Data Analytics
CS301

Exploratory Data Analysis
(Categories and Bins)

Week 5
Spring 2023
Oliver BONHAM-CARTER

Let's Make a Table of Data, *off the cuff*

- What if we want to *quickly* make a data set and work with it?
- This technique could be used to grow data tables from data from copied and pasted data.
- We will be using the "Tibble" package for R.
  - Provides a "tbl_df" class (the "tibble") that provides stricter checking and better formatting than the traditional data frame (2-dim array of data or table).

For example, you could make a data set to track rainfall!

	A	B
1	Daily rainfall	Particulate
2	(centimeters)	(micrograms/cubic meter)
3	4.1	122
4	4.3	117
5	5.7	112
6	5.4	114
7	5.9	110
8	53	114
9	3.6	128
10	1.9	137
11	7.3	104

Installing and Loading the *Tibble* Package

```
# Install the library containing the data.
install.packages("tibble")
library(tibble)
library(tidyverse)
```

Use `tibble()` to Create a Table

```
library(tibble)
# Create a new tibble by combining vectors using the tibble() function.
tibble(
    col1 = c("a1","b1","c1","d1"),
    col2 = c("a2","b2","c2","d2"),
    col3 = c("a3","b3","c3","d3"),
    col4 = c(14, 24, 34, 44)
    )
# What are the data types here? How do you know??
```

Use Create and View a Table

```
# Give your table a name.
SampleData <- tibble(
   col1 =
c("a1","b1","c1","d1"),
   col2 =
c("a2","b2","c2","d2"),
   col3 =
c("a3","b3","c3","d3"),
   col4 = c(14,24,34,44)
   )

  SampleData[,1] #Cols

  SampleData[1,] #Rows

  # Element of first col, first
  row
  SampleData[1,1]
```

Note, with View(), your data table appears transposed

	col1	col2	col3	col4
1	a1	a2	a3	14
2	b1	b2	b3	24
3	c1	c2	c3	34
4	d1	d2	d3	44

Use Check the Rows and Cols

#SampleData[rows,cols]

Allegheny College

```
SampleData <- tibble(
col1 =
c("a1","b1","c1","d1"),
  col2 =
c("a2","b2","c2","d2"),
  col3 =
c("a3","b3","c3","d3"),
  col4 = c(14,24,34,44)
)
```

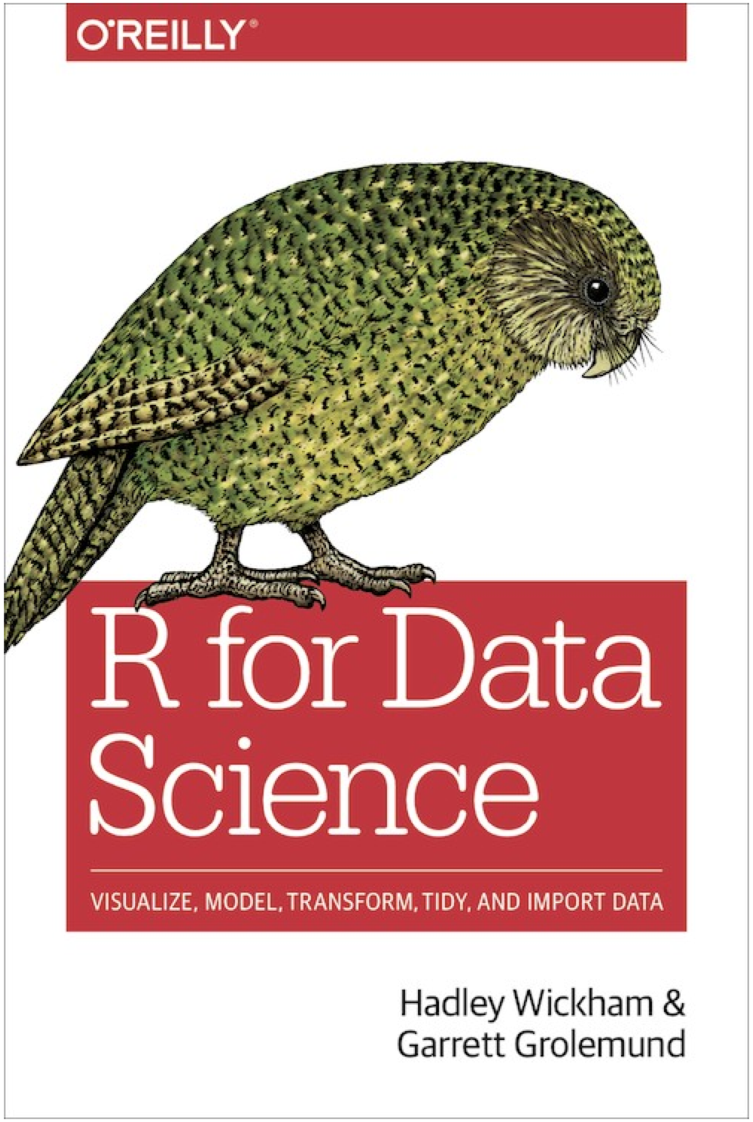
```
# first row
SampleData[1,]
# A tibble: 1 x 4
  col1  col2  col3  col4
  <chr> <chr> <chr> <dbl>
1 a1    a2    a3    14
```

```
SampleData[,1]
# A tibble: 4 x
1
  col1
  <chr>
1 a1
2 b1
3 c1
4 d1
```

Another Tibble Data Frame Using tibble()

```
# Create

friends_data <- tibble(

  name = c("Alexander", "Luke", "Freddy", "Sam",
"Amelia", "Daisy"),

  age = c(27, 25, 29, 26, 03, 25),

  height = c(180, 170, 185, 169, 60, 160),

  inCollege = c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE)

)

# Print

friends_data

#print first two lines

head(friends_data, 2)
```

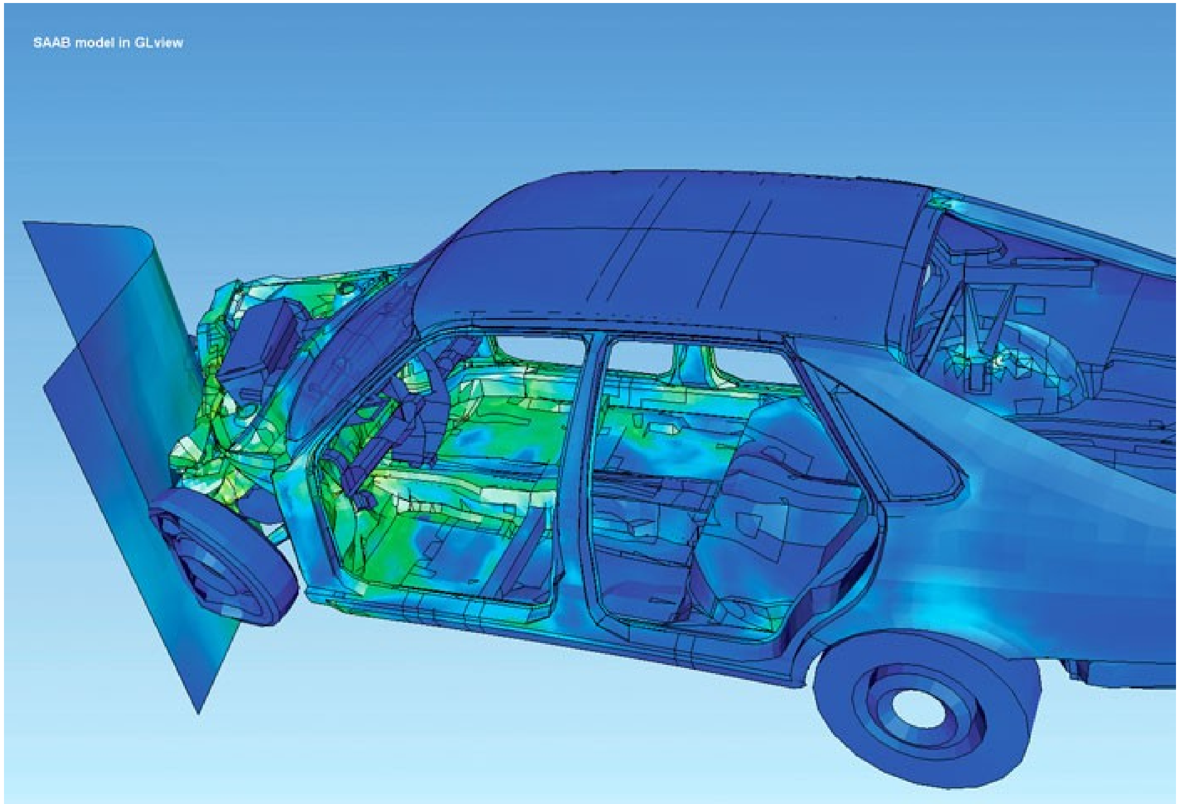
Where in the Web?
Where in the Book?



