Data Analytics
CS301
Exploratory Data Analysis

Week 5
Fall 2018
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 data_frame() to Create a Table

- # Create a new tibble by combining vectors using the *data_frame()* function.

  ```
  data_frame(
     rowA = c("a1","b1","c1","d1"),
     rowB = c("a2","b2","c2","d2"),
     rowC = c("a3","b3","c3","d3"),
     rowD = c(14,24,34,44)
     )
  ```

- **What are the data types here? How do you know??**

Use data_frame() to Create a Table

- # Give your table a name.

```
SampleData <- data_frame(
  rowA = c("a1","b1","c1","d1"),
  rowB = c("a2","b2","c2","d2"),
  rowC = c("a3","b3","c3","d3"),
  rowD = 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

	rowA	rowB	rowC	rowD
1	a1	a2	a3	14
2	b1	b2	b3	24
3	c1	c2	c3	34
4	d1	d2	d3	44

Another Tibble Table Using data_frame()

- # Create

  friends_data <- data_frame(

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

    age = c(27, 25, 29, 26, 01, 25),

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

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

)

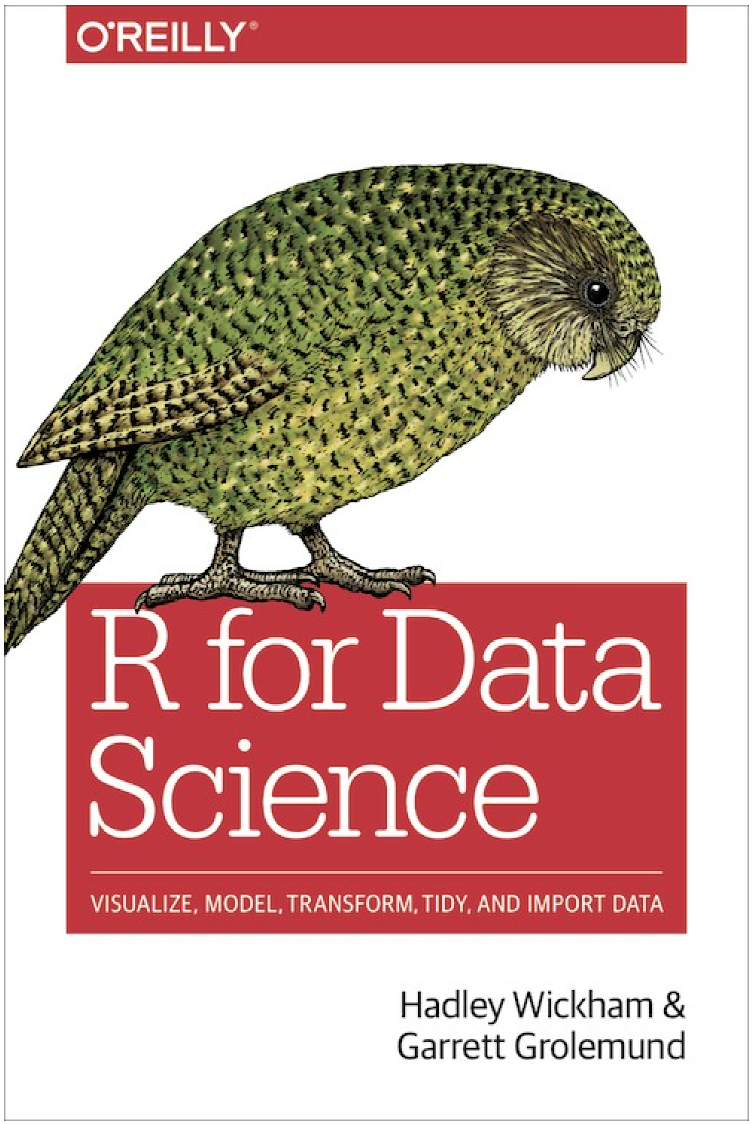
- # Print

  friends_data

- #print first two lines

  head(friends_data, 2)

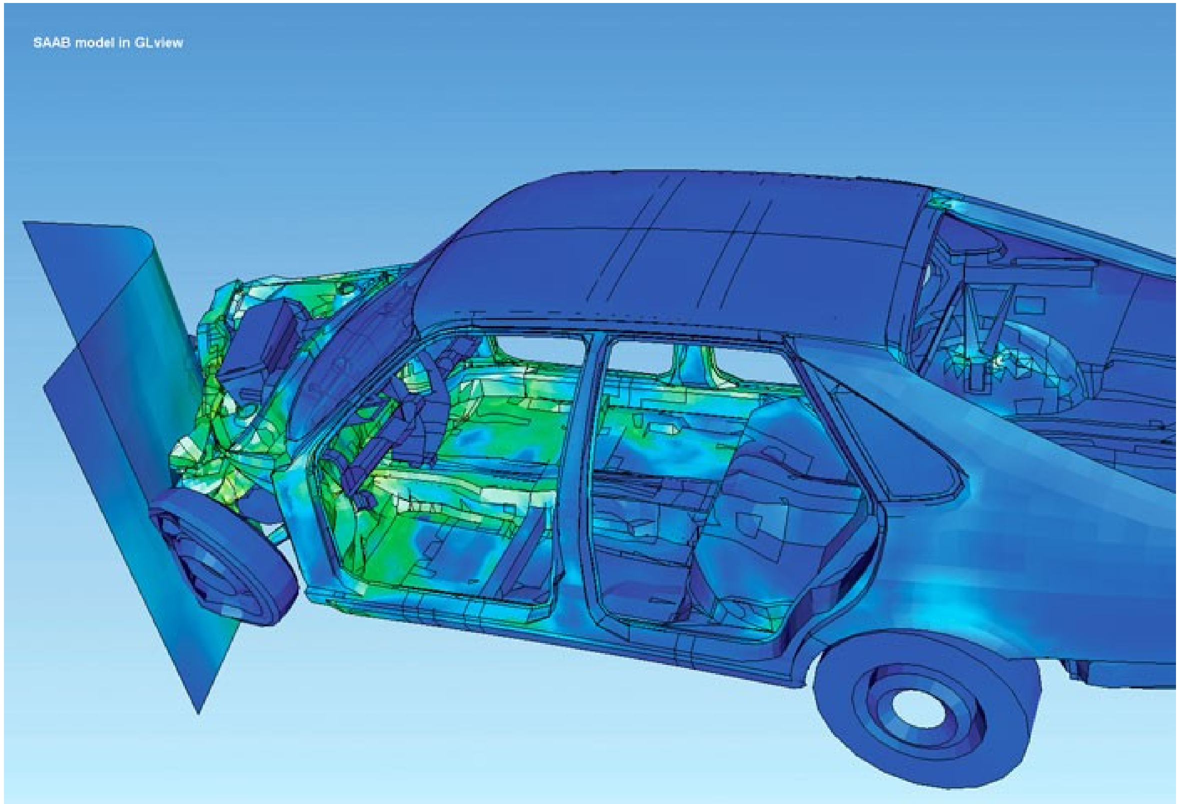
Where in the Web?
Where in the Book?



- 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 covariation 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** is the statistical data type consisting of categorical variables or of data that has been converted into that form, for example as grouped data
- Categorical data can only take one of a small set of values
  - **“M” for male, “F” for female**
  - **January = “1” … December = “12”**

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

What’s Ahead?

- We combine what you’ve learned about *dplyr* and *ggplot2* to interactively ask questions, answer them with data, and then ask new questions

- # **If is it not already installed, install *tidyverse*.**

  install.packages("tidyverse")
- #Otherwise just load the library.

  library(“tibble”)

Categorical Data in Diamonds

- # Is your data loaded?

  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
```

