Slides: Importing, Quality Check and Cleansing

Actuarial Data Science Online Textbook

Fei Huang, UNSW Sydney

Table of contents

Reading List

- R for Data Science Online Book, Chapters 10, 11, 12, 7
- Applied predictive Modelling, 3.3 (only transformations to resolve outliers), 3.4

Tibbles

Tibbles

- We work with “tibbles” instead of R’s traditional data.frame in the `tidyverse` environment.
- The `tibble` package, which provides opinionated data frames that make working in the `tidyverse` a little easier.
- Try `vignette("tibble")` for more information.

```r
#install.packages("tidyverse")
library(tidyverse)
```

Creating Tibbles

- Coerce a data frame to a tibble: `as_tibble()`
- `tibble()`
- Have non-syntactic names with backticks `
- `tribble()`: transposed tibble

Example 1

```r
as_tibble(iris)
```

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1

1-10 of 150 rows | 1-4 of 5 columns   Previous 1 2 3 4 5 6 ... 15 Next

```r
tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)
```

x <int>	y <dbl>	z <dbl>
1	1	2

Example 2

```r
tibble(
  `:)` = "smile",
  ` ` = "space",
  `2000` = "number"
)
```

:) <chr>	<chr>	2000 <chr>
smile	space	number

1 row

```r
tribble(
  ~x, ~y, ~z,
  #--|--|----
  "a", 2, 3.6,
  "b", 1, 8.5
)
```

x <chr>	y <dbl>	z <dbl>
a	2	3.6
b	1	8.5

2 rows

Pipe data using %>%

- Use %>% to emphasise a sequence of actions, rather than the object that the actions are being performed on
- pronounce %>% when reading code “then”
- No need to name things
- %>% should always have a space before it, and should usually be followed by a new line.

```r
iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
  )
```

Species <int>	Sepal.Length <dbl>	Sepal.Width <dbl>
1	5.006	3.428
1	5.936	2.770
1	6.588	2.974

3 rows

Tibble v.s. data.frame

- `tibble()` does much less:
- it never changes the type of the inputs (e.g. it never converts strings to factors!),
- it never changes the names of variables, and
- it never creates row names.
- Printing
  - Tibbles show only the first 10 rows
  - each column reports its type
  - use `print()` to display more rows (`n`) and columns (`width`)
- Subsetting
  - pull out a single variable: `$` (extract by name) and `[[ ]]` (extract by name or position)
  - in a pipe `%>%`, use the special placeholder `.`

Examples

```r
df <- tibble(
  x = runif(5),
  y = rnorm(5)
)
# Extract by name
df$x
```

```
[1] 0.4539176 0.7976358 0.4242757 0.1584838 0.1950428
```

```r
df[["x"]]
```

```
[1] 0.4539176 0.7976358 0.4242757 0.1584838 0.1950428
```

```r
# Extract by position
df[[1]]
```

```
[1] 0.4539176 0.7976358 0.4242757 0.1584838 0.1950428
```

```r
df %>% .$x
```

```
[1] 0.4539176 0.7976358 0.4242757 0.1584838 0.1950428
```

```r
df %>% .[["x"]]
```

```
[1] 0.4539176 0.7976358 0.4242757 0.1584838 0.1950428
```

Import Data

Import Data

- Read The Art of Data Science, Chapter 5
- Read R for Data Science, Chapter 11

Tidy Data

Tidy Data

- The same data can be organised in different ways.
- The tidy data is easy to work with.
- There are three interrelated rules which make a dataset tidy:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each value must have its own cell.
- Practical instructions:
  - Put each dataset in a tibble.
  - Put each variable in a column.
- `dplyr`, `ggplot2`, and all the other packages in the tidyverse are designed to work with tidy data

```r
library(tidyverse)
```

Pivoting: Longer

- A common problem is a dataset where some of the column names are not names of variables, but values of a variable.
- Example: `table4a`: the column names 1999 and 2000 represent values of the year variable, the values in the 1999 and 2000 columns represent values of the cases variable, and each row represents two observations, not one.

```r
#table4a

table4a %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

country <chr>	year <chr>	cases <dbl>
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

6 rows

Pivoting: Wider

- `pivot_wider()` is the opposite of `pivot_longer()`.
- You use it when an observation is scattered across multiple rows.
- For example, take `table2`: an observation is a country in a year, but each observation is spread across two rows.

```r
table2 %>%
    pivot_wider(names_from = type, values_from = count)
```

country <chr>	year <dbl>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Separating

- `table3` has a different problem: we have one column (rate) that contains two variables (cases and population).
- `separate()` pulls apart one column into multiple columns, by splitting wherever a separator character appears.
- By default, `separate()` will split values wherever it sees a non-alphanumeric character (i.e. a character that isn't a number or letter).
- Use `sep` argument to use a specific character to separate a column

::: {.cell}

```
table3
```

::: {.cell-output-display}

country <chr>	year <dbl>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360

Example: Separating

```r
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

country <chr>	year <dbl>	cases <chr>	population <chr>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

```r
#table3 %>%
#  separate(rate, into = c("cases", "population"), sep = "/", convert=TRUE)

#table3 %>%
#  separate(year, into = c("century", "year"), sep = 2)
```

Unite

- `unite()` is the inverse of separate(): it combines multiple columns into a single column.
- The default will place an underscore (_) between the values from different columns. Here we don't want any separator so we use "“".

```r
table5 %>%
  unite(new, century, year)
```

country <chr>	new <chr>	rate <chr>
Afghanistan	19_99	745/19987071
Afghanistan	20_00	2666/20595360
Brazil	19_99	37737/172006362
Brazil	20_00	80488/174504898
China	19_99	212258/1272915272
China	20_00	213766/1280428583

6 rows

```r
#table5 %>%
#  unite(new, century, year, sep = "")
```

Missing Value

- A value can be missing in one of two possible ways:
  - Explicitly, i.e. flagged with NA. – *the presence of an absence*
  - Implicitly, i.e. simply not present in the data. – *the absence of a presence*

```r
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

Making implicit missing values explicit

- using `pivot_wider`
- using `complete()`

```r
stocks %>%
  pivot_wider(names_from = year, values_from = return)
```

qtr <dbl>	2015 <dbl>	2016 <dbl>
1	1.88	NA
2	0.59	0.92
3	0.35	0.17
4	NA	2.66

4 rows

```r
#stocks %>%
#  complete(year, qtr)
```

Making explicit missing values implicit

- Set `values_drop_na = TRUE` in `pivot_longer()` to turn explicit missing values implicit

```r
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(
    cols = c(`2015`, `2016`),
    names_to = "year",
    values_to = "return",
    values_drop_na = TRUE
  )
```

qtr <dbl>	year <chr>	return <dbl>
1	2015	1.88
2	2015	0.59
2	2016	0.92
3	2015	0.35
3	2016	0.17
4	2016	2.66

6 rows

Fill the missing values with `fill()`

- `fill()` takes a set of columns where you want missing values to be replaced by the most recent non-missing value (sometimes called last observation carried forward).