- Note the chapter differences!
- Book:
  - Chap 5: Exploratory Data Analysis
- Web:
  - http://r4ds.had.co.nz/exploratory-data-analysis.html
  - Chap 7: Exploratory Data Analysis

Exploratory Data Analysis

- The use of visualization and transformation to explore data systematically
- Learn more about data using graphical tools (easy to pot trends)
- Any technique for creating images, diagrams, or animations to communicate a message



Questions to Ask?

- No rules about which questions to ask to guide your research.
- Two types of general questions for making discoveries
  - *What type of variation occurs within my variables?*
  - *What type of co-variation occurs between my variables?*

Terms To Know

- A **variable** is a quantity, quality, or property that you can measure.
- A **value** is the state of a variable when you measure it. The value of a variable may change from measurement to measurement.
- An **observation** is a set of measurements made under similar conditions (you usually make all of the measurements in an observation at the same time and on the same object). An observation will contain several values, each associated with a different variable. I'll sometimes refer to an observation as a data point.
- **Tabular data** is a set of values, each associated with a variable and an observation. *Tabular data is tidy if each value is placed in its own "cell", each variable in its own column, and each observation in its own row.*

Terms To Know

- **Categorical variables:** variables that can take on one of a limited and usually fixed number of possible values, assigning each individual or other unit of observation to a particular group or nominal category
- **Categorical data** consists of categorical variables or grouped data
- Categorical data can only take one of a small set of values
  - Gender Identity: **M**ale or **F**emale
  - Months: January = “**1**” … December = “**12**”

Nationality	C1	C2	C3
French	0	0	1
Italian	1	0	0
German	0	1	0
Other	−1	−1	−1

Categorical Data in Diamonds

```
# What kind of data do we have?

View(diamonds), names(diamonds), or
diamonds
```

Where is the categorical data?

```
> diamonds
# A tibble: 53,940 x 10
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

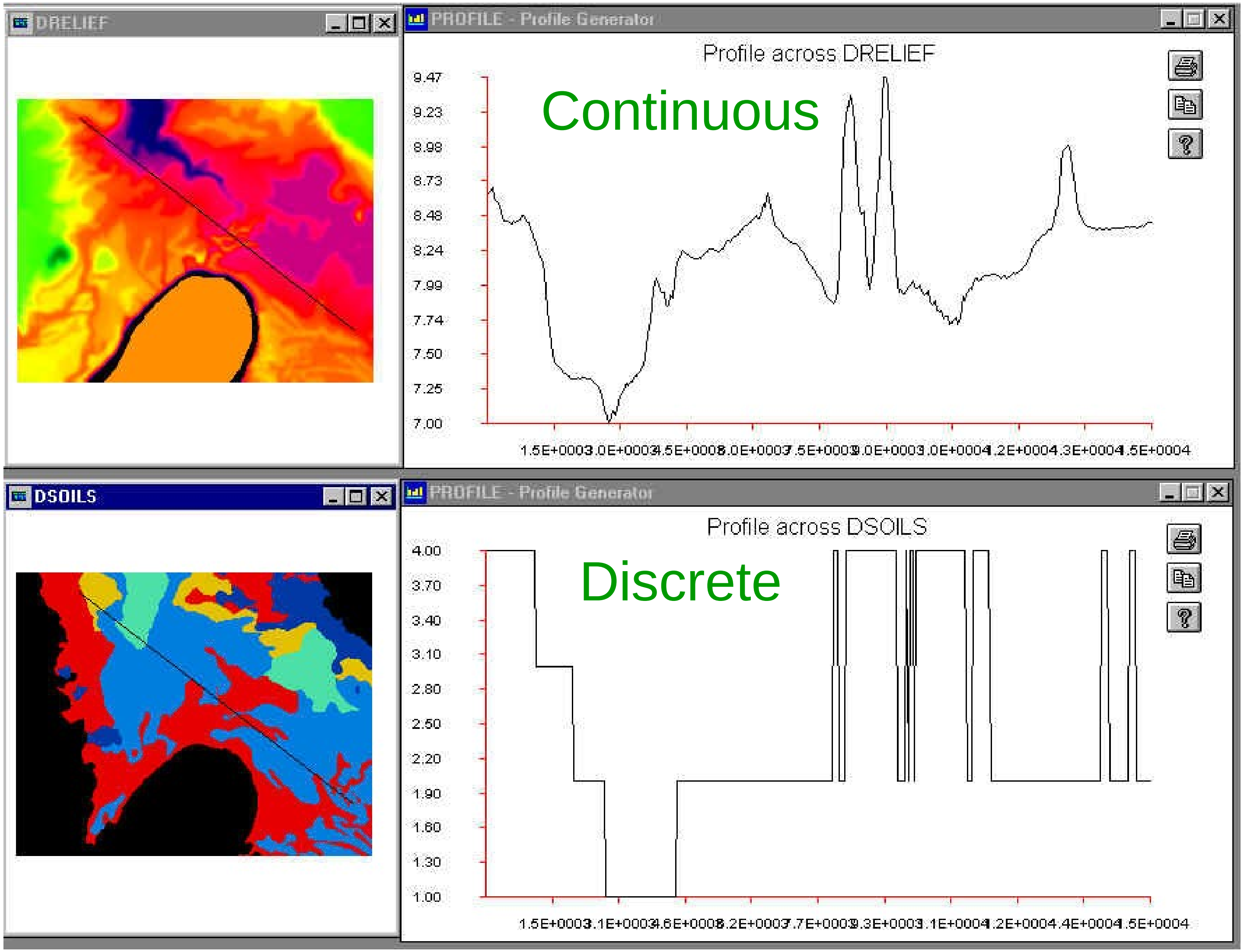
Plotting Categorical Cuts

```
#generate point plot (as we have done before)
ggplot(data = diamonds) +
geom_point(mapping = aes(x = cut, y = carat,
color = clarity))
# generate a histogram
ggplot(data = diamonds) +
geom_bar(mapping = aes(x = cut))
# find “local” statistics about the “cut” column:
diamonds %>% count(cut)
```

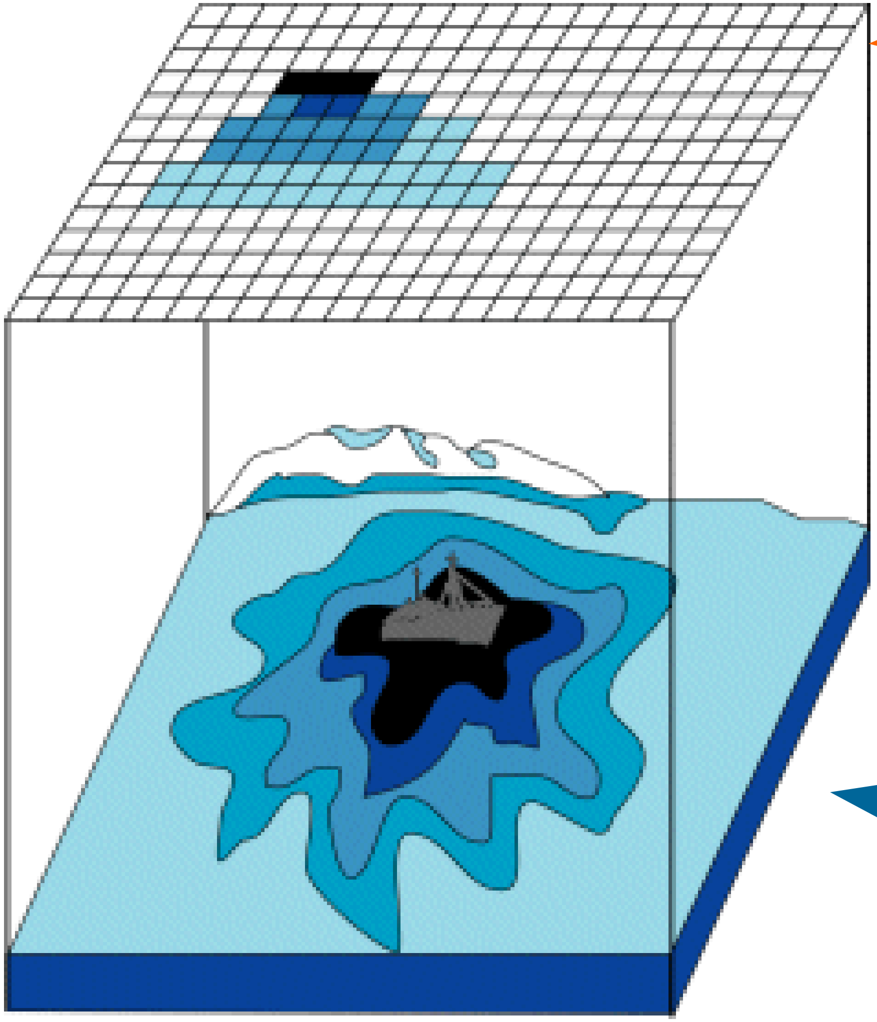
- **What did that last command return?!**
- **What is the categorical data!**

Continuous Data in Diamonds

- **Continuous data** is information that can be measured on a continuum or scale.
- Can have almost any numeric value and can be meaningfully subdivided into finer and finer increments, depending upon the precision of the measurement system.



Continuous Data in Diamonds



Discrete data: Strict cut-offs to define data points in a cell

Note the discrete data as a function of the continuous data.

Continuous data: No strict cut-offs, data points are not grouped in cells.

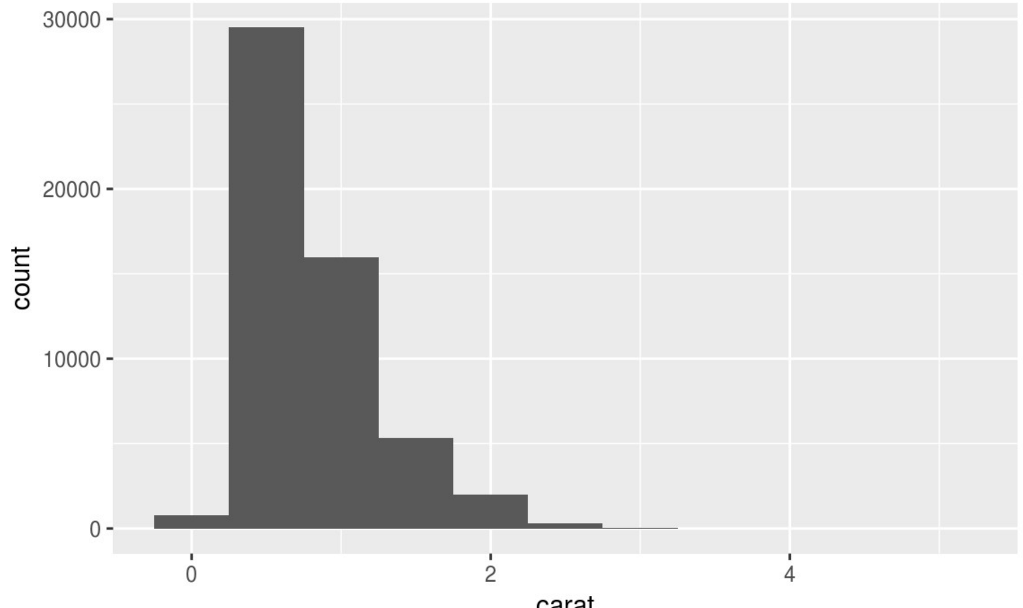
Continuous Data in Diamonds

Where is the continuous data in this table?