Plot the 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**

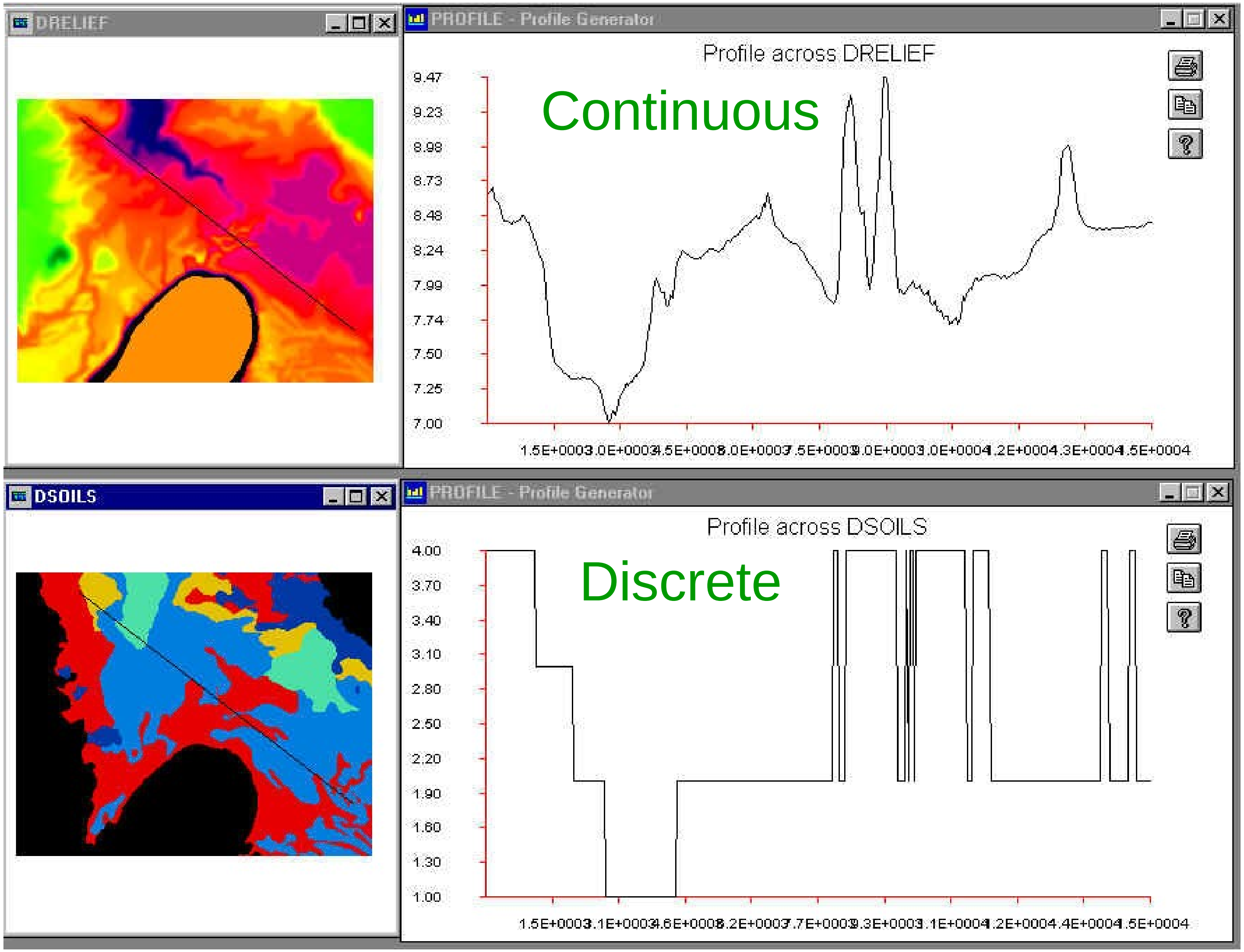
  ggplot(data = diamonds) +

  geom_bar(mapping = aes(x = cut))
- # find “local” statistics about the “cut” column:

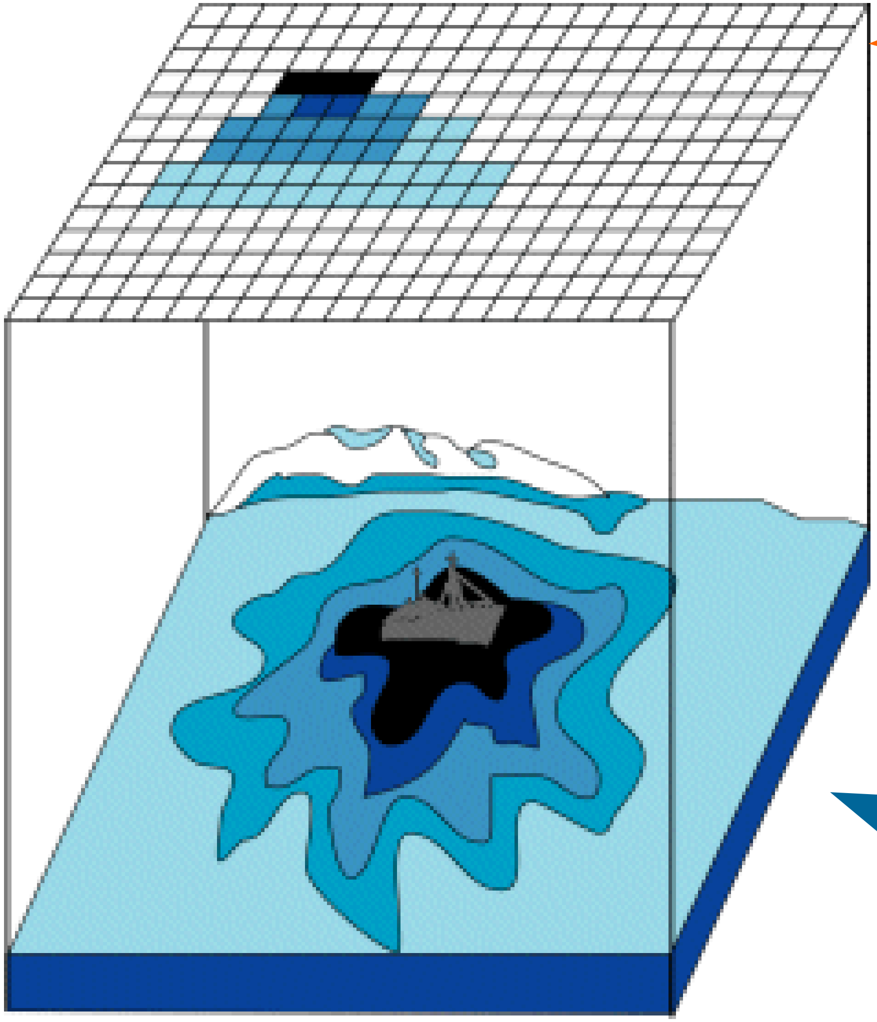
  diamonds %>% count(cut)
- **What did that last command return?!**
- **What is the categorical data!**

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

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

Continuous data

Continuous Data in Diamonds

Where is the continuous data in the table?

