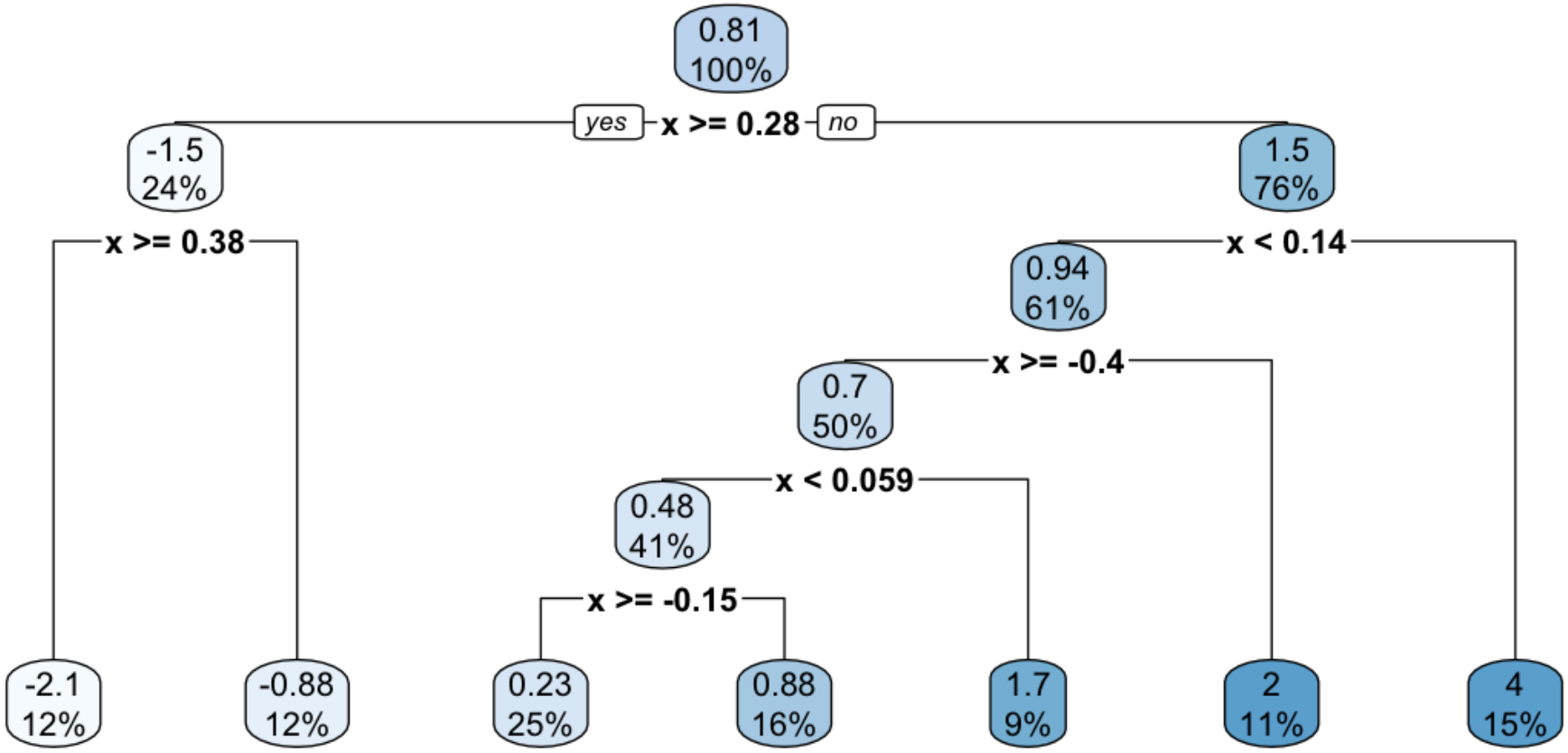
And if we repeated this again

This is how the data is split:



We can represent these splits in a tree format:

```
library(rpart.plot)
rpart.plot(df_rp)
```