```
> diamonds
# A tibble: 53,940 x 10
   carat       cut color clarity depth table price     x     y     z
   <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
 2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
 3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
 4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
 5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

Plotting Bins:
Continuous to the Categorical

```
# Study continuous variable distribution by histogram
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat),
  binwidth = 0.5)
```



The Data Behind the Plots

```
# What data is filling our bins?

# Find "local" statistics about the "carat" column:

diamonds %>% count(carat)
```

Count() finds the number of occurrences of a particular number

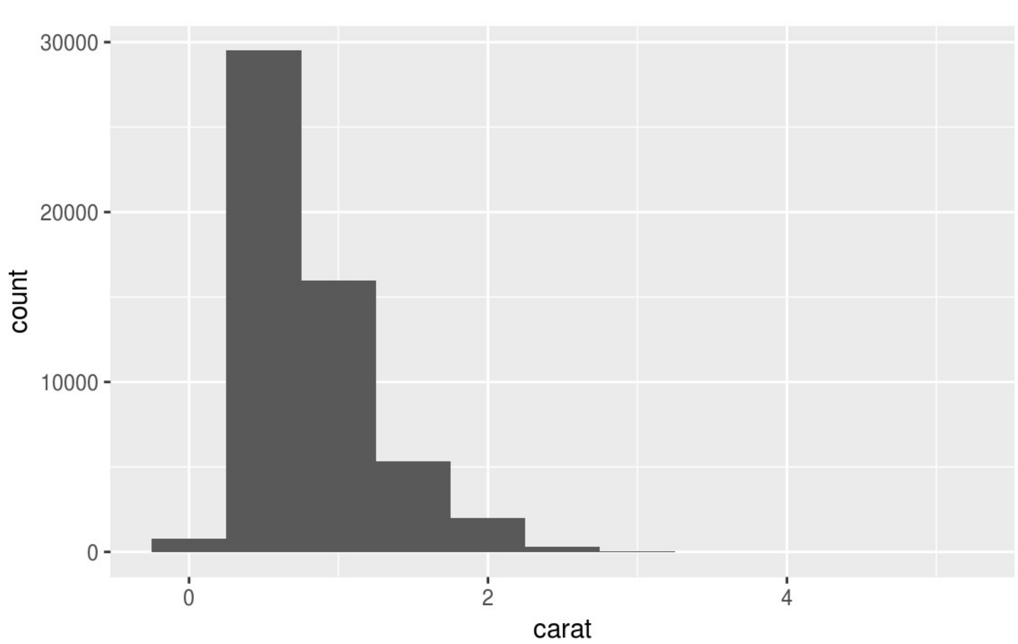
```
# Discretise numeric data into categorical

?cut_width()
```

What did that last command return?!

Pipe: %>% transfers one product to another function. Say, "*and then*" when you see it.

Plot and Cut of Continuous Carats

```
diamonds %>% count(cut_width(carat,0.5))
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat),
  binwidth = 0.5)
```

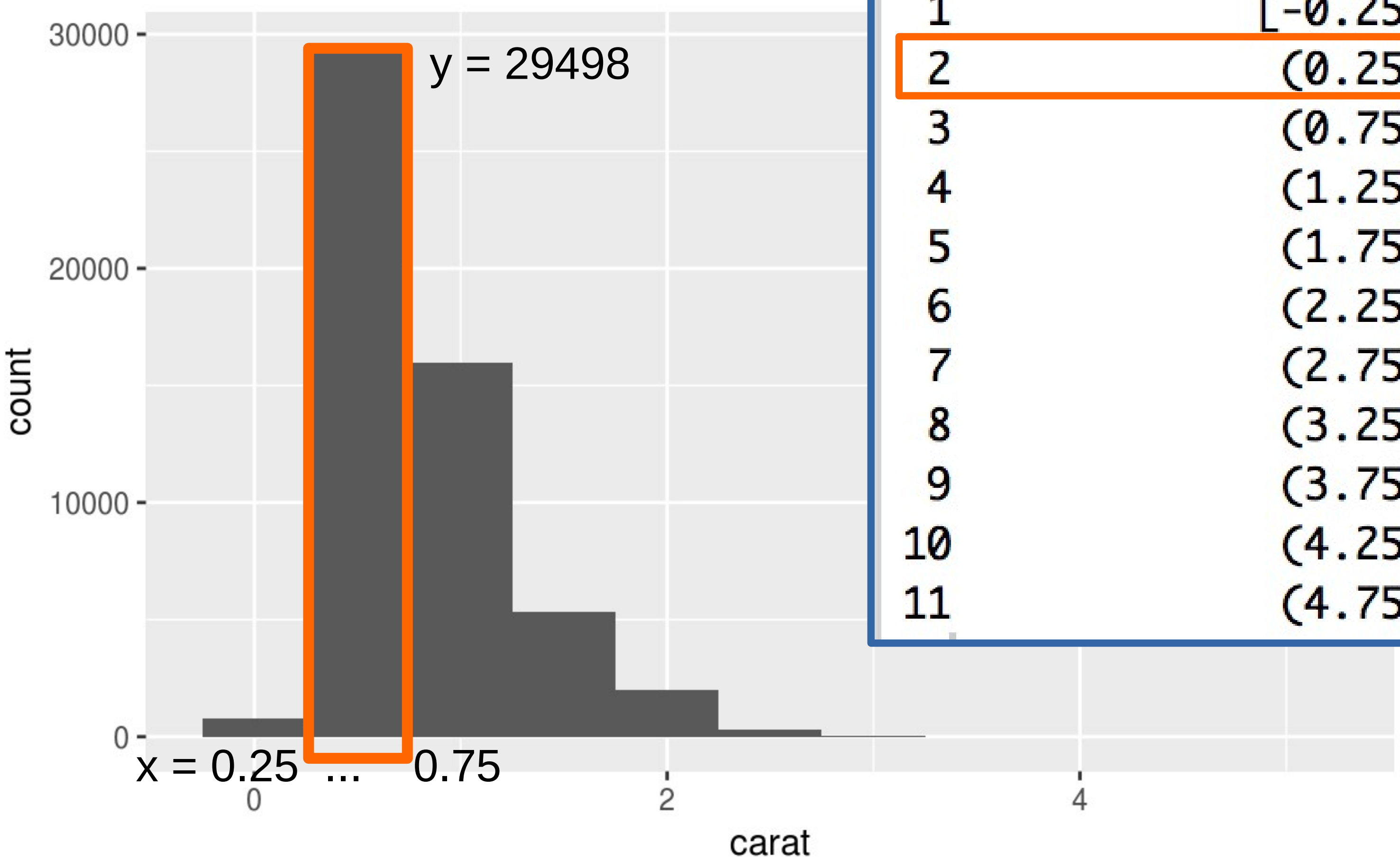



```
> diamonds %>% count(cut_width(carat,0.5))
# A tibble: 11 x 2
   `cut_width(carat, 0.5)`     n
                    <fctr> <int>
1            [-0.25,0.25]    785
2             (0.25,0.75]  29498
3             (0.75,1.25]  15977
4             (1.25,1.75]   5313
5             (1.75,2.25]   2002
6             (2.25,2.75]    322
7             (2.75,3.25]     32
8             (3.25,3.75]      5
9             (3.75,4.25]      4
10            (4.25,4.75]      1
11            (4.75,5.25]      1
```