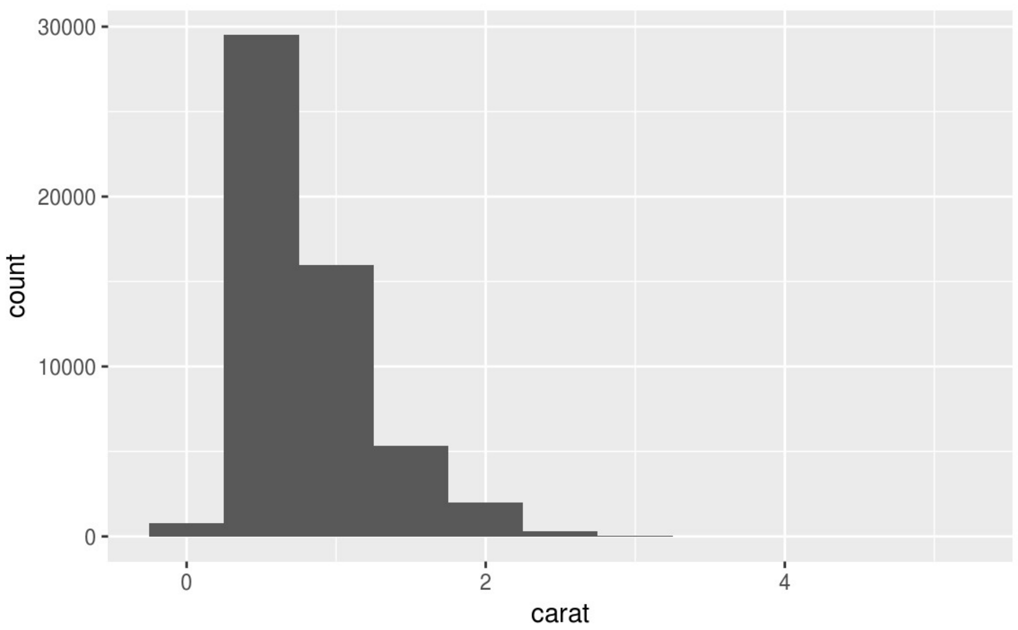
```
> diamonds
# A tibble: 53,940 x 10
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  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
```

Plot the Continuous Carats

- # To examine the distribution of a continuous variable, use a histogram
- ggplot(data = diamonds) +

  geom_histogram(mapping = aes(x = carat),

  binwidth = 0.5)



Plot the Continuous Carats

- # 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?!

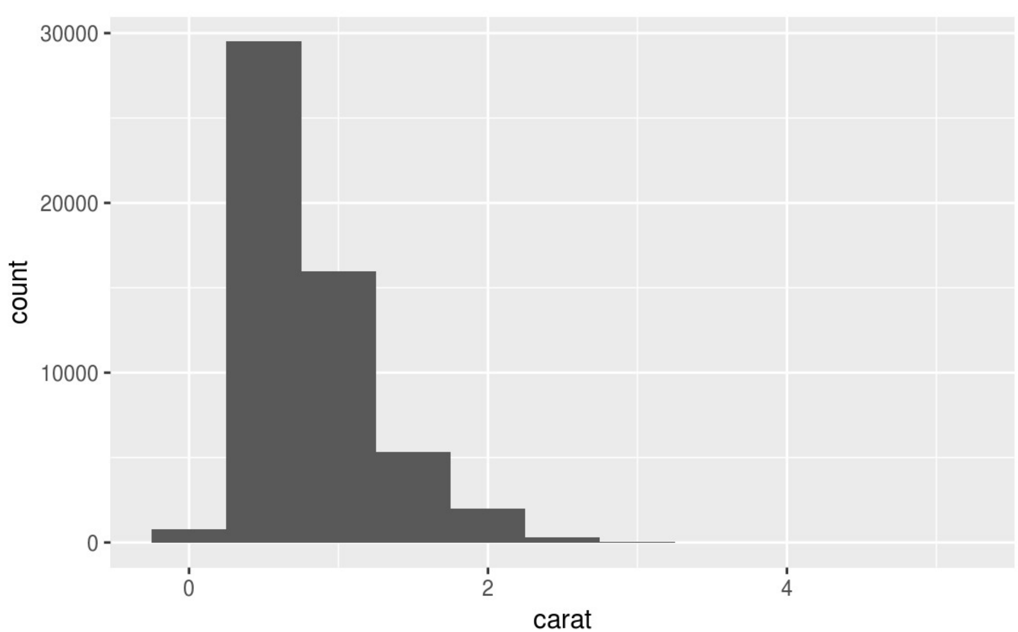
Pipe: %>% transfers one product to another function
Say, “and then”

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)