```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)

treatment %>%
  fill(person)
```

person <chr>	treatment <dbl>	response <dbl>
Derrick Whitmore	1	7
Derrick Whitmore	2	10
Derrick Whitmore	3	9
Katherine Burke	1	4

4 rows

Assessing data quality

Data issues

- Missing data
- Irregular data and outliers
- Uninformative data
- Censored and truncated data
- High cardinality features
- Imbalanced data

Diagnosing missing data

- Descriptive statistics
- Plots
- and

Missing data

- Understand why the values are missing
- Structurally missing or not
  - e.g. the number of children a man has given birth to
- Informative missingness: the pattern of missing data is related to outcome
  - e.g. in a drug study, the side effect so bad that the patients drop out

Handling missing values

- include a missing indicator (dummy variable)
  - if the pattern of missingness is informative
- some models can account for missing data, such as tree-based techniques
- many models cannot tolerate missing values
  - linear models, neural networks, SVMs
- remove the observations or variables as a last resort
  - may be feasible for large dataset
- impute missing values

Imputing missing values

- use the information in the training set predictors to estimate the values of other predictors
  - via mean, median or mode
  - via model-based, such as K-nearest neighbor model
- extensively studied in the statistical literature in terms inference; not a big concern for predictive modelling

Irregular data/Outliers

- Detection
  - descriptive statistics
  - plots, such as boxplot, scatter plot
  - outlier detection models
- Handling
  - data validation, make sure no recording errors
  - remove or change values
  - outliers might belong to a different population than the other samples
  - models resistant to outliers, e.g. tree-based classification methods, SVM
  - transformations to minimise the problem using *spatial sign* – each sample is divided by its squared norm.

Example:

Uninformative data

- repetitive
- duplicates
- irrelevant
- collinearity: a pair of predictor variables have a substantial correlation with each other
    - redundant predictors add more complexity
    - results in highly unstable models, numerical errors and degraded predictive performance
    - For linear regression, models such as variance inflation factor (VIF) can be used to identify predictors that are impacted

Censored data

- The value of an observation is only partially known
- For interpretation or inference
  - usually treated in a formal manner by making assumptions about the censoring mechanism
- For prediction
  - usually treated as missing data or use the censored value as observed value

Examples: general insurance (policy limits); life insurance (age groups of mortality data)

High cardinality features

- Categorical predictors with many unique factor levels
- High cardinality features (eg. post codes, medical condition coding or similar)

Some references for dealing with high cardinality features:

- Dealing with features that have high cardinality
- Encoding High-Cardinality String Categorical Variables
- Similarity Encoding for Learning with Dirty Categorical Variables
- Nonlife Insurance Risk Classification Using Categorical Embedding
- Using Random Effects to Account for High-Cardinality Categorical Features and Repeated Measures in Deep Neural Networks

Imbalanced data

- Imbalance between control and treatment observations can cause modelling problems
- Construct a balanced training set to improve modelling outcomes for imbalanced data
  - Undersampling: reduce the number of patterns within the majority class data set to make it equivalent to other classes
  - Oversampling: generate more data within the minority class

Data validation

- Validate data against other sources and the same data from previous runs
- Talk to people who input and use the data to assess data quality

Exploratory Data Analysis (EDA)

Exploratory Data Analysis

- Generate questions about your data.
- Search for answers by visualising, transforming, and modelling your data.
- Use what you learn to refine your questions and/or generate new questions.
- We'll combine `dplyr` and `ggplot2` to interactively ask questions, answer them with data, and then ask new questions.
- Two types of questions will always be useful for making discoveries within your data:
  - What type of variation occurs within my variables?
  - What type of covariation occurs between my variables?

```r
library(tidyverse)
```

Variation

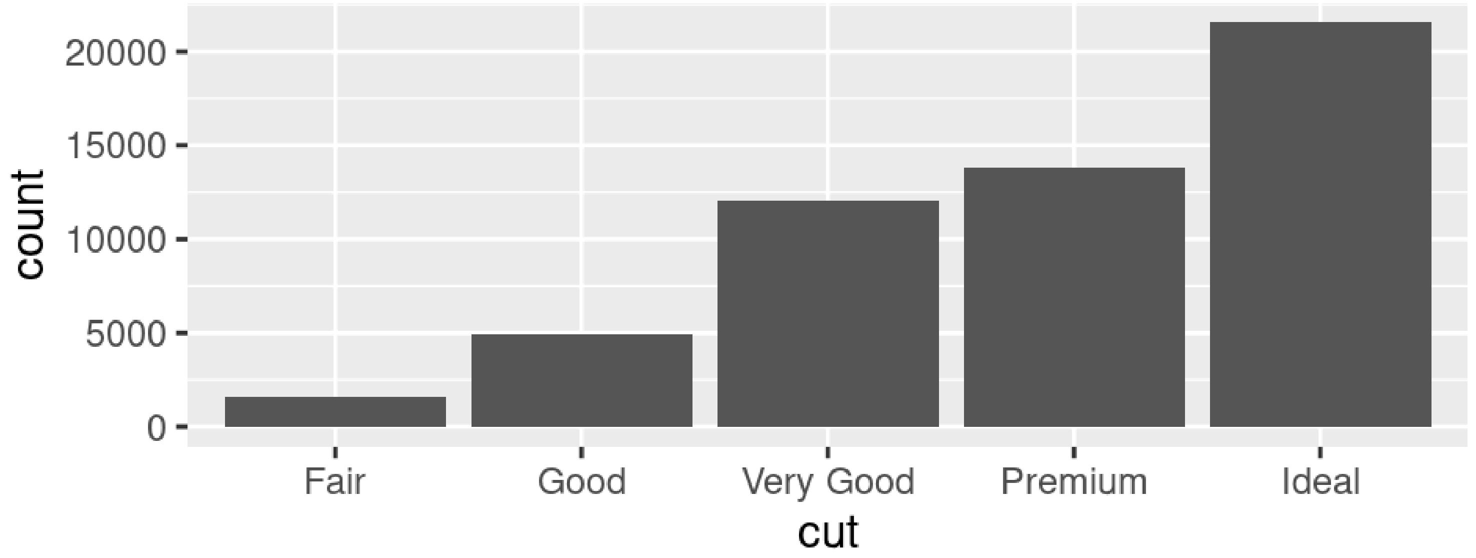
- Variation is the tendency of the values of a variable to change from measurement to measurement.
- Variables:
  - continuous variable: if it can take any of an infinite set of ordered values
  - categorical variable: if it can only take one of a small set of values.

Visualising Distributios: Categorical Variable

- To examine the distribution of a categorical variable, use a bar chart.
- Data: `diamonds`. Check `?diamonds` for more information of the dataset.

```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```

```r
#diamonds %>%
#  count(cut)
```