Your turn: compute a regression tree

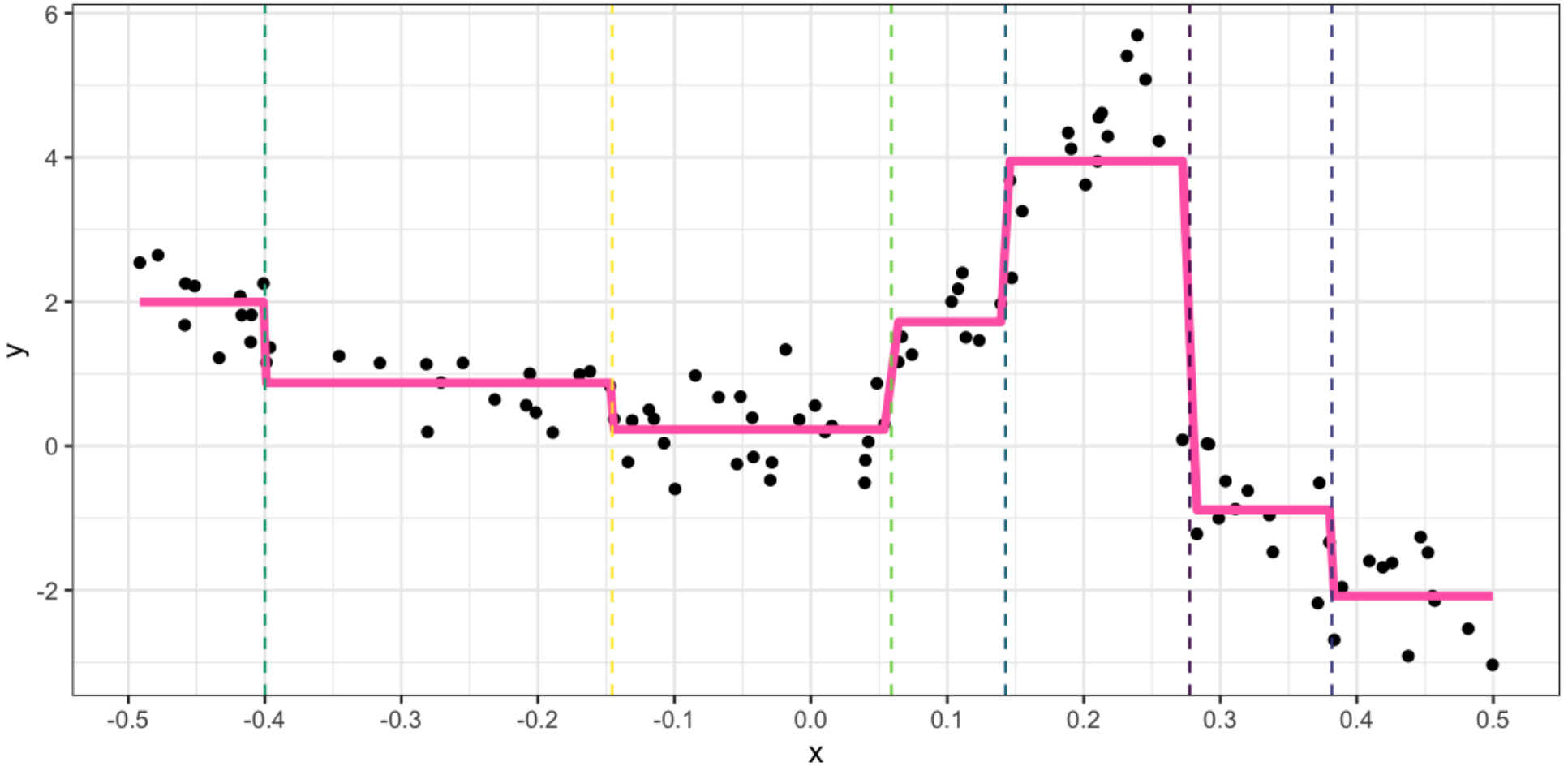
Using the small data set, manually compute a regression tree model for the data. Sketch the model.

```
d <- tibble(x=c(1, 2, 3, 4, 5), y=c(10, 12, 5, 4, 3))
d
ggplot(d, aes(x=x, y=y)) +
  geom_???()
```