Histogram as Text

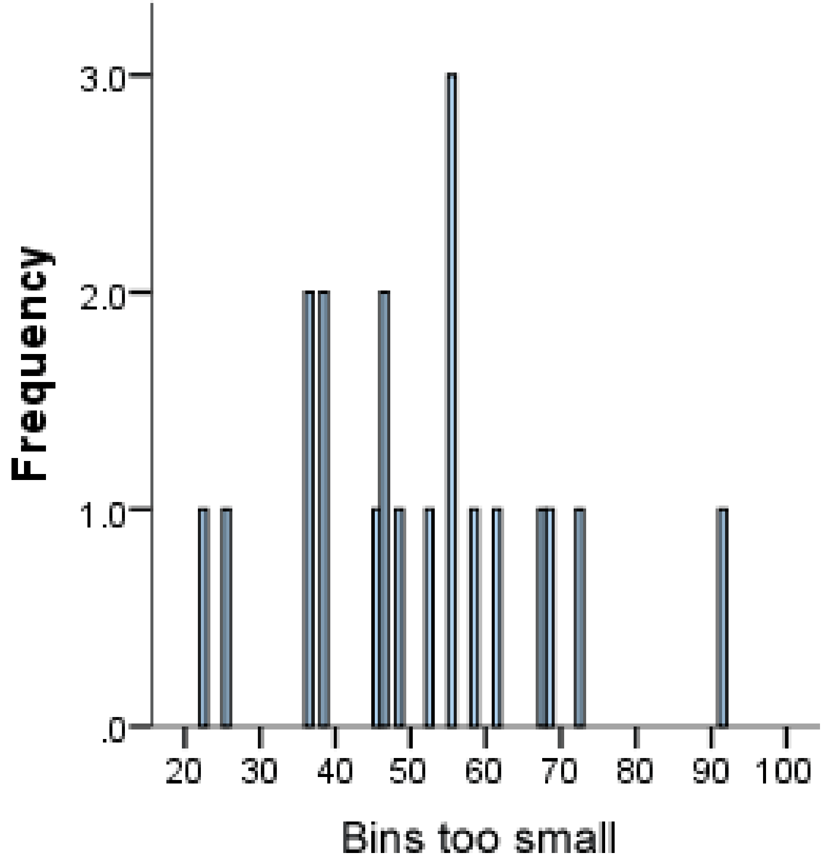
- The *cut_width()* gives a textual representation of the histogram.

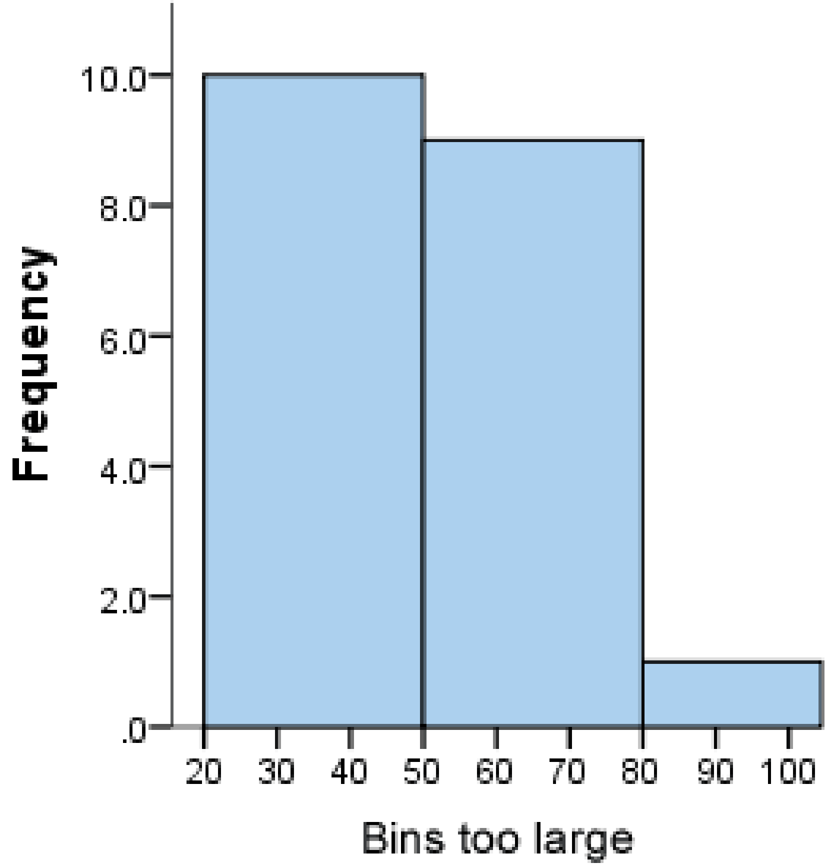
```
> diamonds %>%
+       count(cut_width(carat, 0.5))
# A tibble: 11 x 2
   `cut_width(carat, 0.5)`      n
                    <fctr> <int>
1            [-0.25,0.25]    785
2             (0.25,0.75]  29498
3             (0.75,1.25]  15977
4             (1.25,1.75]   5313
5             (1.75,2.25]   2002
6             (2.25,2.75]    322
7             (2.75,3.25]     32
8             (3.25,3.75]      5
9             (3.75,4.25]      4
10            (4.25,4.75]      1
11            (4.75,5.25]      1
```



Different Bin Widths

- Set the width of the intervals in a histogram with the `binwidth` argument, which is measured in the units of the *x* variable.
- **Left histogram**: bins are too small, too much individual data and hides underlying pattern (frequency distribution).
- **Right histogram**: bins are too large, hard to spot trends in the data.





Different Bin Widths


```
# histograms
# Note: we zoom in on carats sizes < 3
  smaller <- diamonds %>% filter(carat < 3)
  ggplot(data = smaller, mapping = aes(x = carat)) +
  geom_histogram(binwidth = ??)
```

Which is the best *binwidth* setting for this data?? Why??