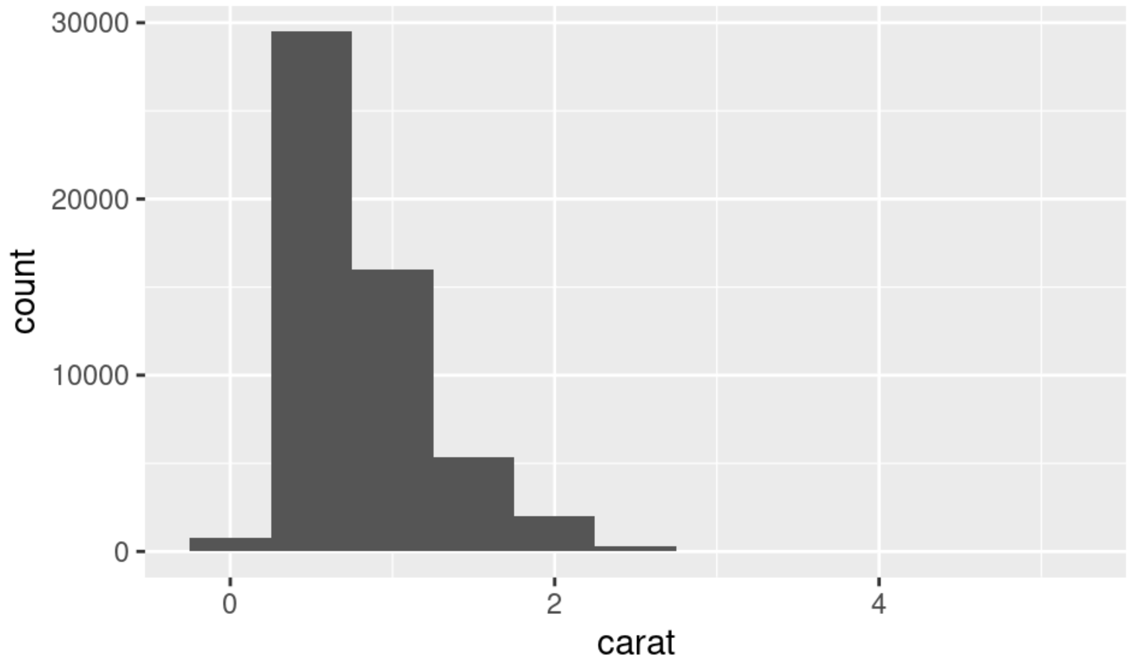


Visualising Distributions: Continuous Variable

- To examine the distribution of a continuous variable, use a histogram.

```r
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat), binwidth = 0.5)
```

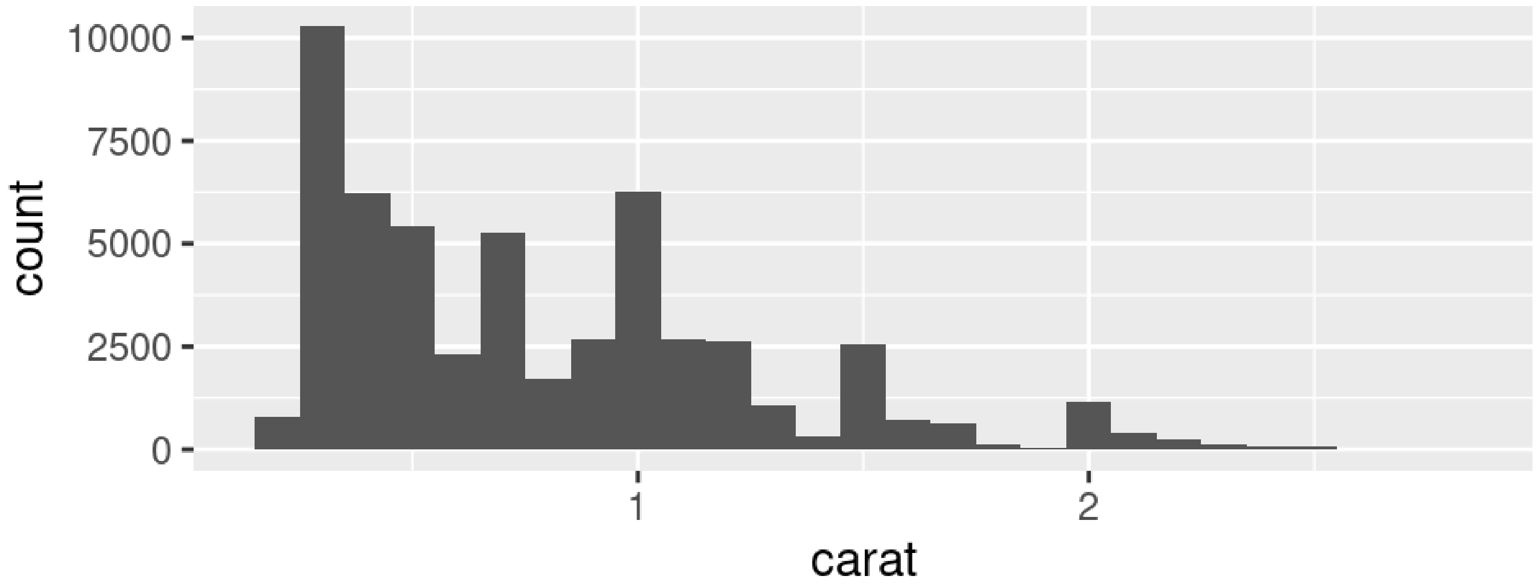
```r
#diamonds %>%
#  count(cut_width(carat, 0.5))
```



Exercise 1

- Plot the histogram of the diamonds with a size of less than 3 carats (using `filter`) and choose a smaller binwidth of 0.1.

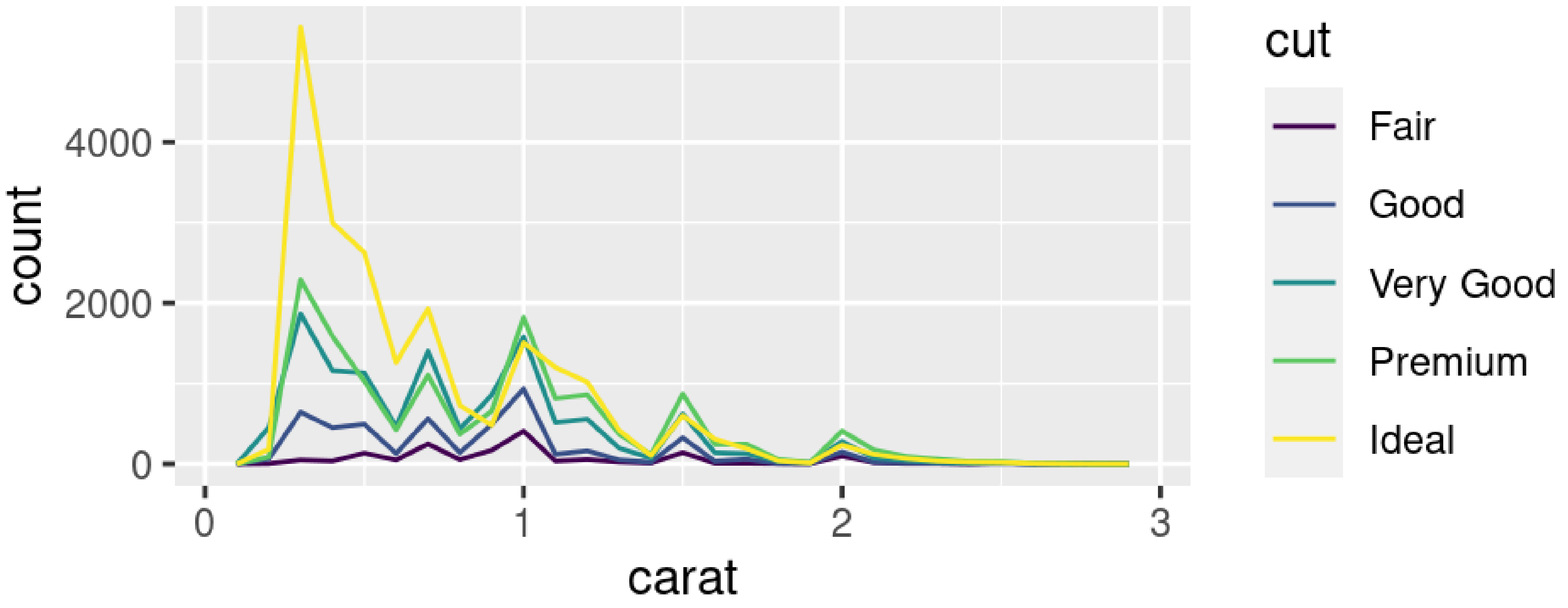
```r
smaller <- diamonds %>%
  filter(carat < 3)

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



Exericese 2

- Overlay multiple histograms in the same plot by `cut` using `geom_freqpoly()`