Understanding rpart

```
df_rp
## n= 100
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 100 359.245100  0.8081071
##    2) x>=0.2775916 24  16.840100 -1.4822830
##      4) x>=0.3817438 12   3.832238 -2.0814410 *
##      5) x< 0.3817438 12   4.392090 -0.8831252 *
##    3) x< 0.2775916 76 176.745400  1.5313880
##      6) x< 0.1426085 61  41.562800  0.9365995
##       12) x>=-0.3999242 50  24.519860  0.7035330
##         24) x< 0.05905847 41  11.729940  0.4807175
##           48) x>=-0.1455513 25   5.653876  0.2281914 *
##           49) x< -0.1455513 16   1.990829  0.8752895 *
##         25) x>=0.05905847 9   1.481498  1.7185820 *
##       13) x< -0.3999242 11   1.981477  1.9959930 *
##      7) x>=0.1426085 15  25.842970  3.9501960 *
```

This is how the model looks:



Stopping rules

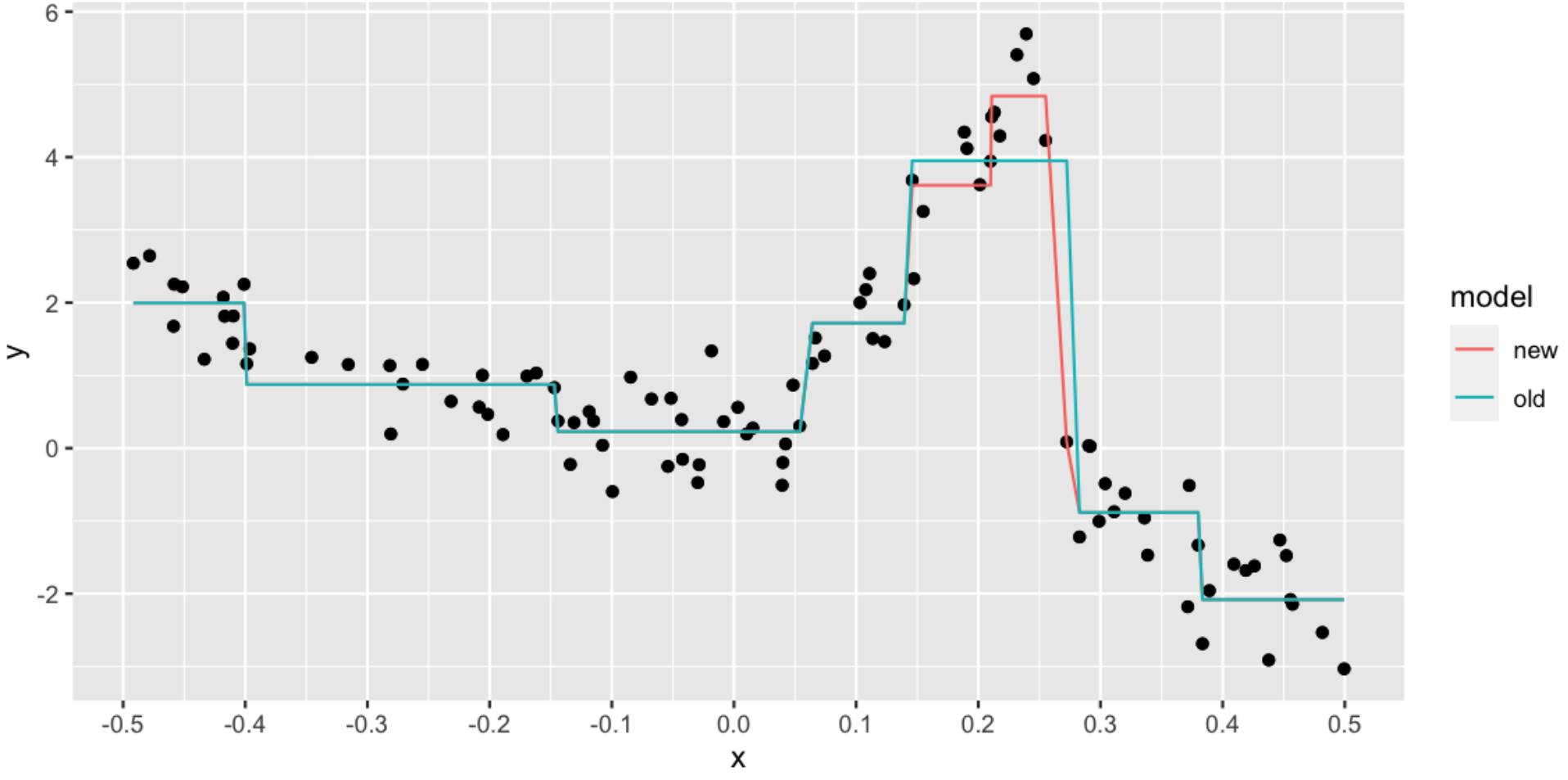
- Its an algorithm. Why did it stop at 7 terminal nodes?
- Stopping rules are needed, else the algorithm will keep fitting until every observation is in its own group.
- Control parameters set stopping points:
  - minsplit: minimum number of points in a node that algorithm is allowed to split
  - minbucket: minimum number of points in a terminal node
- We can also look at the change in value of $SS_T - (SS_L + SS_R)$ at each split, and if the change is too *small*, stop.