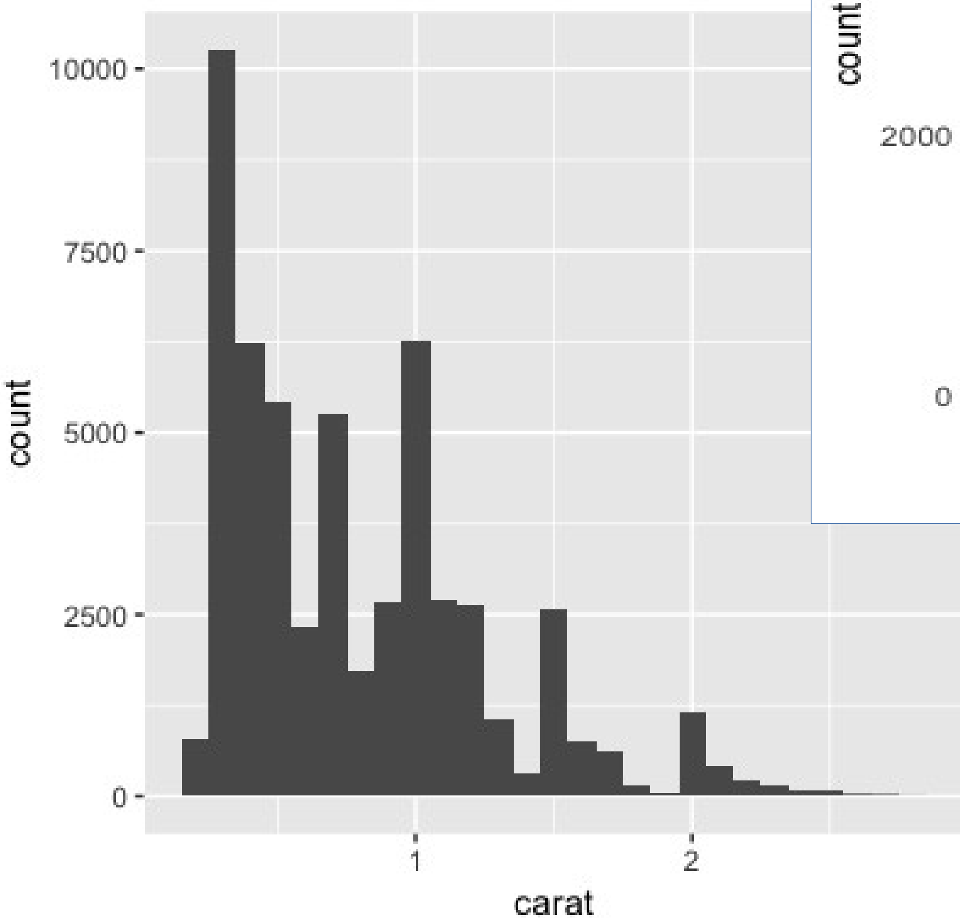
Different Bin Widths

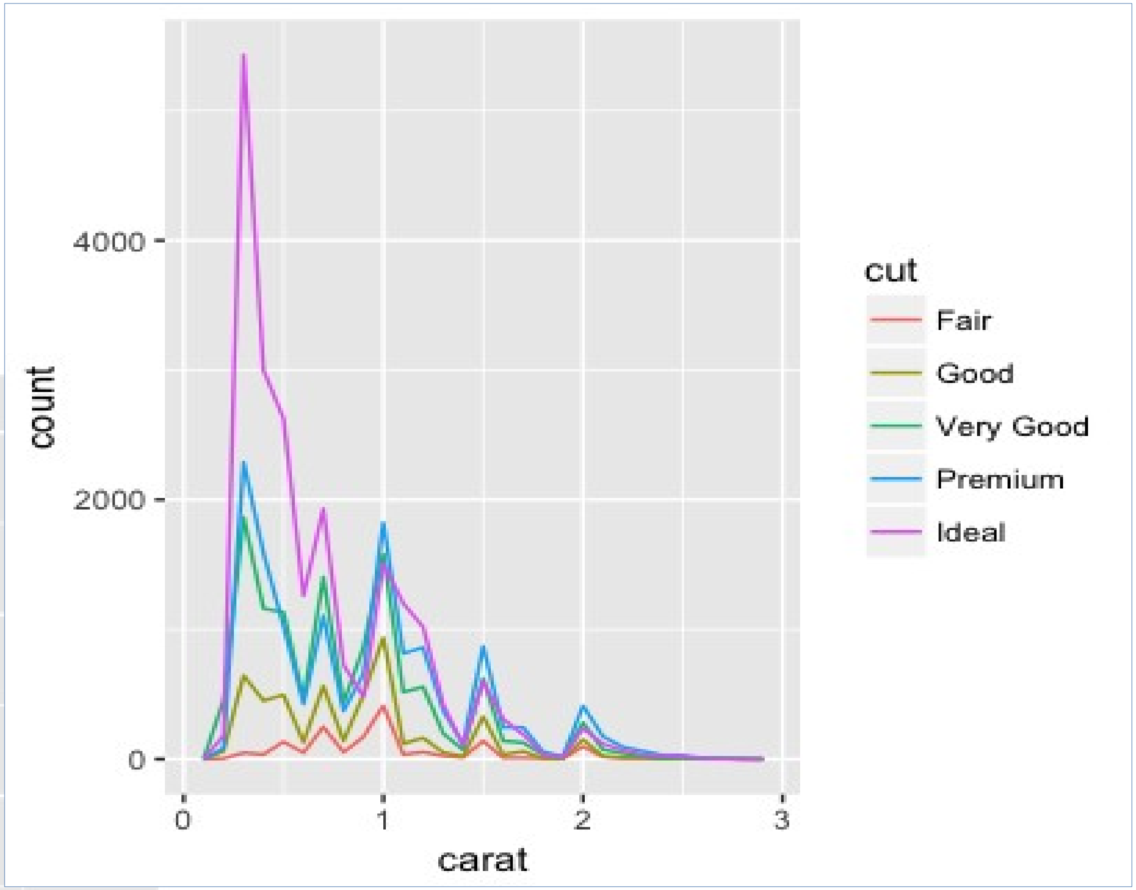
```
# freqPoly plot
smaller <- diamonds %>% filter(carat < 3)
ggplot(data = smaller, mapping = aes(x =
carat, colour = cut))  +
geom_freqpoly(binwidth = ??)
```

What does this graphic inform us? What *binwidth* setting is too small? Too large? Is perfect?

THINK

Same Data, Different Plot...





- **The way you present your data may help you to see more.**

Viewing Data: *Diamond*

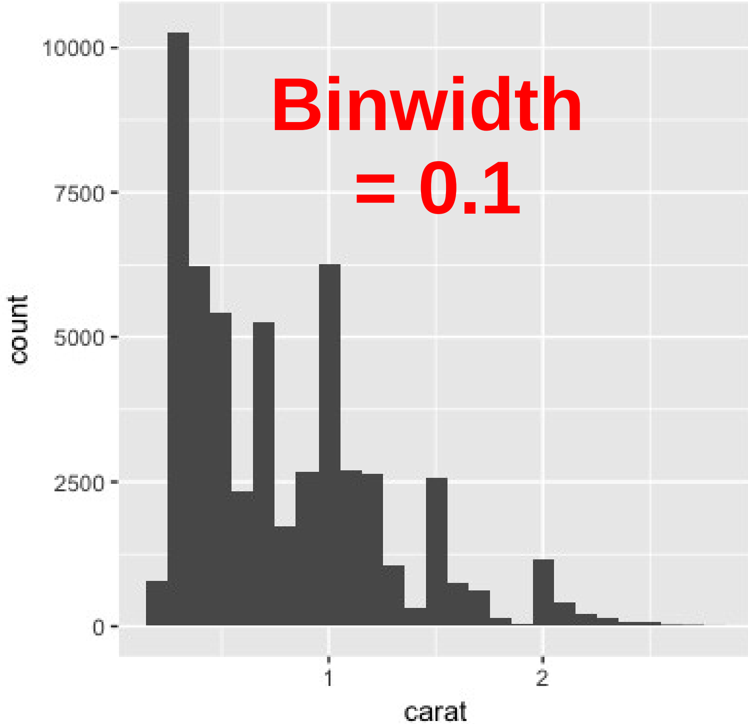
```
 smaller <- diamonds %>%
 filter(carat < 3)
  ggplot(data = smaller, mapping = aes(x =
carat)) + geom_histogram(binwidth = 0.1)


#  instead of displaying the counts with bars,
use lines instead that can be clearly seen.
  ggplot(data = smaller, mapping = aes(x =
carat, colour = cut)) + geom_freqpoly(binwidth
= 0.1)
# exact numbers
diamonds %>% count(cut_width(carat, 0.1))
```

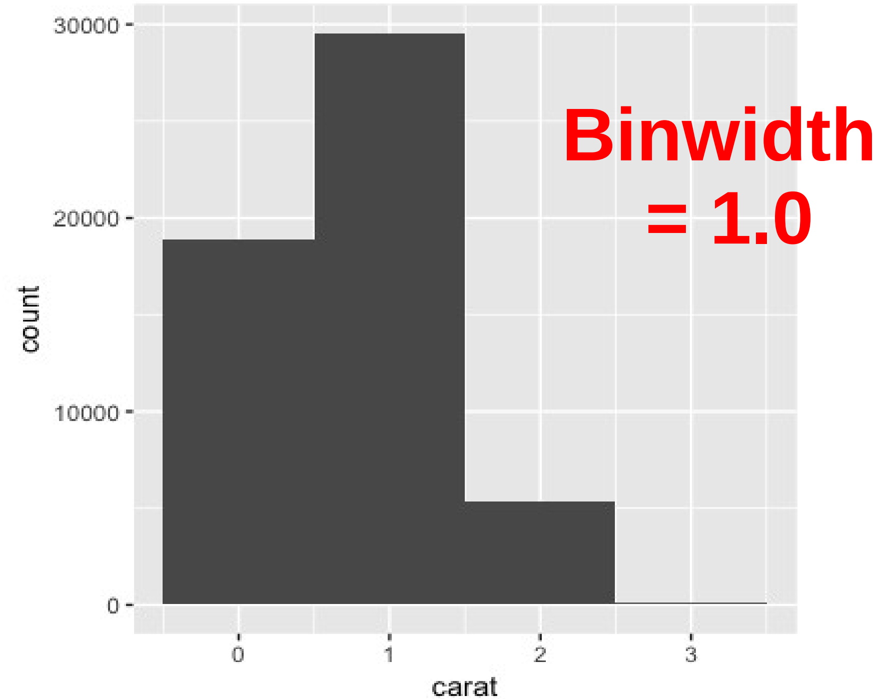
Data and Binwidths

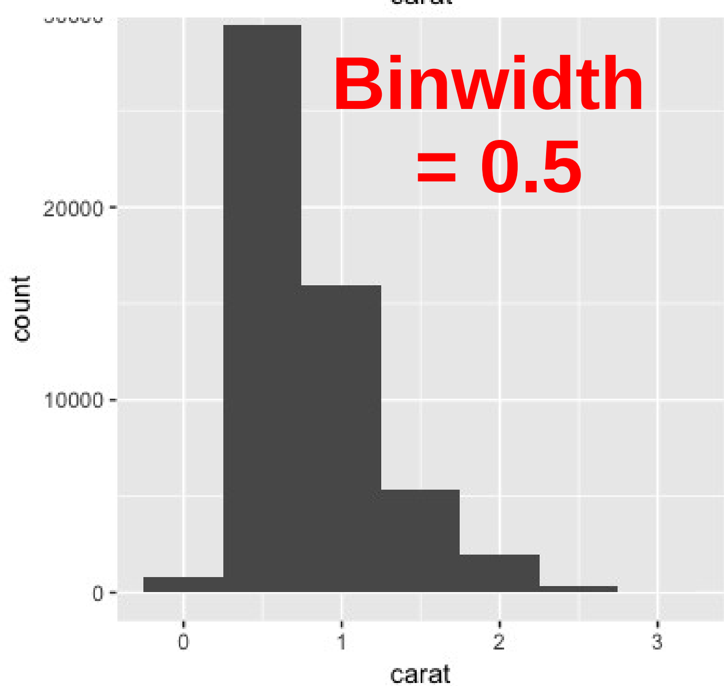
- Use this set or find another one using *data()* to play around with histograms of polyfreq plots
- Try changing the *binwidth* settings to see what new patterns you can see.
- What other types of graphs from your notes can you make?