```r
ggplot(data = smaller, mapping = aes(x = carat, colour = cut)) +
  geom_freqpoly(binwidth = 0.1)
```



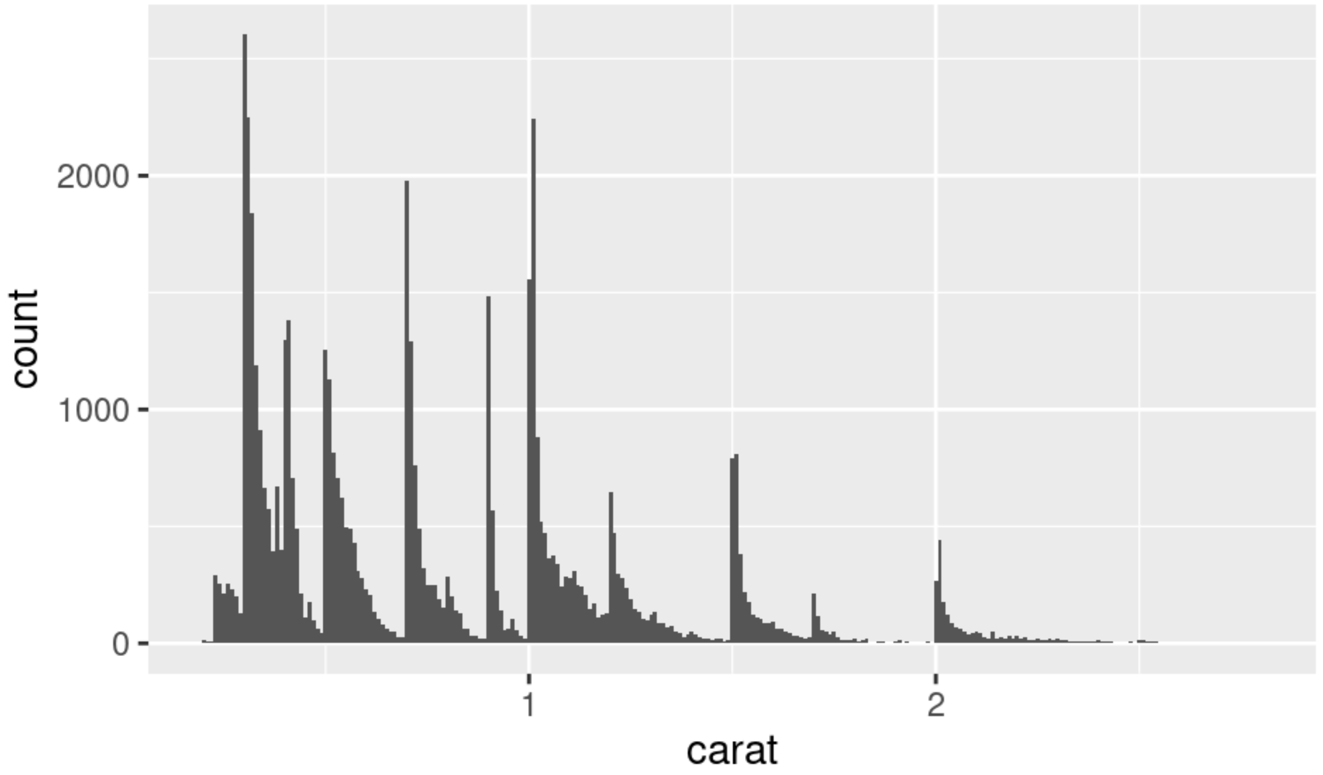
Typical Values

- In both bar charts and histograms, tall bars show the common values of a variable, and shorter bars show less-common values. Places that do not have bars reveal values that were not seen in your data.
- To turn this information into useful questions, look for anything unexpected:
  - Which values are the most common? Why?
  - Which values are rare? Why? Does that match your expectations?
  - Can you see any unusual patterns? What might explain them?

Example

- Look at the histogram below, what questions can you ask?

```r
ggplot(data = smaller, mapping = aes(x = carat)) +
  geom_histogram(binwidth = 0.01)
```



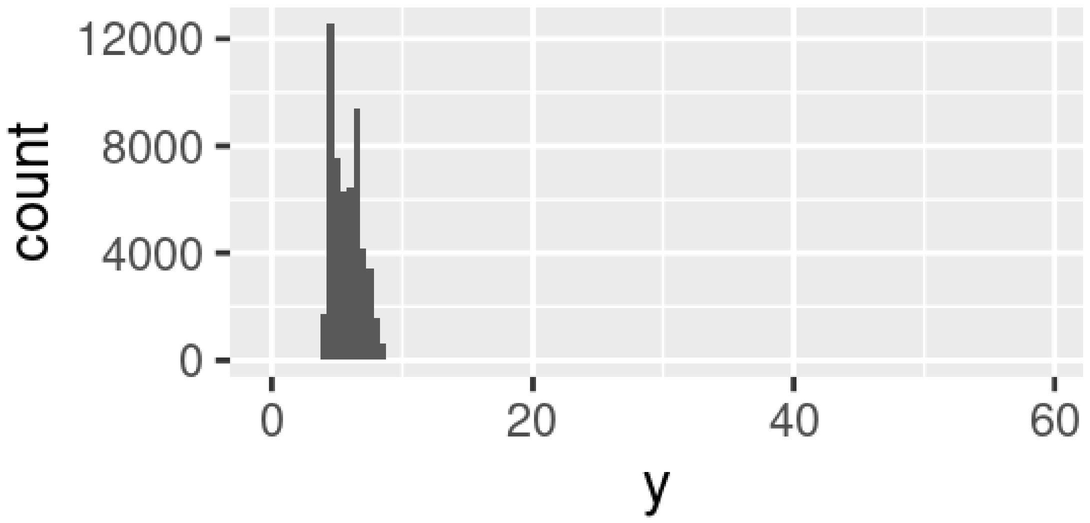
Example: Solution

- Why are there more diamonds at whole carats and common fractions of carats?
- Why are there more diamonds slightly to the right of each peak than there are slightly to the left of each peak?
- Why are there no diamonds bigger than 3 carats?

Unusual Values (Outliers)

- Outliers are observations that are unusual; data points that don't seem to fit the pattern.
- Sometimes outliers are data entry errors; other times outliers suggest important new science.
- When you have a lot of data, outliers are sometimes difficult to see in a histogram.

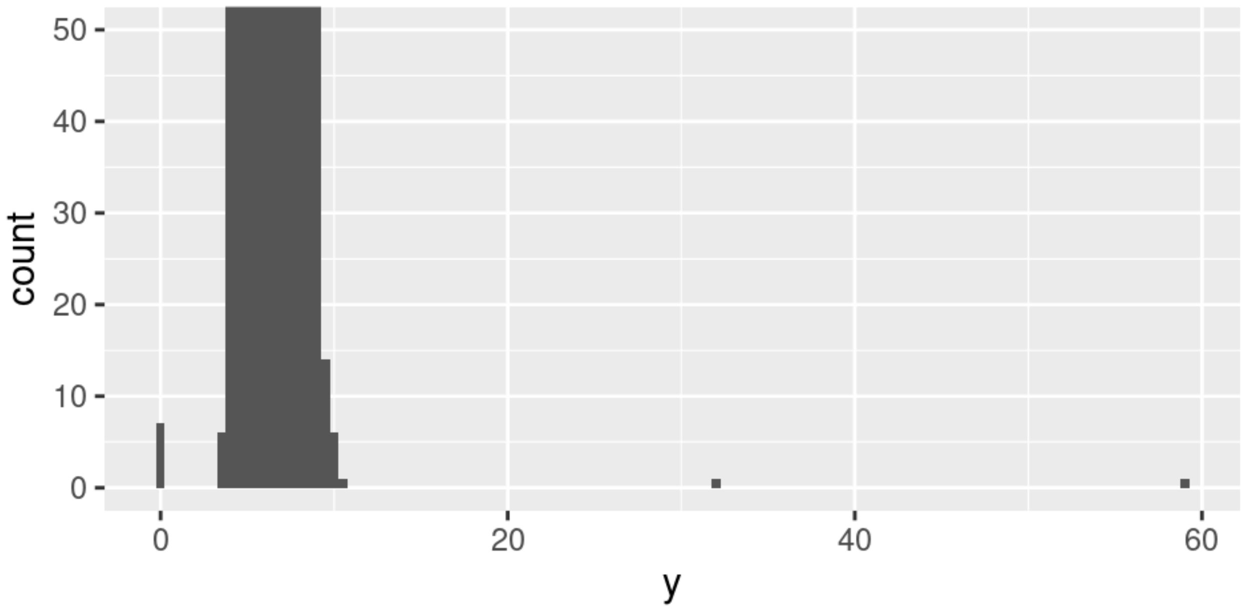
```r
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = y), binwidth = 0.5)
```



Visualising Outliers

- To make it easy to see the unusual values, we need to zoom to small values of the y-axis with `coord_cartesian()` and `ylim()` or `xlim()` to zoom into the y-axis or x-axis.

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



Display all the unusual values

- We pluck them out with `dplyr`.
- What questions you may have?