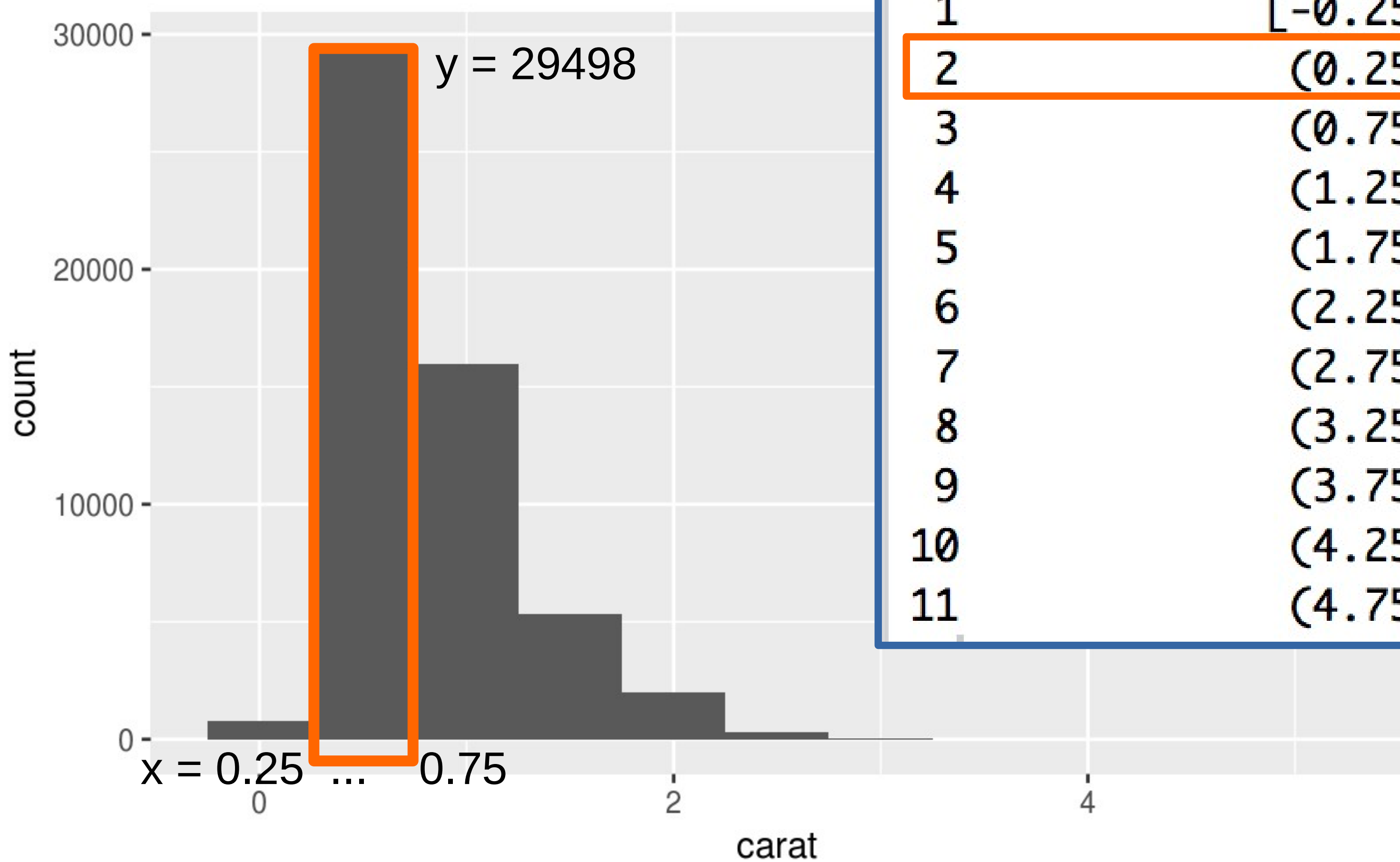



```
> diamonds %>% count(cut_width(carat,0.5))
# A tibble: 11 x 2
   `cut_width(carat, 0.5)`     n
                    <fctr> <int>
 1            [-0.25,0.25]   785
 2             (0.25,0.75] 29498
 3             (0.75,1.25] 15977
 4             (1.25,1.75]  5313
 5             (1.75,2.25]  2002
 6             (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

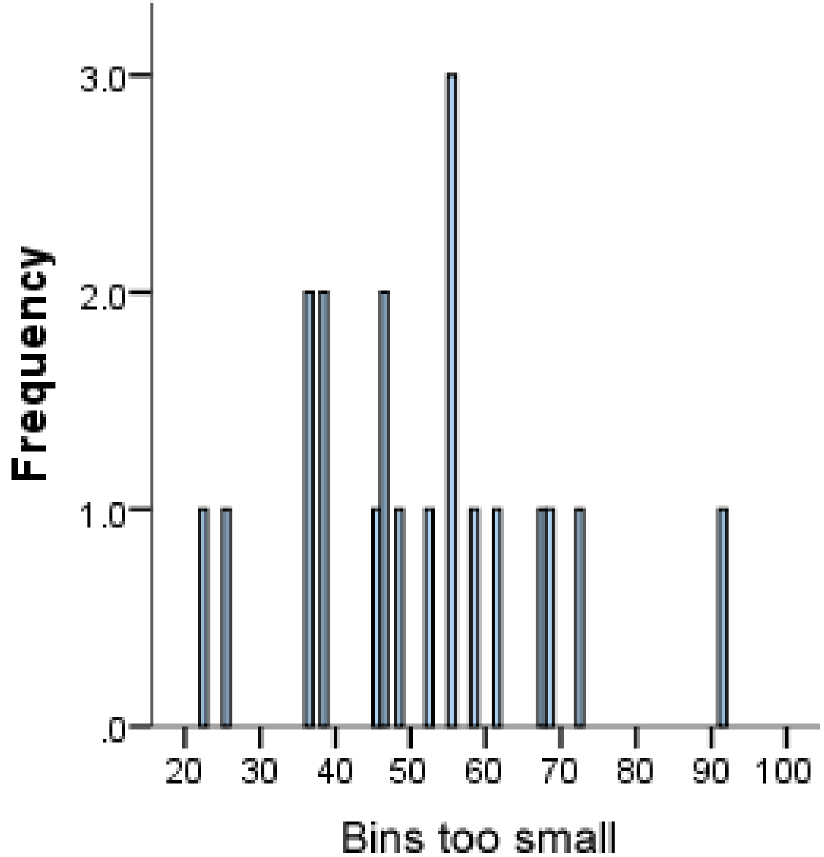
- The *cut_width()* gives a textual representation of the histogram.

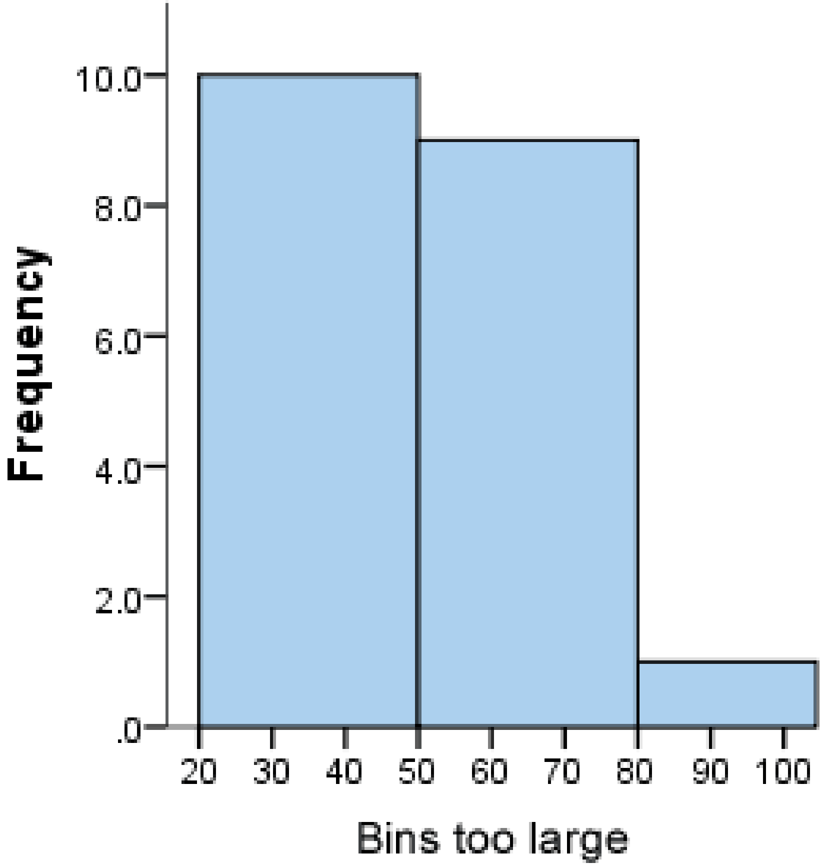
```
> diamonds %>%
+     count(cut_width(carat, 0.5))
# A tibble: 11 x 2
   `cut_width(carat, 0.5)`     n
                    <fctr> <int>
 1           [-0.25,0.25]   785
 2            (0.25,0.75] 29498
 3            (0.75,1.25] 15977
 4            (1.25,1.75]  5313
 5            (1.75,2.25]  2002
 6            (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 = ??)
```

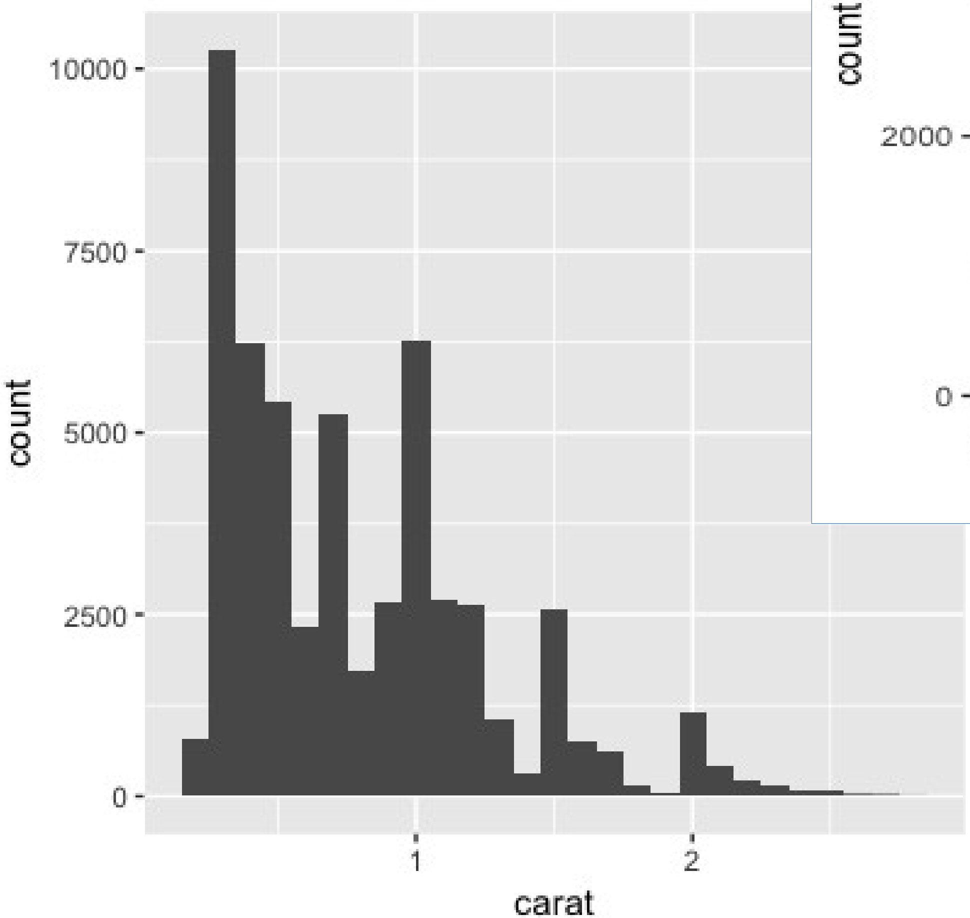
Which is the best *binwidth* setting for this data?? Why??

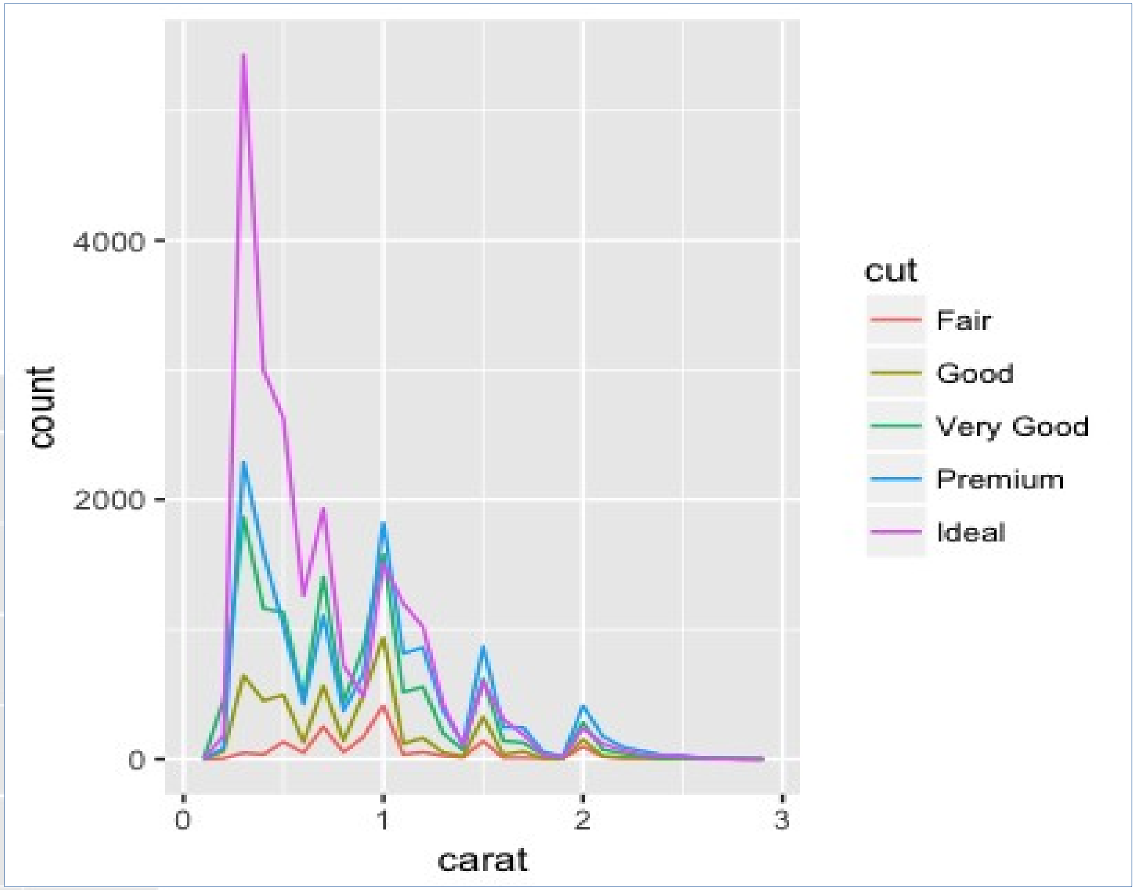
Different Bin Widths