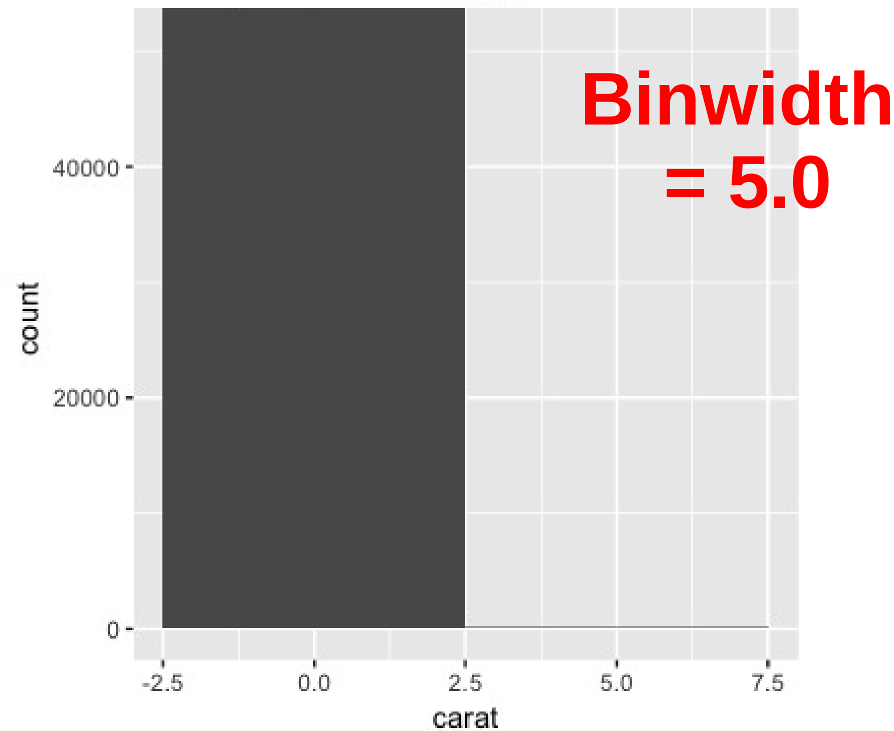
Let’s Explore Bin Widths!

```
# Install the library containing the data.
 library(tidyverse)
 smaller <- diamonds %>%
 filter(carat < 3)
 ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 0.1)
```


Binwidth
= 0.1
10000
7500
5000
2500
0
count
1
2
carat


Binwidth
= 1.0
30000
20000
10000
0
count
0
1
2
3
carat


Binwidth
= 0.5
20000
10000
0
count
0
1
2
3
carat

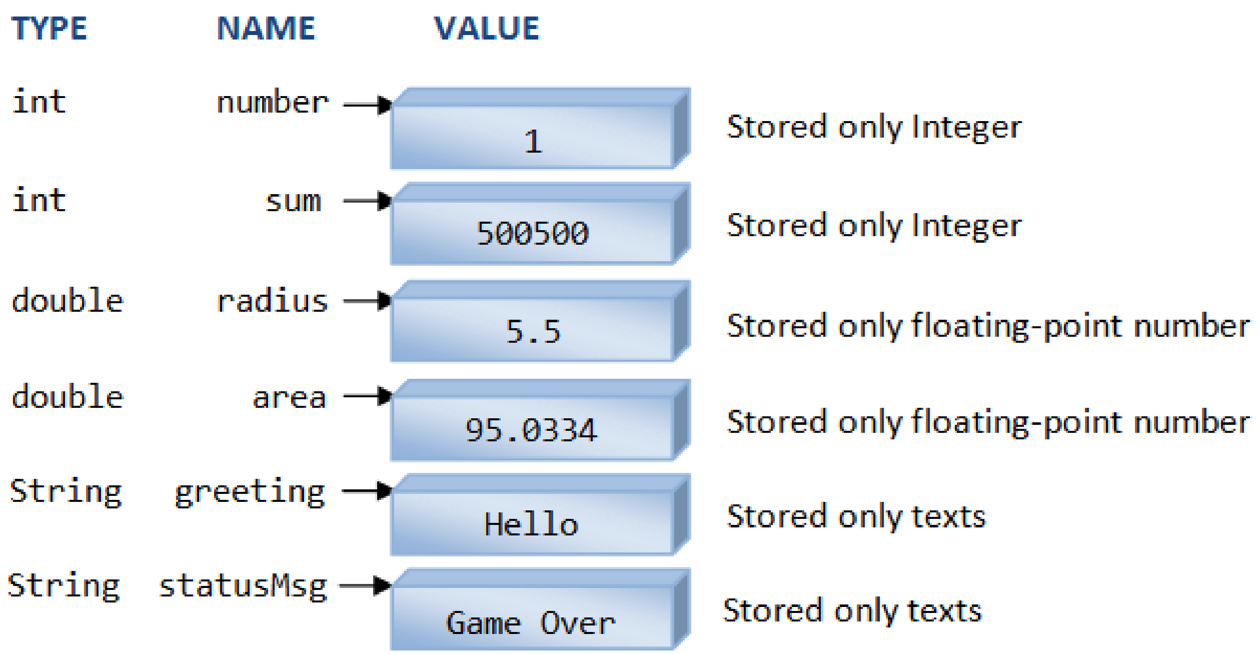

Binwidth
= 5.0
40000
20000
0
count
-2.5
0.0
2.5
5.0
7.5
carat

Let’s Explore Bin Widths!

```
# try more bin sizes!

ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 0.1)

ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 0.2)

ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 0.3)

ggplot(data = smaller, mapping = aes(x = carat)) + geom_histogram(binwidth = 5)
```

ALLEGHENY COLLEGE

R prefers DOUBLES over INTEGERS



A *variable* has a ***name***, stores a ***value*** of the declared ***type***.

- R uses IEEE 754 double-precision floating-point numbers. Floating-point numbers are more dense near zero.
- This is a result of their being designed to compute accurately (the equivalent of about 16 significant decimal digits, as you have noticed) over a very wide range.

R Likes DOUBLES But Can Use INTEGERS

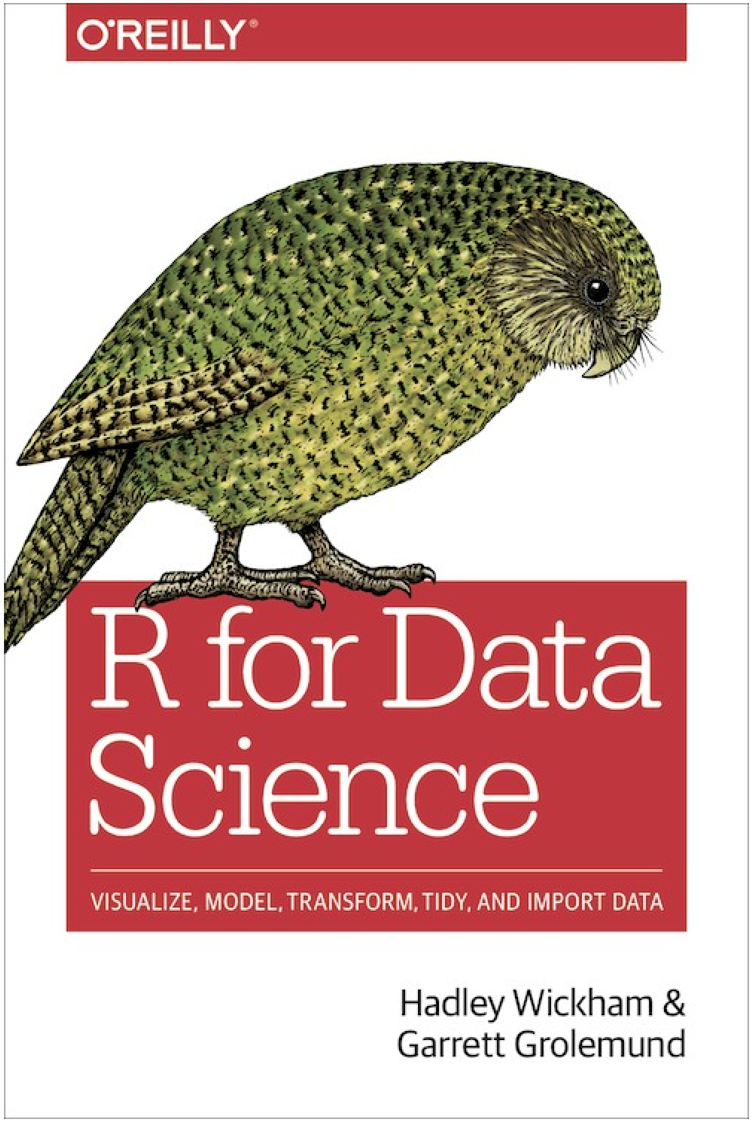
```
# Assign value of 1 to x_dbl
 x_dbl <- 1
# what type is x_dbl?
 typeof(x_dbl)

# Assign integer value to x_int
 x_int <- as.integer(1)
 typeof(x_int)
```

What variable types did you find?!

