Lecture 5 - Tibbles, Data Manipulation and Visualization

Sajid Bashir, PhD

22 February, 2020

Contents

Agenda

- Tip of the Day
- Steps in Data Science
- Extracting Functions
- Tibbles
- Introduction to R graphics (ggplot2)

```
###  Packages
library(tidyverse)
library(knitr)
```

Tip of the Day!

The R Project

Comprehensive R Archived Network

It is a content delivery network (CDN) of 97 `ftp` and `web` servers in 47 different countries. `CRAN` acts as a repository of identical, up-to-date, versions of code and documentation for `R`.

The 0-Cloud is the 1st or base server in this CDN that automatically redirects the packages to all the servers worldwide called `mirrors` and is currently sponsored by `Rstudio`.

Submitting Package

Two ways you can submit your package to CRAN:

1. Submission should meet the CRAN Repository Policy and then follow the three steps using the web form.
2. Upload to CRAN ftp server and send an email to CRAN-submissions@R-project.org.

Steps in Data Science

1. Data Requirements

Data is the essential input for *analysis*. The demand for analysis is the outcome of a *requirement* which is generated when a *question* is asked. Generally, this happens without considering whether the data to respond an answer exists or it does not. Hence, the requirement of data can be intentional or unintentional but for an analyst this is actually the starting point.

2. Data Collection

Data can be collected from a variety of sources. The requirements may be communicated by analysts to custodians of the data if it exists. Otherwise, the data can be collected from a source or a sensor in the environment, such as traffic cameras, satellites, recording devices, etc. It may also be obtained through interviews, downloads from online sources, or reading documentation. It can also be artificially generated through experimentation and simulation.

3. Data Processing

Data initially obtained must be processed or organised for analysis. For instance, these may involve placing data into rows and columns in a table format (i.e., structured data) for further analysis, such as within a spreadsheet or statistical software.

4. Data Cleaning or Munching

The processed data can be incomplete, contain duplicates, or contain errors. Data cleaning is the process of preventing and correcting these anamolies. Common tasks include record matching, identifying inaccuracy of data, overall quality of existing data, deduplication, and column segmentation.

5. Exploratory Data Analysis

Analysts apply a variety of techniques referred to as *exploratory data analysis* to begin understanding the messages contained in the data. Descriptive statistics, such as the average or median, may be generated to help understand the data. Data visualization may also be used to examine the data in graphical format, to obtain additional insight regarding the messages within the data.

6. Modeling and Algorithms

Algorithms are applied to identify relationships among the variables, such as correlation or causation. Algorithms e.g. regression analysis may be used to models and evaluate a particular variable in the data based on other variable(s), with some residual error depending on model accuracy. Analysts attempt to build models that are descriptive of the data to simplify analysis and communicate results.

7. Communicate and Visualize

This is arguably the most important step. While it might seem obvious and simple, the ability to conclude your results in a digestible format is much more difficult than it seems. The output of the analysis can be reported in many formats to the users of the analysis to support their requirements. The users may have feedback, which results in additional analysis. As such, much of the analytical cycle is iterative. When determining how to communicate the results, the analyst may consider data visualization techniques to help clearly and efficiently communicate the message to the audience.

Extracting Functions

Masked Functions

Some functions in `R` are `masked` because they exist in several different packages. We can have a list of duplicate function and for each of them, the list of `packages` in which it exists.

```
# Get complete list of masked functions
conflicts(detail = TRUE)

# List of packages having a function with name 'filter'
getAnywhere(filter())
```

We're going to start by operating on the `birthwt` dataset from the `MASS` library. Let's get it loaded and see what we're working with.

Remember

- Loading the `MASS` library overrides certain `tidyverse` functions.
- We don't want to do that so we'll extract the required dataset or function directly from `MASS`.

```
birthwt <- MASS::birthwt
typeof(birthwt)
```

```
## [1] "list"
```

Data Frames and Tibbles

Data Frames

These are the fundamental data structure in R. A dataframe is a rectangular collection of variables (in columns) and observations (in rows).

> A dataframe in tidyverse package - `mpg` - Observations collected by US Environment Protection Agency. - It shows data regarding fuel economy of 38 car models between 1999 and 2008.