```
# 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?

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)

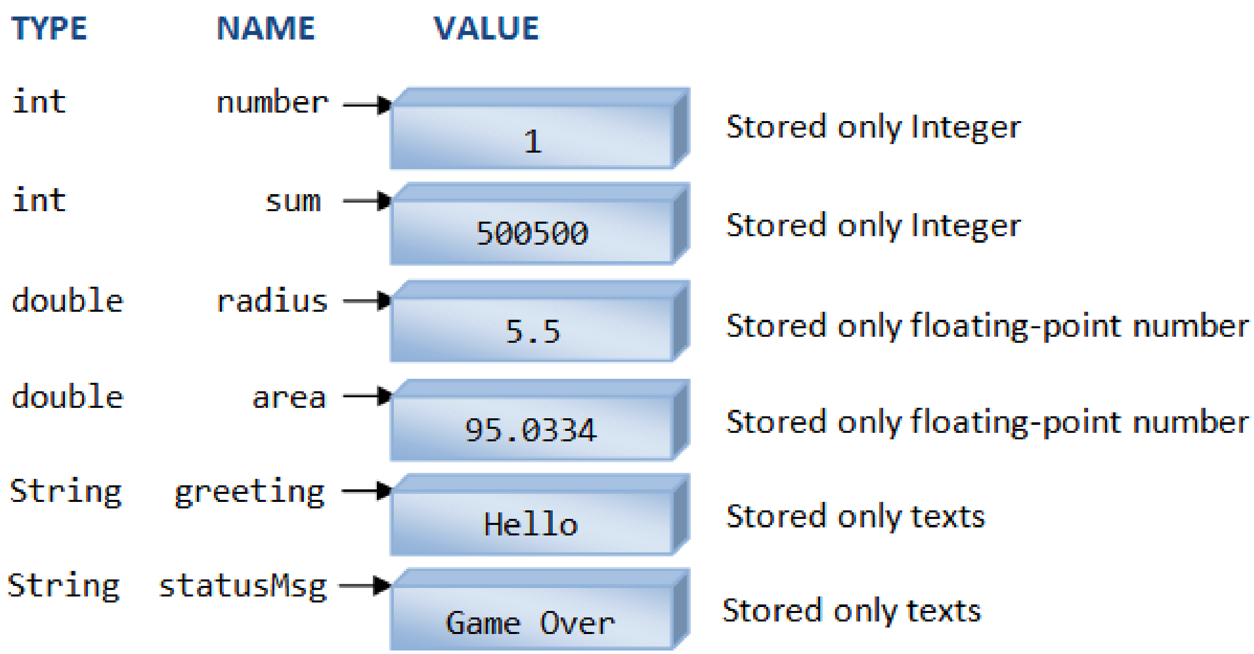
exact numbers

diamonds %>% count(cut_width(carat, 0.01))

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?