Let’s DOUBLE Some INTEGERS

```
#Assign a set of numbers to x_list
x_int <- 0:10
typeof(x_int)
#Assign a set and multiply each element by double
x_dbl <- 0:10 * 3.14
typeof(x_dbl)
x_int <- as.integer(x_dbl)
#Automatic changing of ints to doubles
```

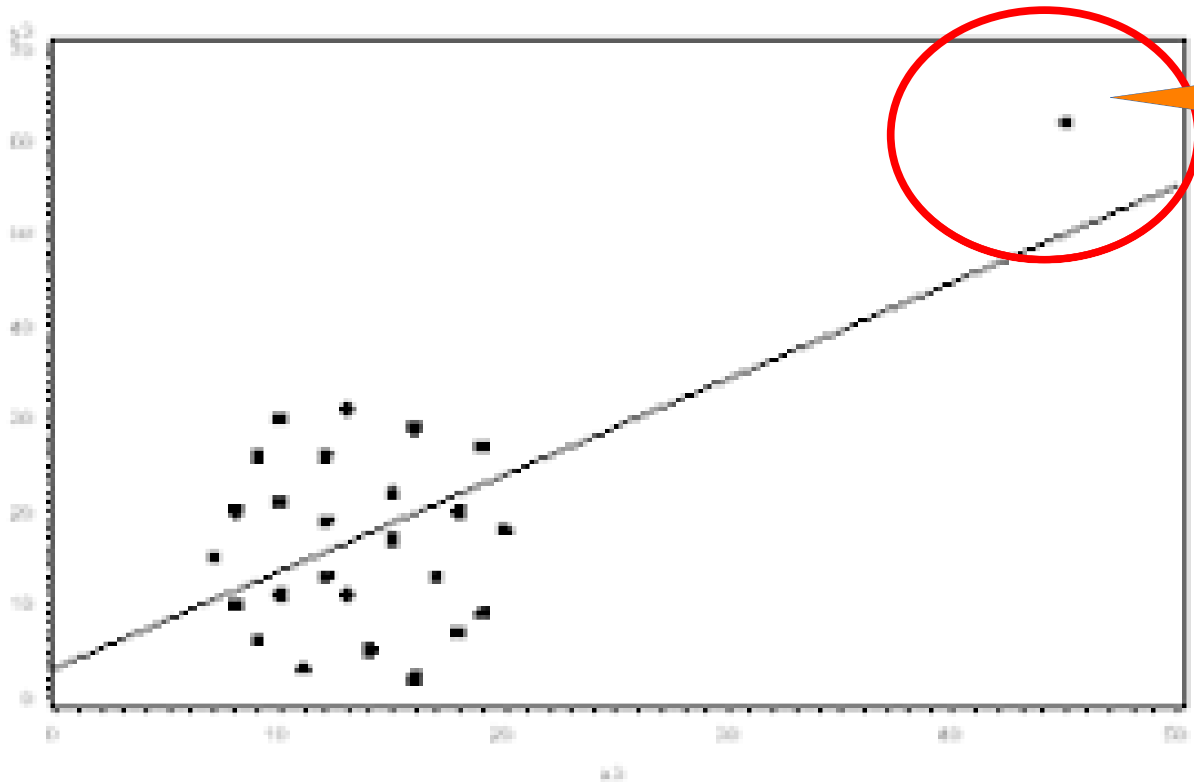
Where in the Web?
Where in the Book?



- Note the chapter differences!
- Book:
  - Chap 5: Exploratory Data Analysis
- Web:
  - http://r4ds.had.co.nz/exploratory-data-analysis.html
  - Chap 7: Exploratory Data Analysis

Outliers

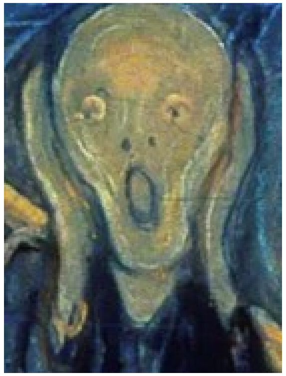
- Something that lies outside the main body or group that it is a part of, as a cow far from the rest of the herd, or a distant island belonging to a cluster of islands:



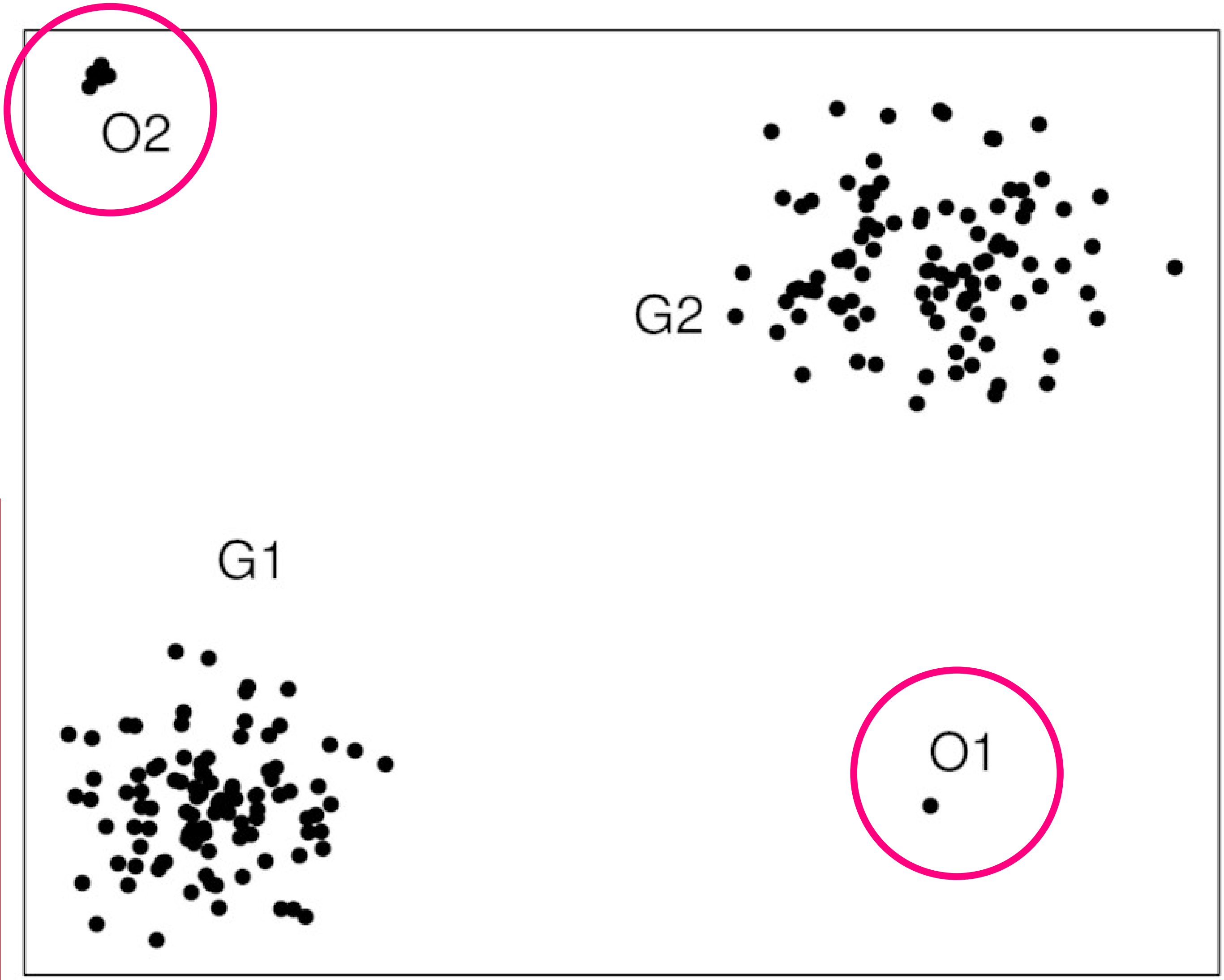
Is this an ***outlier*** or a ***discovery***?

Outliers

- Two groups with an outlier (O1 and O2) from each.



Where did These outliers Come from?