You can change the options to fit a different model

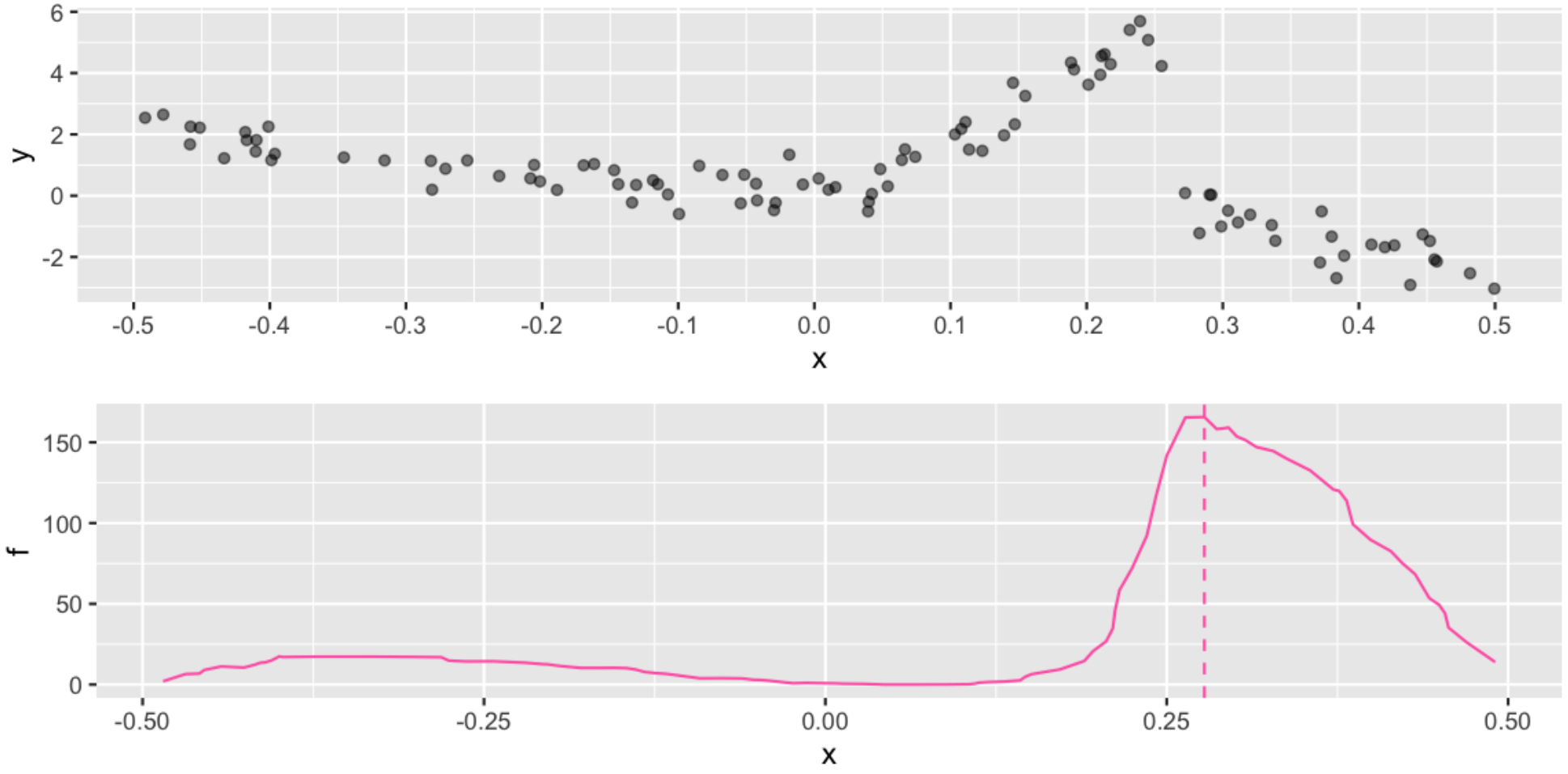
An re-fit, the model will change. Here we reduce the `minbucket` parameter.

```
df_rp_m10 <- rpart(y~x, data=df,
                   control = rpart.control(minsplit = 2))
```