```r
unusual <- diamonds %>%
  filter(y < 3 | y > 20) %>%
  select(price, x, y, z) %>%
  arrange(y)
unusual
```

price <int>	x <dbl>	y <dbl>	z <dbl>
5139	0.00	0.0	0.00
6381	0.00	0.0	0.00
12800	0.00	0.0	0.00
15686	0.00	0.0	0.00
18034	0.00	0.0	0.00
2130	0.00	0.0	0.00
2130	0.00	0.0	0.00
2075	5.15	31.8	5.12
12210	8.09	58.9	8.06

9 rows

Deal with Outliers: Example

- In the ‘Diamond’ example, the y variable measures one of the three dimensions of these diamonds, in mm.
- We know that diamonds can’t have a width of 0mm, so these values must be incorrect.
- We might also suspect that measurements of 32mm and 59mm are implausible: those diamonds are over an inch long, but don’t cost hundreds of thousands of dollars!

Deal with Outliers

- Repeat your analysis with and without the outliers.
- If they have minimal effect on the results and you can't figure out why they're there, it's reasonable to replace them with missing values and move on.
- If they have a substantial effect on your results, you shouldn't drop them without justification. You'll need to figure out what caused them (e.g. a data entry error) and disclose that you removed them in your write-up.

Missing Values

- If you’ve encountered unusual values in your dataset, and simply want to move on to the rest of your analysis, you have two options.
  - Drop the entire row with the strange values (not recommend, why?)

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

- Replace the unusual values with missing values (`NA`) using `mutate()` with `ifelse()` or `case_when()`

```r
diamonds2 <- diamonds %>%
  mutate(y = ifelse(y < 3 | y > 20, NA, y))
#diamonds2
```

- `ggplot2` doesn’t include them in the plot, but it does warn that they’ve been removed.

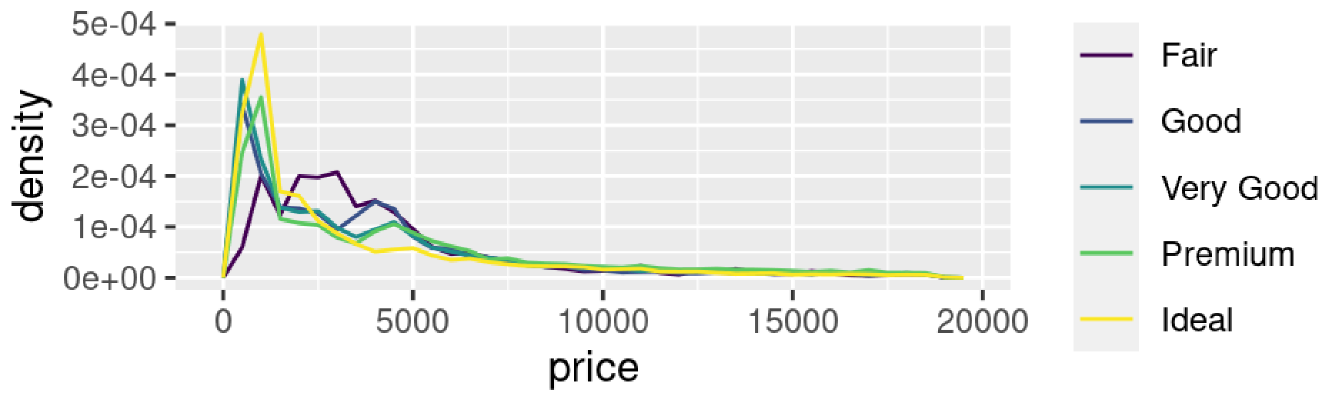
Covariation

- If variation describes the behavior within a variable, covariation describes the behavior between variables.
- Covariation is the tendency for the values of two or more variables to vary together in a related way.
- The best way to spot covariation is to visualise the relationship between two or more variables.
- We consider three different combinations
  - A categorical and continuous variable
  - Two categorical variables
  - Two continuous variables

A Categorical and Continuous Variable

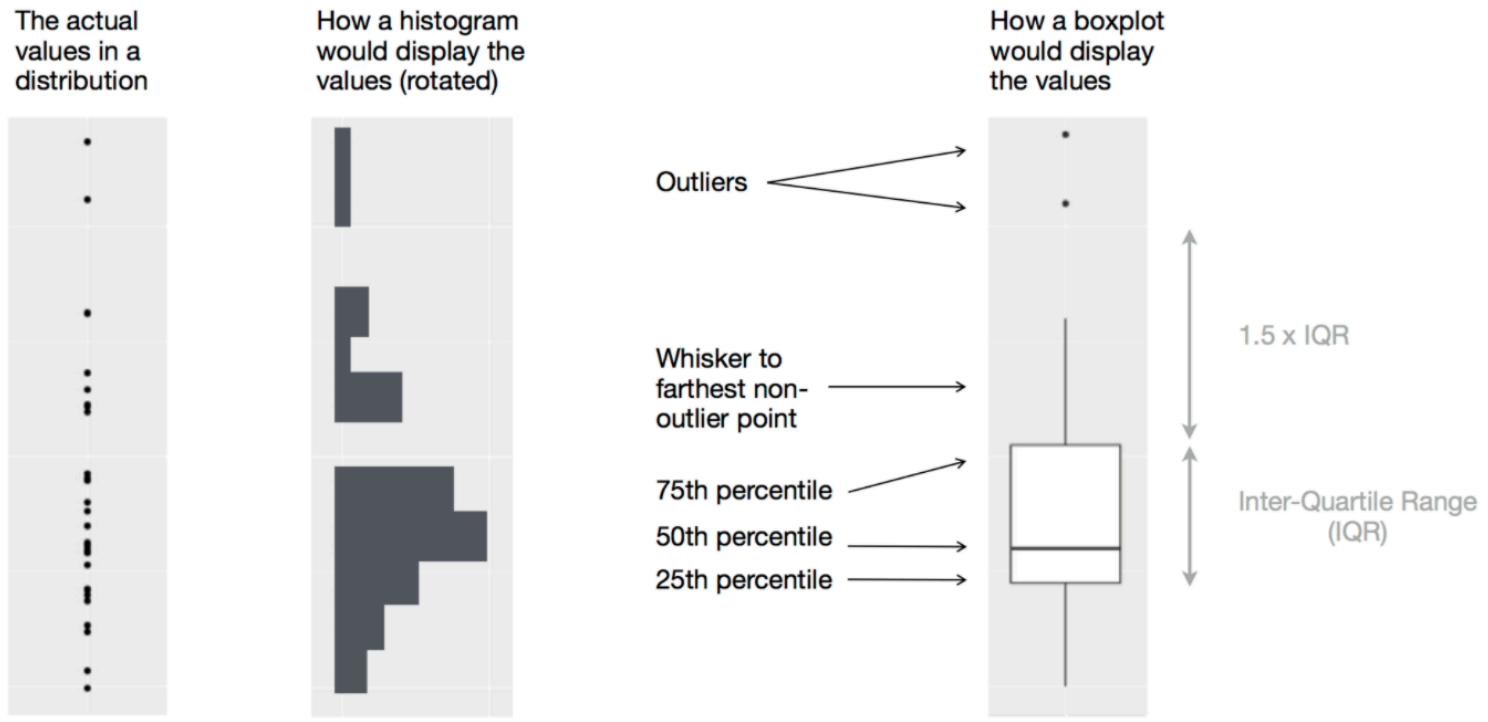
- Explore the distribution of a continuous variable broken down by a categorical variable
- using `geom_freqpoly()`, for example see
- It's hard to see the difference in distribution because the overall counts differ so much, see .
- Instead of displaying count, we'll display density, which is the count standardised so that the area under each frequency polygon is one.