Data: *Diamond*

#Plot the *y* column of data.

ggplot(diamonds) + geom_histogram(mapping = aes(**x = y**), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 50)**)

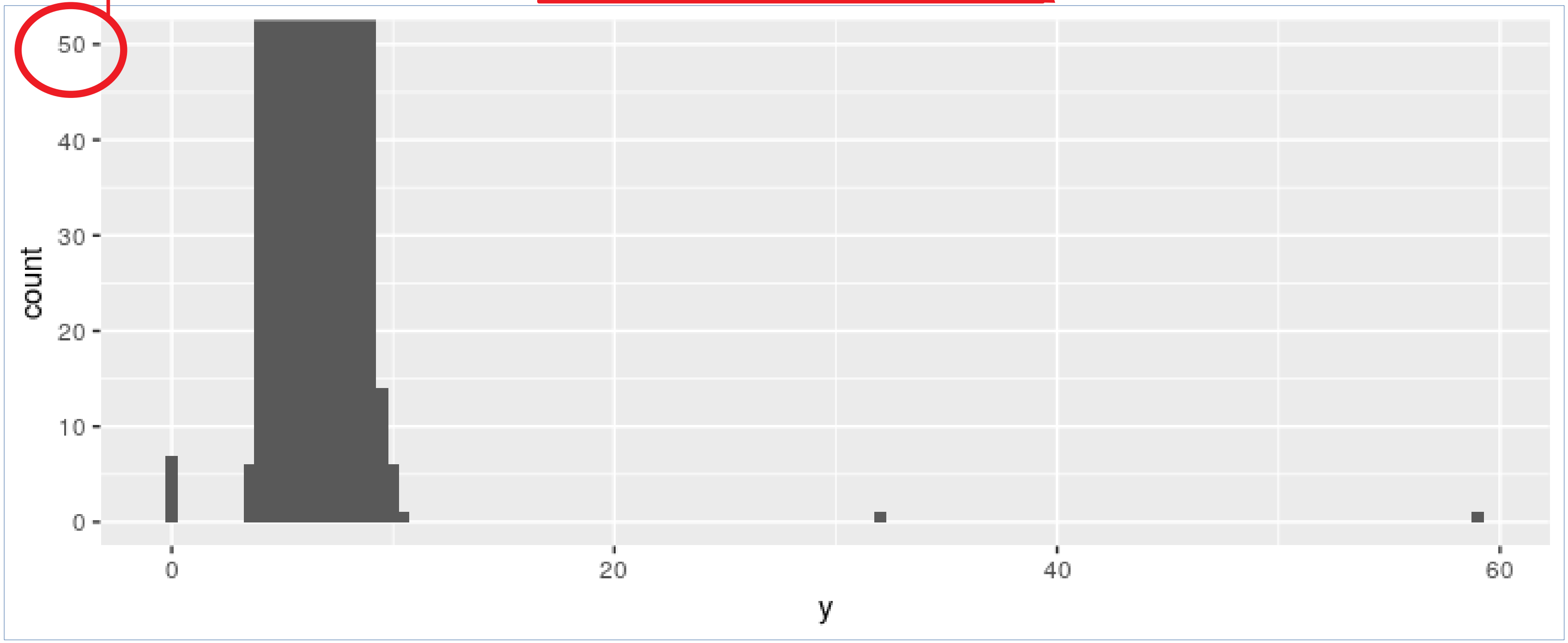
ggplot(diamonds) + geom_histogram(mapping = aes(x = y), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 20)**)

Ylim: Y-axis range: change to zoom-in on outliers. You might otherwise miss them. Try ylim = 10 to 10k

Data: *Diamond*

#Plot the *y* column of data.

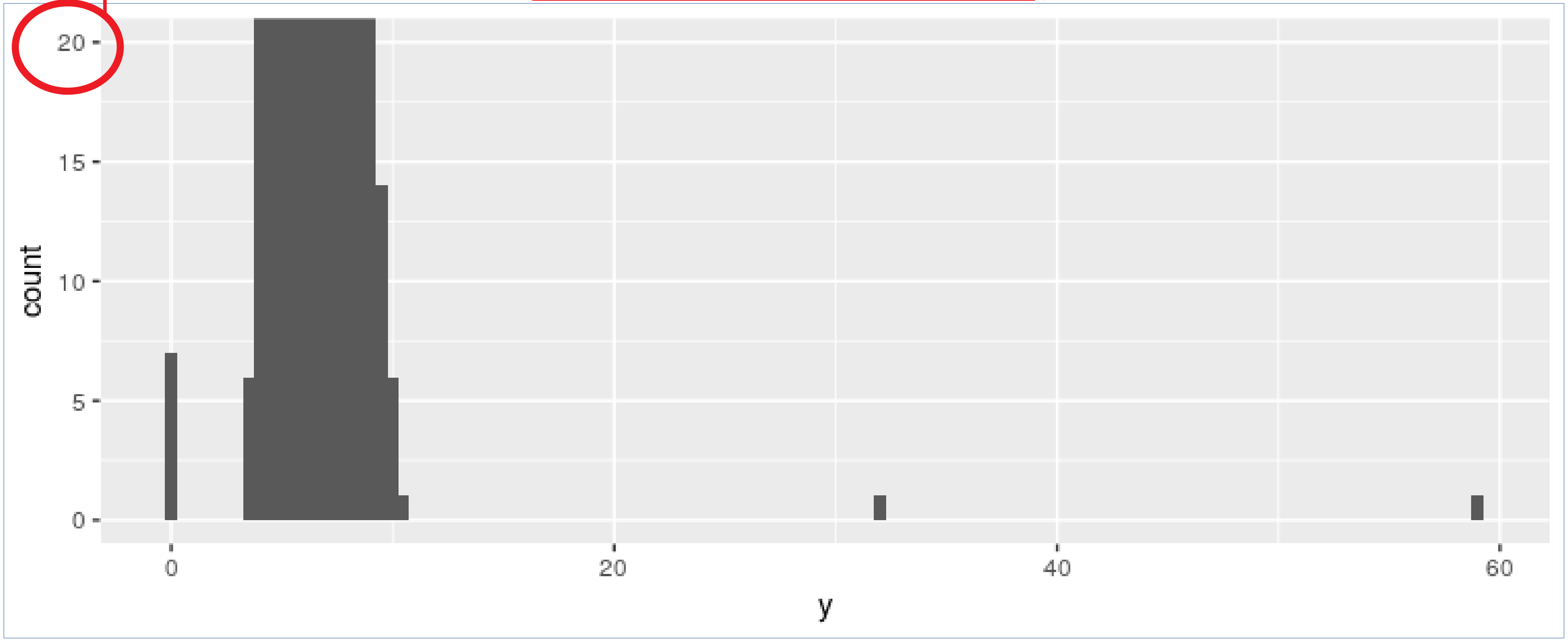
ggplot(diamonds) + geom_histogram(mapping = aes(**x = y**), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 50)**)



Data: *Diamond*

#Plot the *y* column of data.

ggplot(diamonds) + geom_histogram(mapping = aes(**x = y**), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 20)**)



Unusual Values:
isolate the outliers

```
# Collect the rows containing outliers
unusual <- diamonds %>%
    filter(y < 3 | y > 20)
    select(price, x, y, z)
    arrange(y)
```

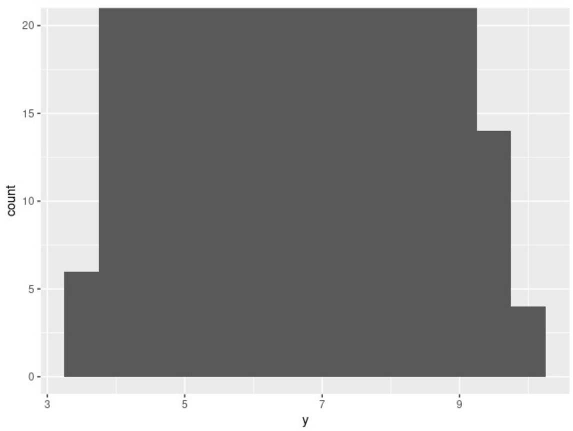
	price	x	y	z
1	5139	0.00	0.0	0.00
2	6381	0.00	0.0	0.00
3	12800	0.00	0.0	0.00
4	15686	0.00	0.0	0.00
5	18034	0.00	0.0	0.00
6	2130	0.00	0.0	0.00
7	2130	0.00	0.0	0.00
8	2075	5.15	31.8	5.12
9	12210	8.09	58.9	8.06

- Use filter and select from *dplyr* to isolate.
- There there are three unusual values: 0, ~30, and ~60.

Plot All Data But Outliers

```
noOutliers <- diamonds %>%
  filter(y > 3) %>%
  filter(y<10) %>%
  select(price, x, y, z) %>%
  arrange(y)


ggplot(noOutliers) +
  geom_histogram(mapping = aes(x = y), binwidth = 0.5) +
  coord_cartesian(ylim = c(0, 20))
```

Does this plot help
In our analysis?



Maybe a Better Plot?

```
noOutliers <- diamonds %>% filter(y > 3) %>%
  filter(y<10) %>% select(price, x, y, z) %>%
arrange(y)


ggplot(noOutliers) +
  geom_histogram(mapping = aes(x = y), binwidth =
0.09) + coord_cartesian(ylim = c(0, 2000))
```

Is this any better?