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

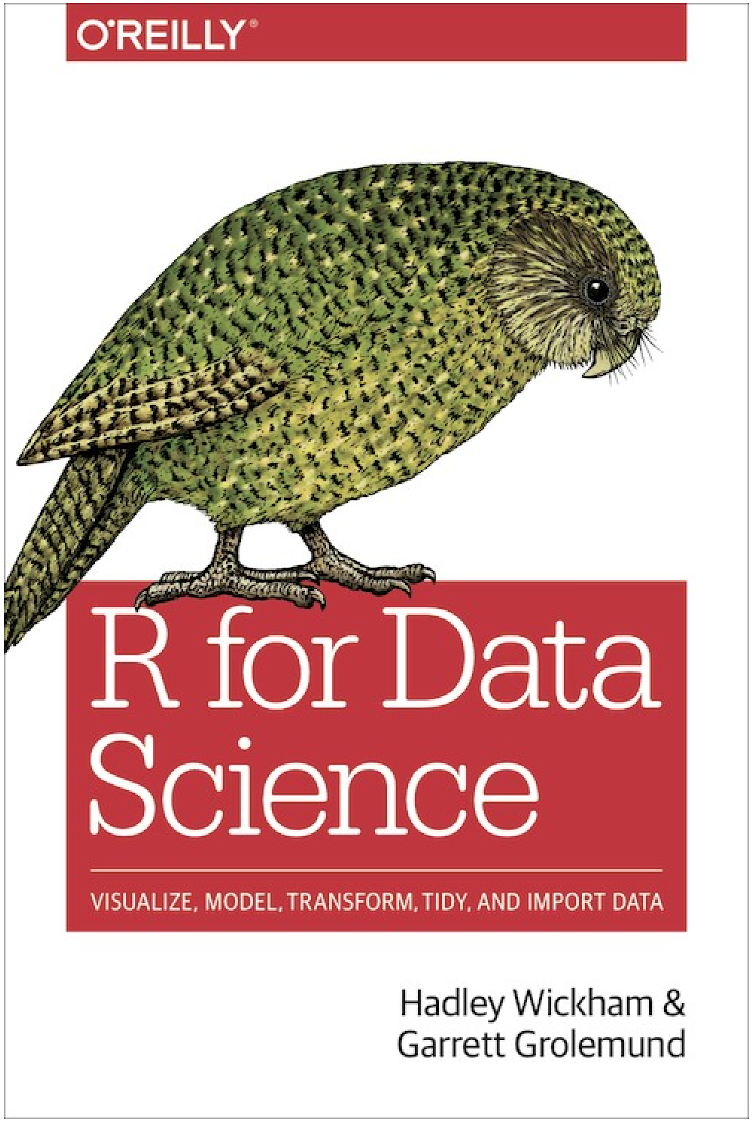
```
# Assign value of 1 to x_dbl
 x_dbl <- 1
# what type is x_dbl?
 typeof(x_dbl)

# 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
```

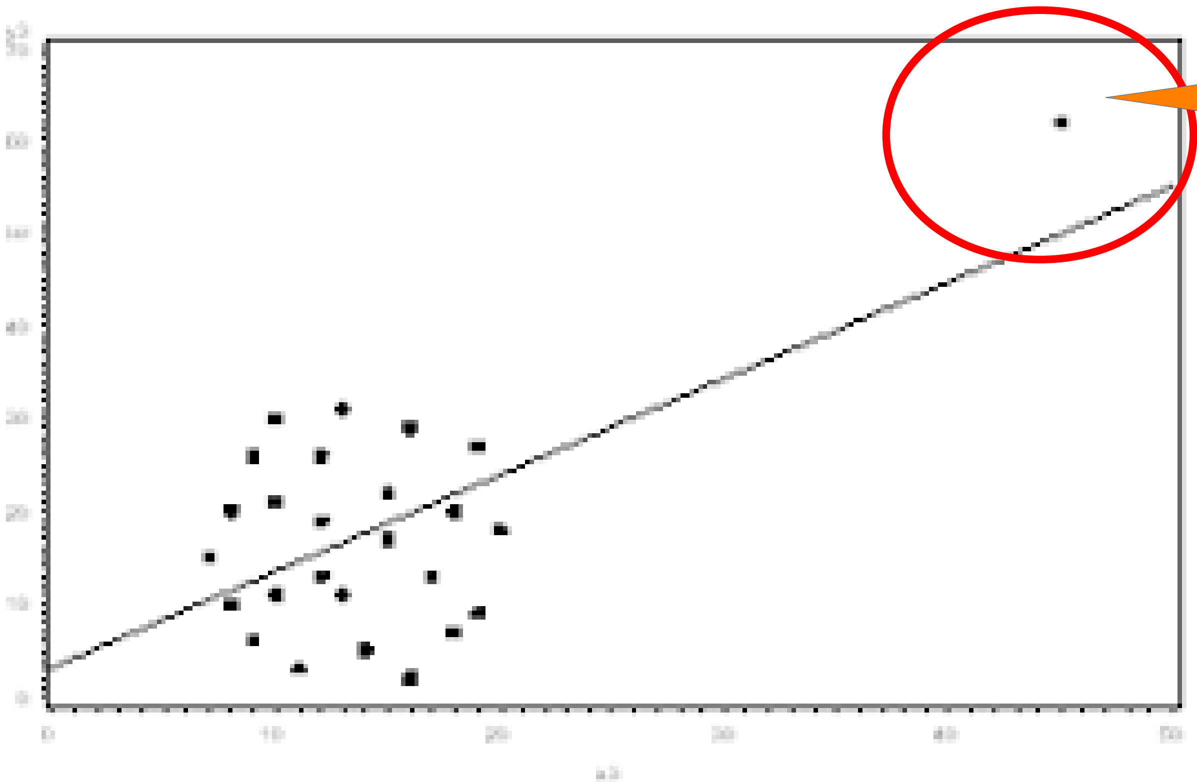
Where in the Web?
Where in the Book?



- 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

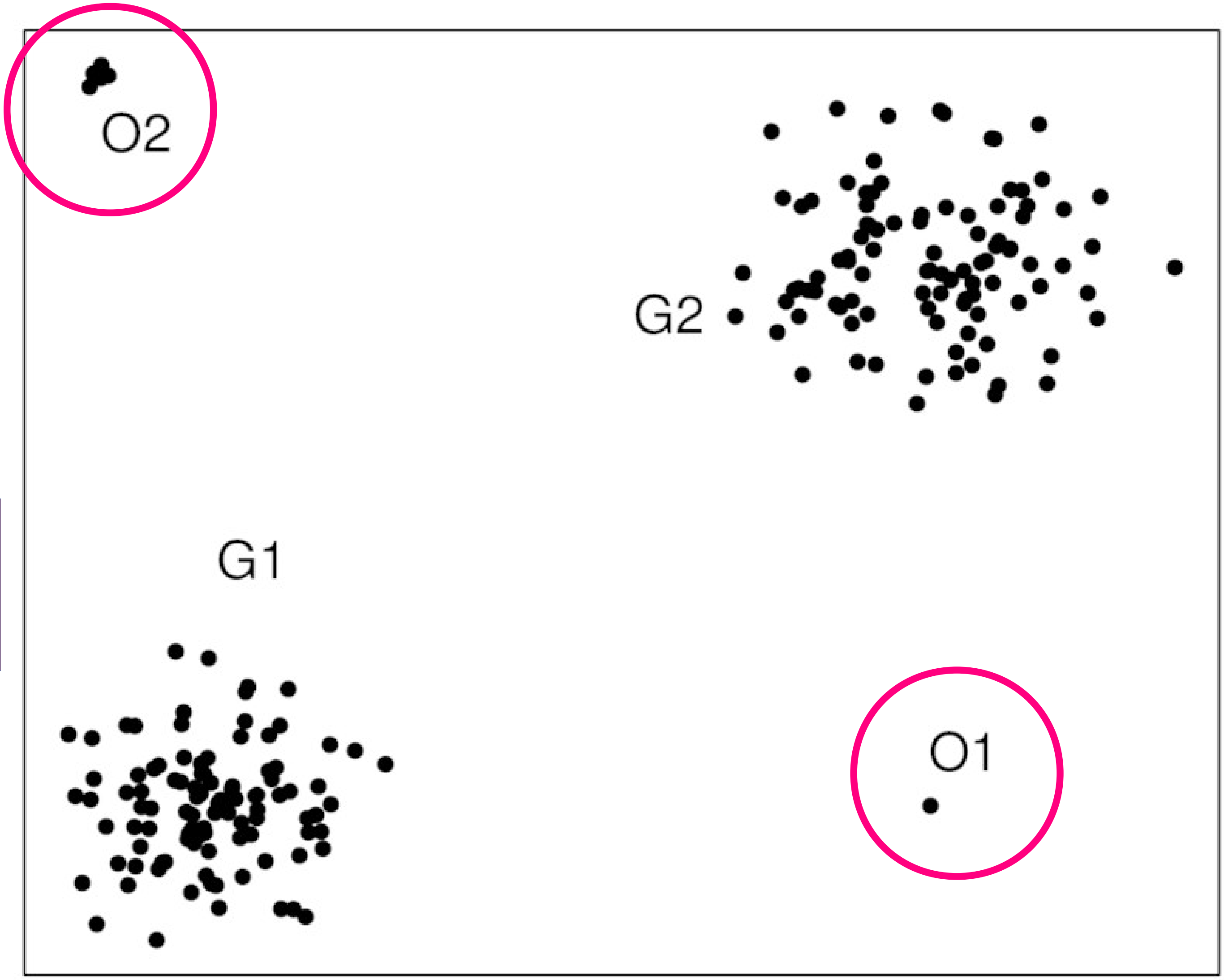
- Something that lies outside the main body or group that it is a part of, as a cow far from the rest of the herd, or a distant island belonging to a cluster of islands:



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)**)

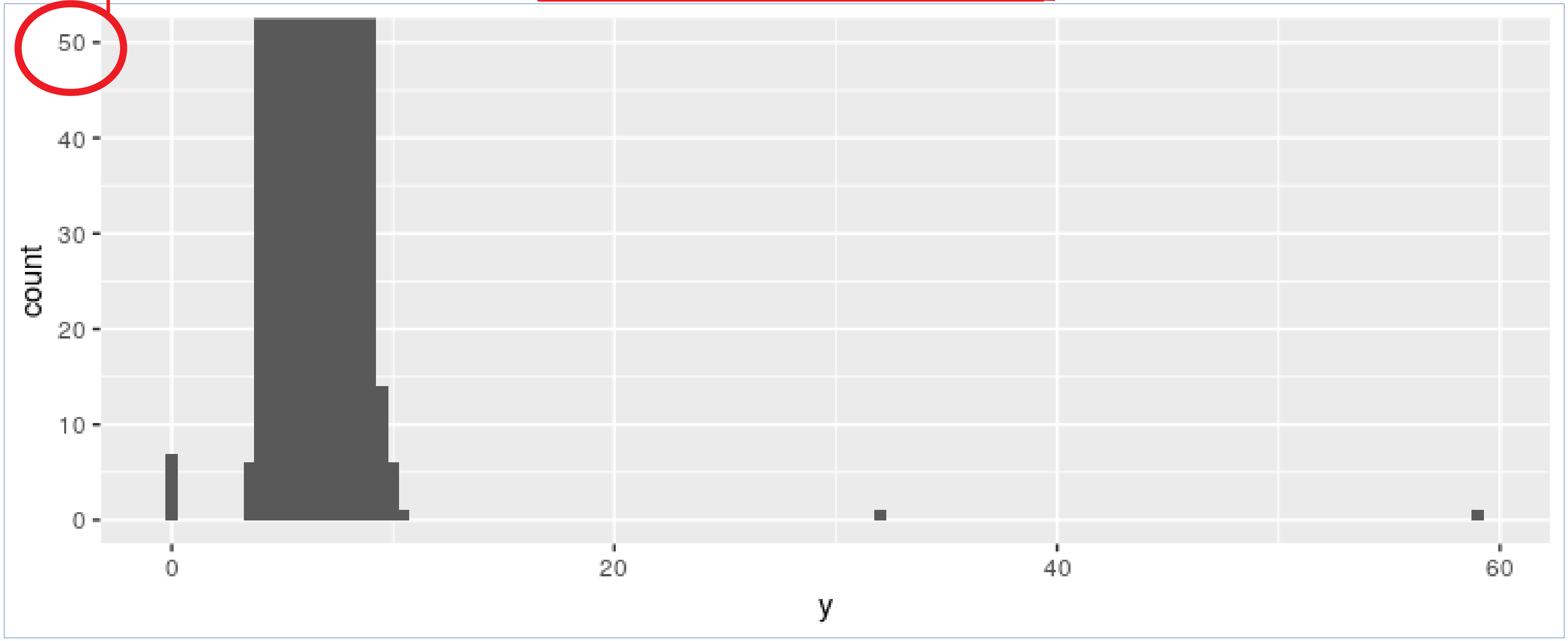
ggplot(diamonds) + geom_histogram(mapping = aes(x = y), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 20)**)

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

Data: *Diamond*

#Plot the *y* column of data.

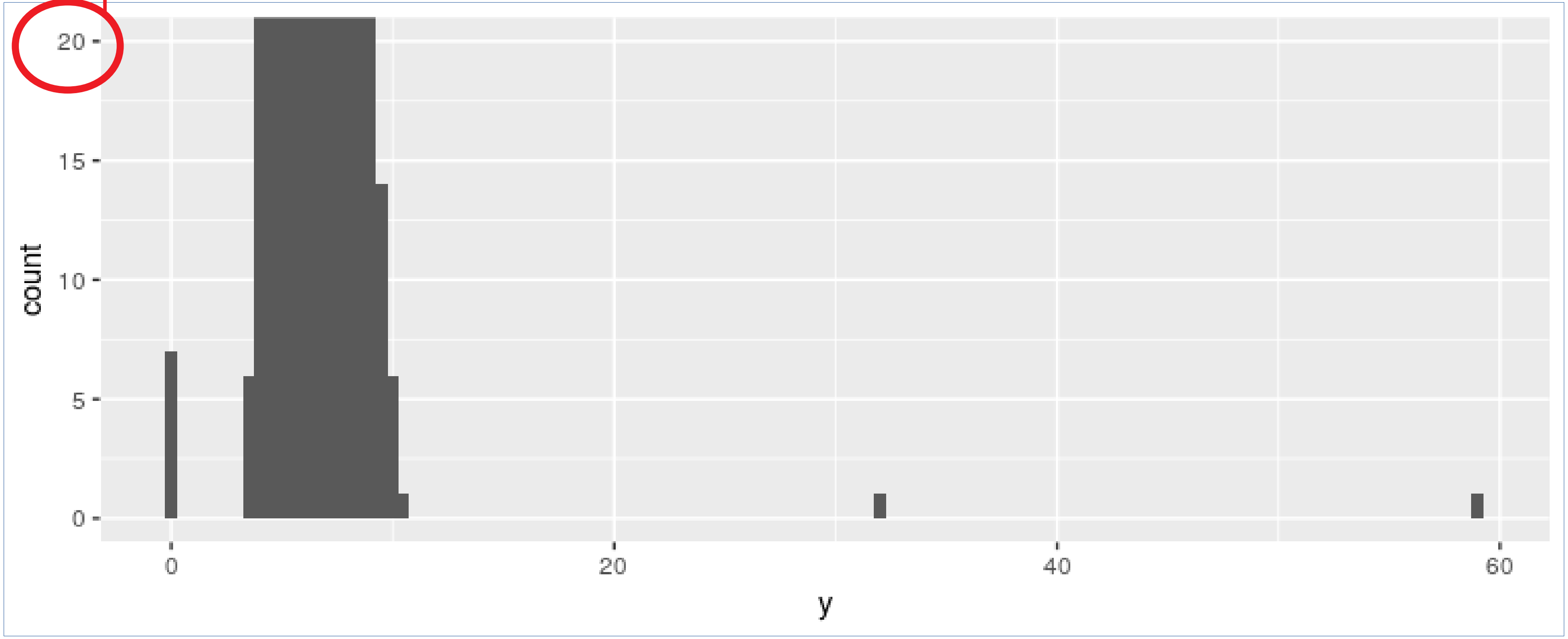
ggplot(diamonds) + geom_histogram(mapping = aes(**x = y**), binwidth = 0.5) + coord_cartesian(**ylim = c(0, 50)**)