```r
ggplot(data = diamonds, mapping = aes(x = price, y = ..density..)) +
  geom_freqpoly(mapping = aes(colour = cut), binwidth = 500)
```



Boxplot

- Another alternative to display the distribution of a continuous variable broken down by a categorical variable is the **boxplot**.

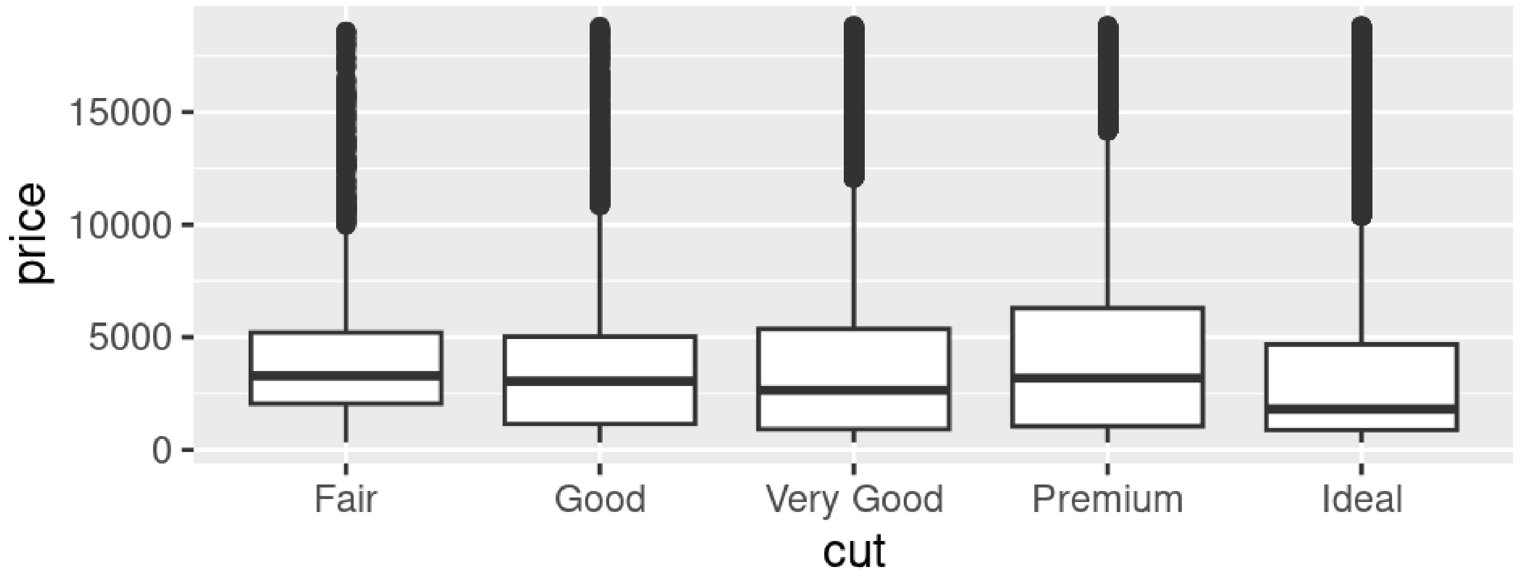

The actual values in a distribution
How a histogram would display the values (rotated)
How a boxplot would display the values
Outliers
Whisker to farthest non-outlier point
75th percentile
50th percentile
25th percentile
1.5 x IQR
Inter-Quartile Range (IQR)

Boxplot, source: R for Data Science

Example 1

- Take a look at the distribution of diamond price by cut. What can you find?

```r
ggplot(data = diamonds, mapping = aes(x = cut, y = price)) +
  geom_boxplot()
```



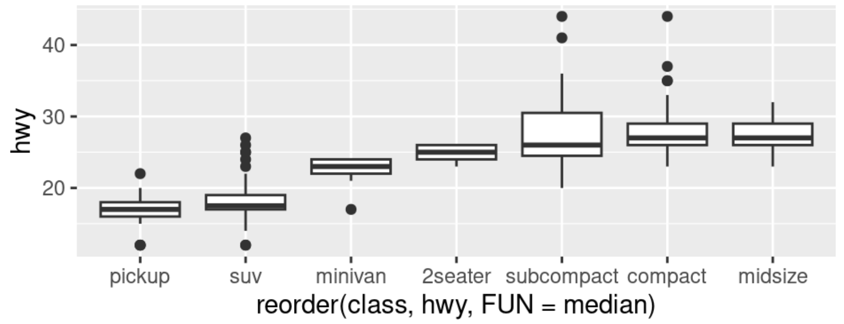
- It supports the counterintuitive finding that better quality diamonds are cheaper on average! Why?

Example 2

- Look at the `mpg` dataset. We are interested to know how highway mileage (`hwy`) varies across classes (`class`)
- To make the trend easier to see, we can reorder (`reorder`) class based on the median value (`FUN=median`) of hwy.
- If you have long variable names, `geom_boxplot()` will work better if you flip it 90°. You can do that with `coord_flip()`