This yields a (slightly) more complex model.

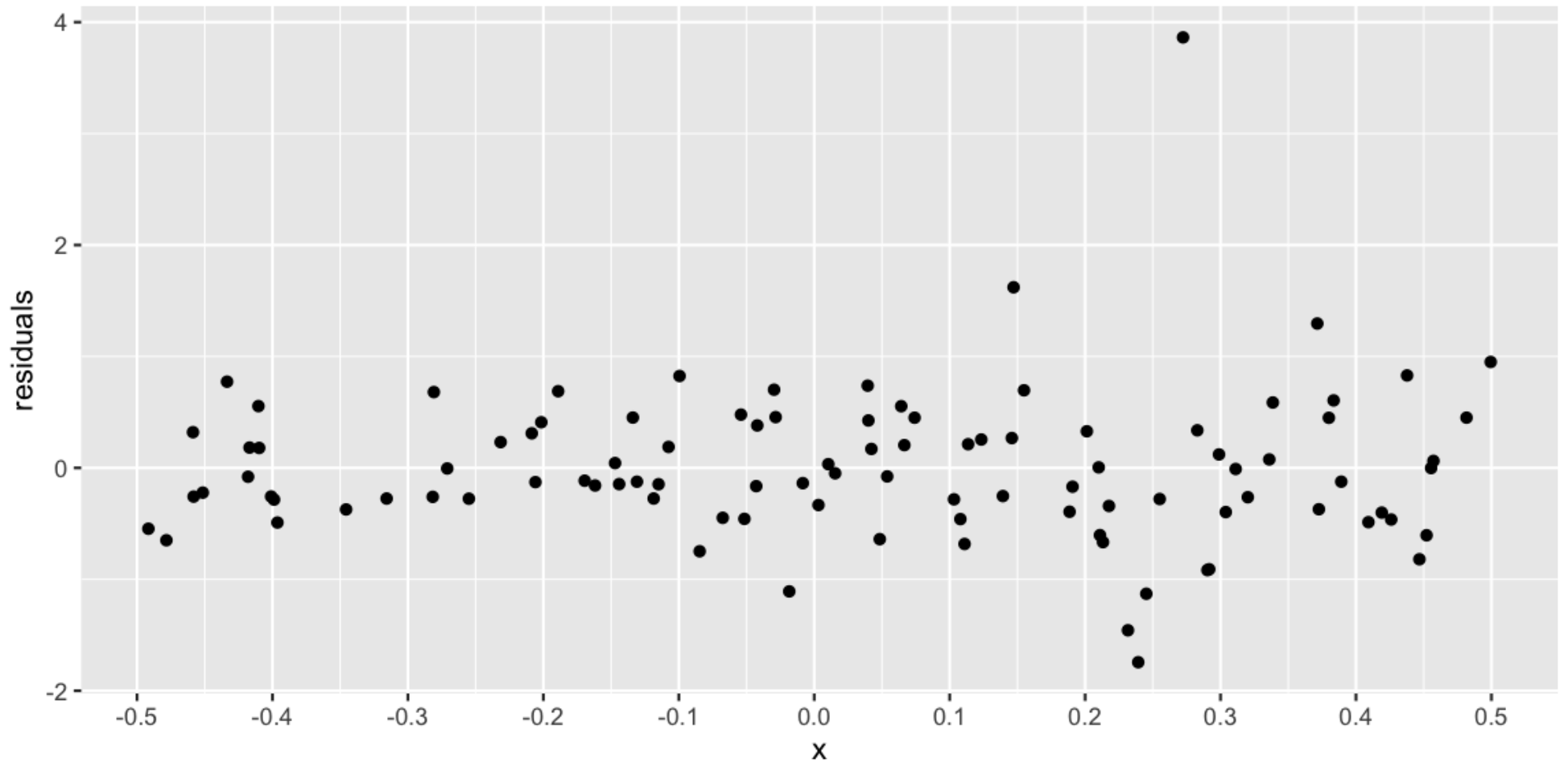


What's computed?



Residuals

```
ggplot(df_rp_aug, aes(x=x, y= .resid)) + geom_point() +
  ylab("residuals") + scale_x_continuous(breaks=seq(-0.5, 0.5, 0.1))
```



Goodness of fit - Root Mean Square Error

```
gof <- printcp(df_rp, digits=3)
##
## Regression tree:
## rpart(formula = y ~ x, data = df)
##
## Variables actually used in tree construction:
## [1] x
##
## Root node error: 359/100 = 3.59
##
## n= 100
##
##       CP nsplit rel error xerror   xstd
## 1 0.4611      0     1.000  1.008 0.1426
## 2 0.3044      1     0.539  0.580 0.0891
## 3 0.0419      2     0.235  0.301 0.0643
## 4 0.0315      3     0.193  0.248 0.0621
## 5 0.0240      4     0.161  0.246 0.0623
## 6 0.0114      5     0.137  0.218 0.0617
## 7 0.0100      6     0.126  0.216 0.0616
```

goodness of fit?

The relative error is $1 - R^2$. For this example, after 6 splits it is 0.1371214. So $R^2 = 0.8628786$.

```
1-sum(df_rp_aug$e^2)/sum((df$y-mean(df$y))^2)
```

Strengths

- There are no parametric assumptions underlying partitioning methods
- Can handle data of unusual shapes and sizes?
- Can identify unusual groups of data
- Provides a tree based graphic that is fun to interpret
- Has an efficient heuristic of handling missing values.
- The method could be influenced by outliers, but it would be isolating the effect to one partition.

Weaknesses

- Doesn't really handle data that is linear very well
- Can require tuning parameters to get good model fit
- Also means that there is not a nice formula for the model as a result, or inference about populations available

Next week: Classification trees

When the response is categorical, the model is called a classification tree. The criteria for making the splits changes also. There are a number of split criteria commonly used. If we consider a binary response ($y=0, 1$), and p is the proportion of observations in class 1.

- Gini: $2p(1-p)$
- Entropy: $-p(\log_e p) - (1-p)\log_e(1-p)$

Which rewards splits where the observations are all one class.

Your Turn: Lab exercise

- Return of the paintings data
- Just predict price with year

```
pp <- read_csv(here::here("slides/data/paris-paintings.csv"))

pp_lm <- lm(logprice ~ Height_in + Width_in, data = pp)
pp_rp <- rpart(logprice ~ Height_in + Width_in + year, data = pp)

pp_lm_aug <- augment(pp_lm)
pp_rp_aug <- augment(pp_rp)
library(rpart.plot)
rpart.plot(pp_rp)
```

```
tidy(pp_rp_all) %>%
  ggplot(aes(x = importance,
             y = reorder(variable, importance))) +
  geom_col()
```