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

```
# 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.

Missing Data Points?

Missing Data Entries

- Missing data in R appears as ***NA***.
- *NA* is not a string or a numeric value, but an indicator of missing data.
- Let's create vectors with missing values to test

```
x1 <- c(1, 4, 3, NA, 7)
x2 <- c("a", "B", NA, "NA")
is.na(x1)
is.na(x2)
```

Spot missing data

Missing Data Entries

- What to do when elements of your data go missing?
- **Why not just DROP the ENTIRE ROW??**

```
diamonds2 <- diamonds %>%  filter(between(y, 3, 20))

 View(diamonds2)

# compare to the the size of original dataset

View(diamonds)

# maybe good data was also lost that was contained
in the dropped rows.
```

Data: *Diamond*

The book recommends to *mark* the data as bad or missing.

diamonds2 <- diamonds %>%

mutate(y = **ifelse(y < 3 | y > 20, NA, y)**)

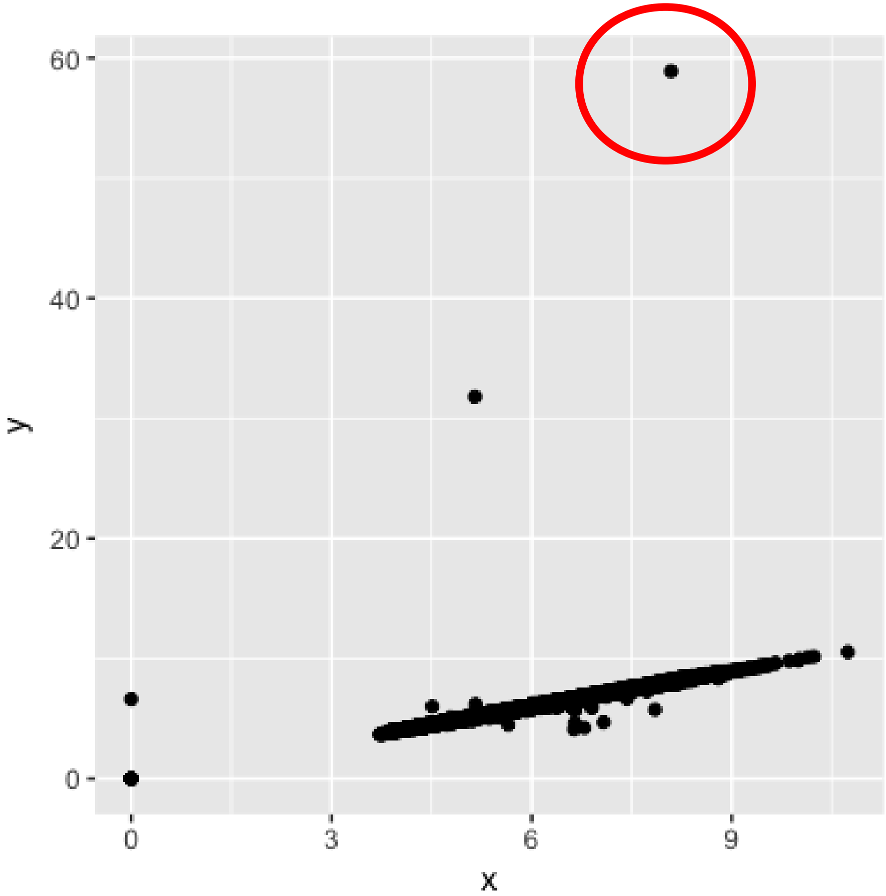
syntax: `ifelse(test, yes, no)`

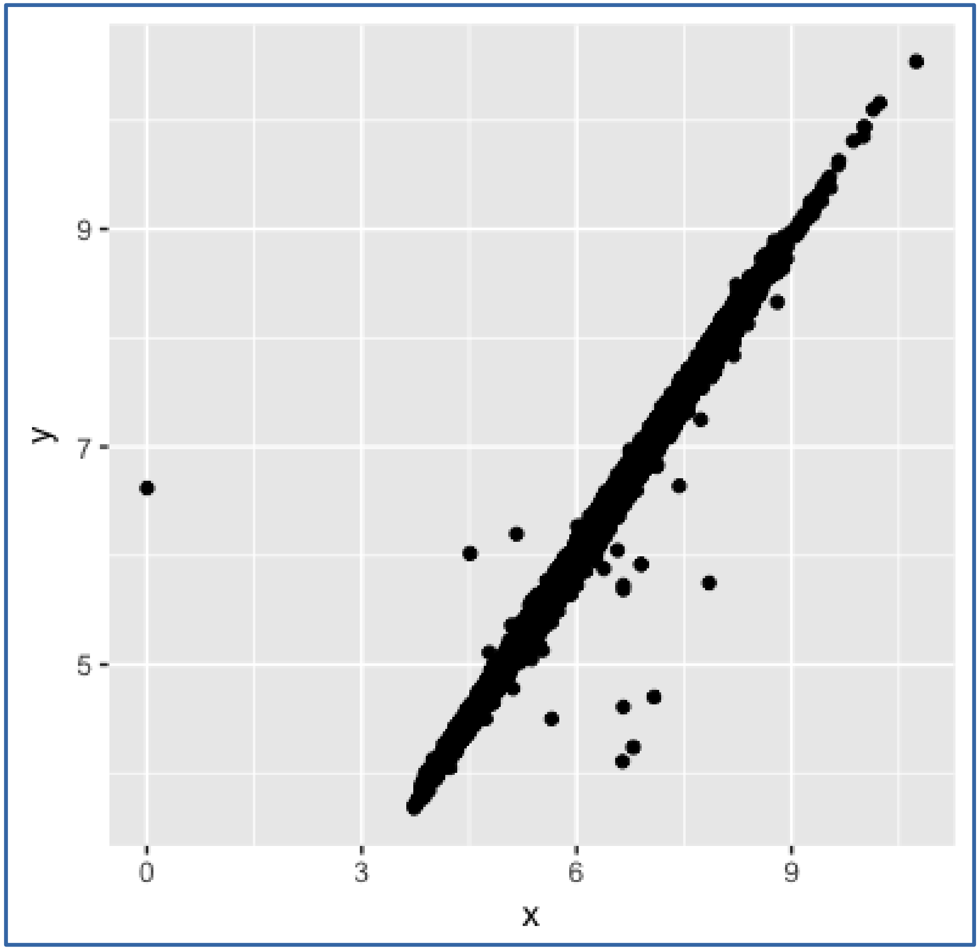
Inspect each value of *y*. If the *y* is not between 3 and 20, then *y* = NA, else *y* = *y*

We Plot All Non-NA Values

```
# Missing, outliers values marked as NA

ggplot(data = diamonds2, mapping = aes(x = x, y = y)) +  geom_point()

# compared to, no removed missing or outlier values

ggplot(data = diamonds, mapping = aes(x = x, y = y)) +  geom_point()
```

Trimmed Data, Slightly Different Plot...





- Left: WITH outliers
- Above: NO outliers

ALLEGHENY COLLEGE

Data: *Diamond*

Can you use the below code to further trim outliers or missing data?

Plot your new graphic

```
diamonds3 <- diamonds %>%
  mutate(y = ifelse(y < ## | y > ##, NA, y))
```