```r
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = reorder(class, hwy, FUN = median), y = hwy)) #+
```

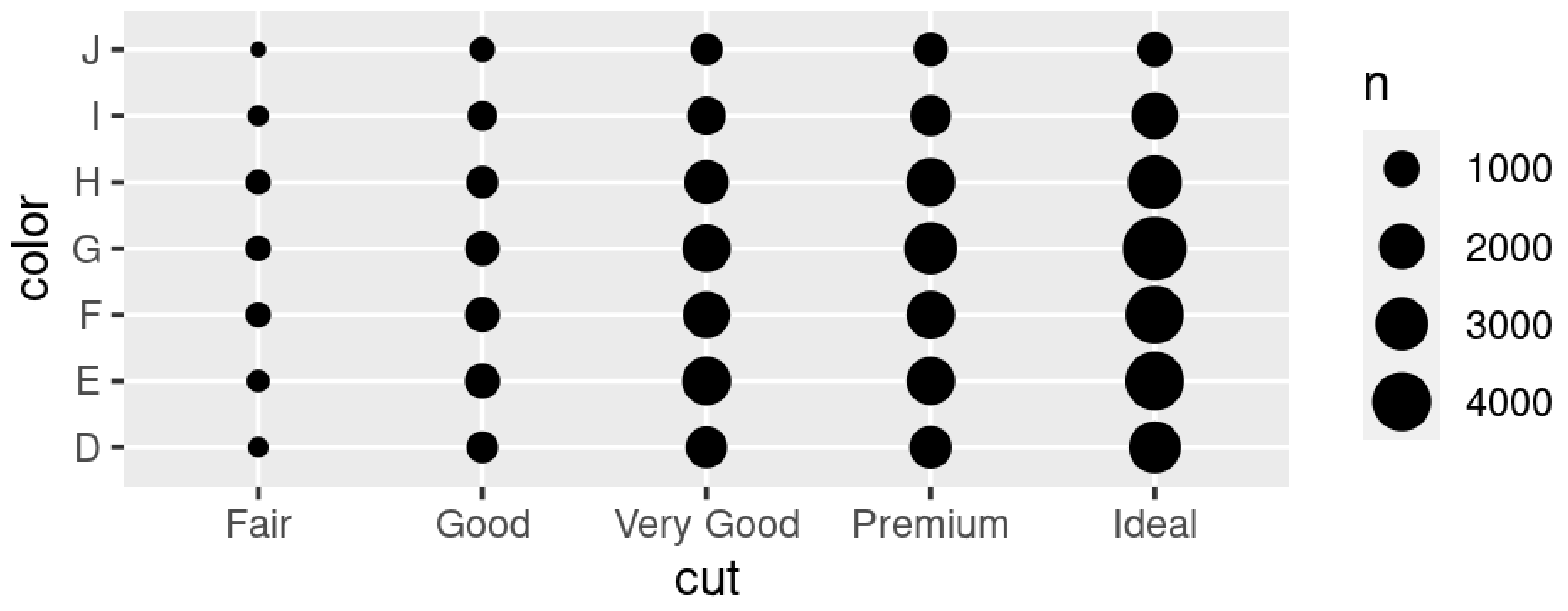
```r
#  coord_flip()
```



Two Categorical Variables

- To visualise the covariation between categorical variables, you'll need to count the number of observations for each combination. One way to do that is to rely on the built-in `geom_count()`

```r
ggplot(data = diamonds) +
  geom_count(mapping = aes(x = cut, y = color))
```



Example

- Another approach is to compute the count with dplyr.

```r
diamonds %>%
  count(color, cut)
```

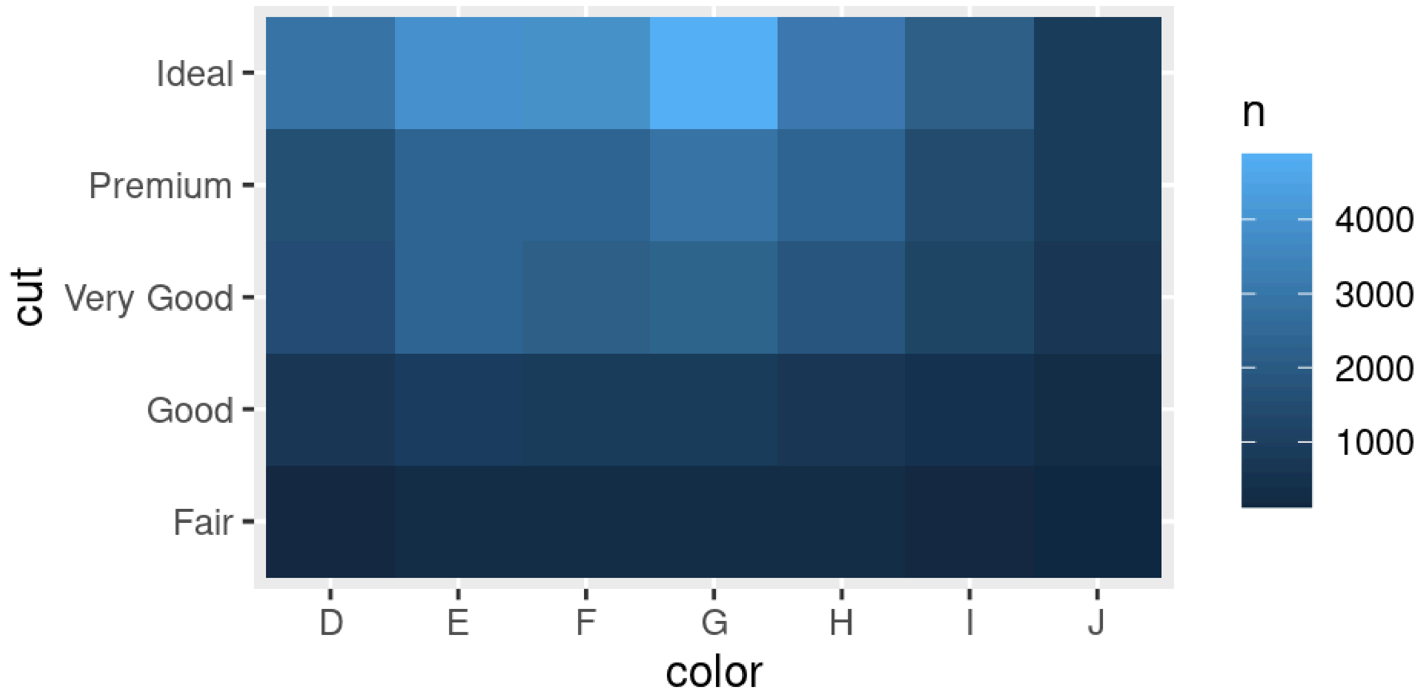
color <ord>	cut <ord>	n <int>
D	Fair	163
D	Good	662
D	Very Good	1513
D	Premium	1603
D	Ideal	2834
E	Fair	224
E	Good	933
E	Very Good	2400
E	Premium	2337
E	Ideal	3903

1-10 of 35 rows   Previous **1** 2 3 4 Next

Example

- Then visualise with geom_tile() and the fill aesthetic.

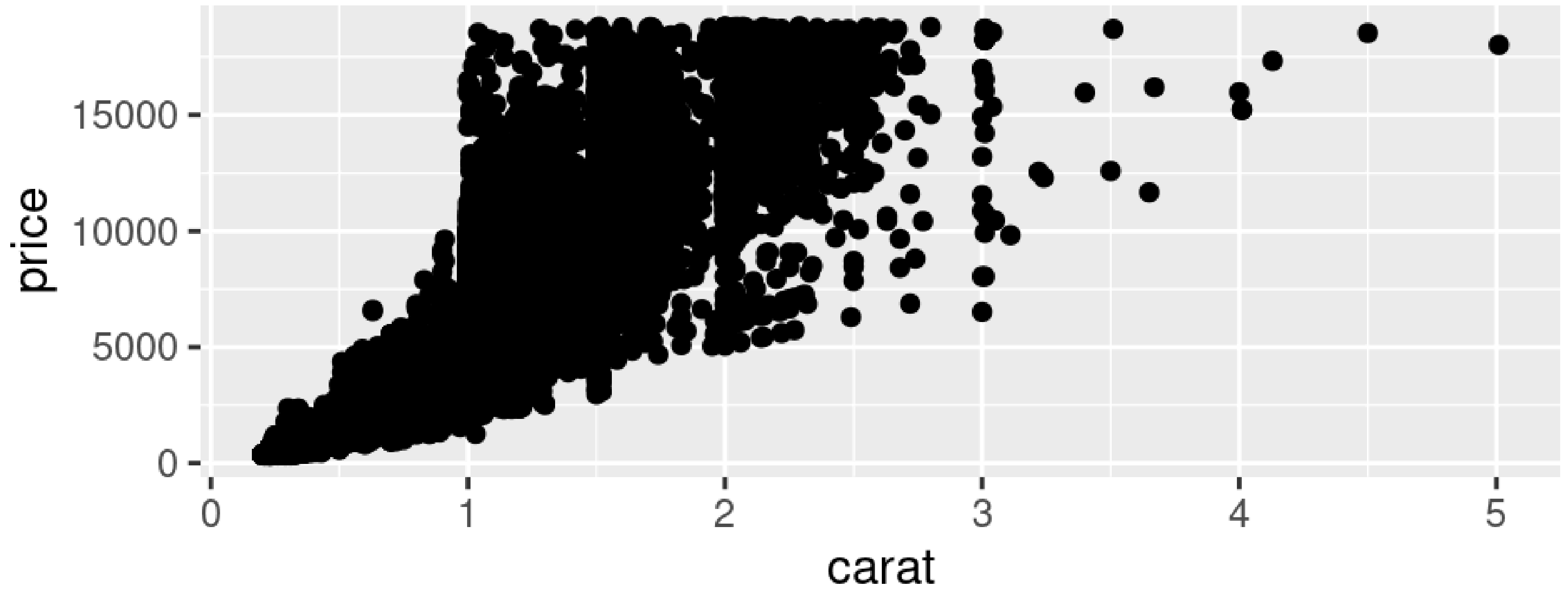
```r
diamonds %>%
  count(color, cut) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
    geom_tile(mapping = aes(fill = n))
```



Two Continuous Variables

- One great way to visualise the covariation between two continuous variables is to draw a **scatter plot** with `geom_point()`. You can see covariation as a pattern in the points.
- Example: visualise the relationship between the carat size and price of a diamond.

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

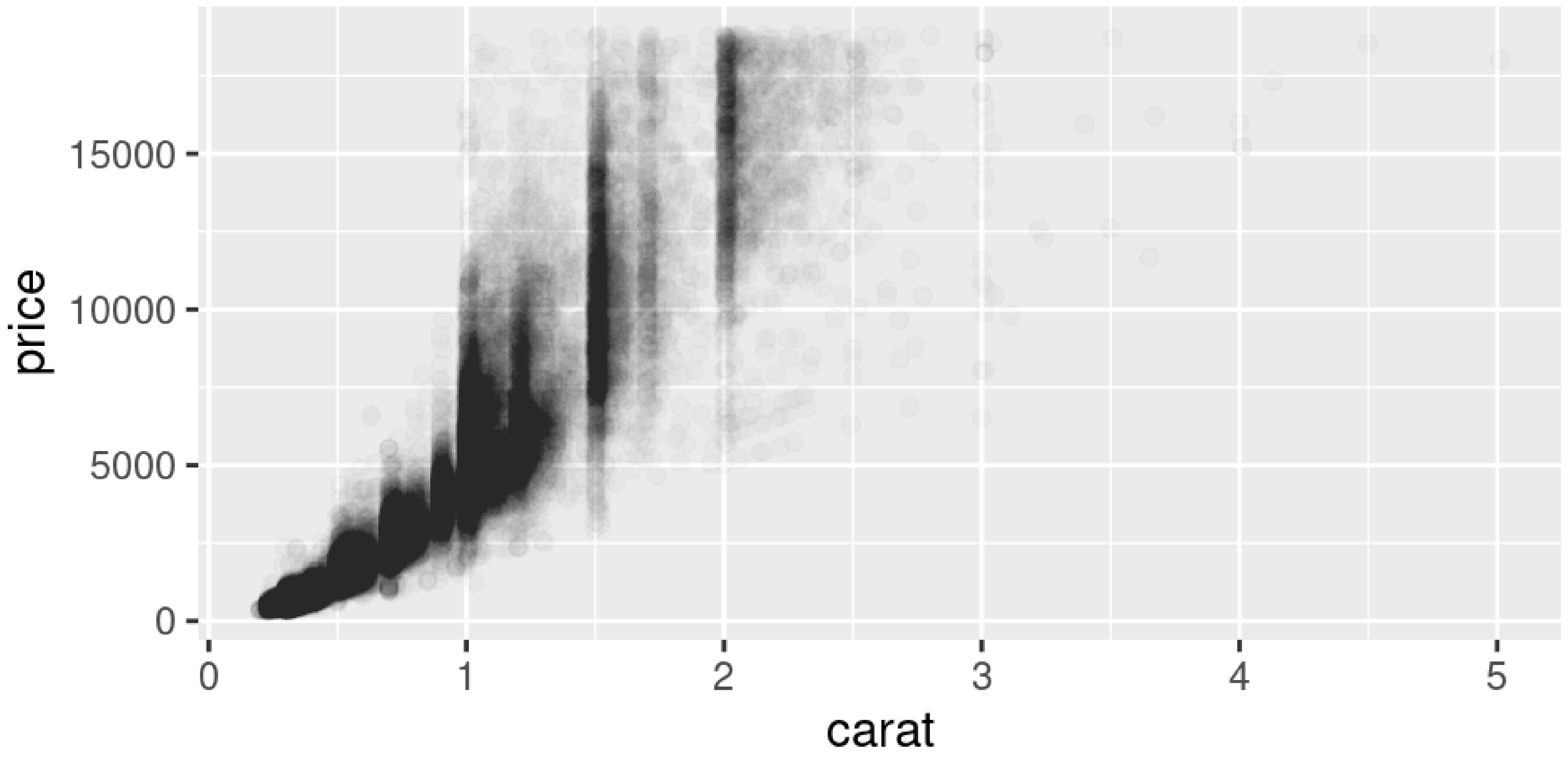


Other ways to visualisize the relationship

- use the `alpha` aesthetic to add transparency
- use `geom_bin2d()` and `geom_hex()` to bin in two dimensions
- bin one continuous variable so it acts like a categorical variable

Example: add transparency

```r
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = carat, y = price), alpha = 1 / 100)
```



From Data Patterns to Models

- Patterns in your data provide clues about relationships
- Models are a tool for extracting patterns out of data