Tibbles

Tibbles are a modernised version of R's original `data.frame`. These are nicer and more convenient to work with than that of data frames. In particular, they have nicer and more informative default print settings (customized number of rows and columns to be displayed - 10 rows by default). The `dplyr` functions we've been using are very nice because they map tibbles to other tibbles.

```
birthwt <- as_tibble(birthwt)
birthwt
```

```
## # A tibble: 189 x 10
##      low   age   lwt  race smoke   ptl    ht    ui   ftv   bwt
##    <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1     0    19   182     2     0     0     0     1     0  2523
##  2     0    33   155     3     0     0     0     0     3  2551
##  3     0    20   105     1     1     0     0     0     1  2557
##  4     0    21   108     1     1     0     0     1     2  2594
##  5     0    18   107     1     1     0     0     1     0  2600
##  6     0    21   124     3     0     0     0     0     0  2622
##  7     0    22   118     1     0     0     0     0     1  2637
##  8     0    17   103     3     0     0     0     0     1  2637
##  9     0    29   123     1     1     0     0     0     1  2663
## 10     0    26   113     1     1     0     0     0     0  2665
## # ... with 179 more rows
```

Run the code and see the output `df` in the console. Instead of using `data.frame` use `tibble` function and view the output again. You will be able to appreciate main differences between `data frame` and `tibble.`

```
df <- data.frame(
  name = seq(1:25),
  age = seq(1:25),
  height = seq(1:25),
  married = seq(1:25)
)
class(df)

my_df <- as_data_frame(df)
class(my_df)
my_df
```

> **Note**: If you want to import data directly into `tibble` format, you may use `read_delim()` and `read_csv()` instead of their base-R alternatives. Even though we started with the base alternatives, I recommend using these improved import commands going forward.

We can also create our own tibbles (data frames) by specifying the and the entries in each column. However, use has little control over customization as compared to another option i.e. `tribble` discussed next.

```
mytibble <- tibble(
x = 1:4,
y = x^2,
z = y + 0.75
)
```

```
mytibble
```

```
## # A tibble: 4 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1  1.75
## 2     2     4  4.75
## 3     3     9  9.75
## 4     4    16 16.8
```

The Tribble

Tribbles referred to as *Transposed Tibble* are customized tables. It helps to lay out small amounts of data in easy-to-read form where column headings are defined by formulas (start with ~) and the data entries are separated by commas.

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

```
## # A tibble: 2 x 3
##   x         y     z
##   <chr> <dbl> <dbl>
## 1 a         2   3.6
## 2 b         1   8.5
```

Renaming

Variables

Generally, the dataset doesn't come with very descriptive variable names. Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```
# The default names are not very descriptive
colnames(MASS::birthwt)
```

```
##  [1] "low"   "age"   "lwt"   "race"  "smoke" "ptl"   "ht"    "ui"    "ftv"
## [10] "bwt"
```

```
# Better names!
colnames(birthwt) <- c("birthwt.below.2500", "mother.age",
                       "mother.weight", "race", "mother.smokes",
                       "previous.prem.labor", "hypertension",
                       "uterine.irr", "physician.visits", "birthwt.grams")
colnames(birthwt)
```

```
##  [1] "birthwt.below.2500"  "mother.age"          "mother.weight"
##  [4] "race"                "mother.smokes"       "previous.prem.labor"
##  [7] "hypertension"        "uterine.irr"         "physician.visits"
## [10] "birthwt.grams"
```

Alternate Approach

`rename` operates by allowing you to specify a new variable name for whichever old variable name you want to change.

```
rename(new variable name = old variable name)
```

```
# Reload the data again
birthwt <- as_tibble(MASS::birthwt)

birthwt <- birthwt %>%
  rename(birthwt.below.2500 = low,
         mother.age = age,
         mother.weight = lwt,
         mother.smokes = smoke,
         previous.prem.labor = ptl,
         hypertension = ht,
         uterine.irr = ui,
         physician.visits = ftv,
         birthwt.grams = bwt)

colnames(birthwt)
```

```
##  [1] "birthwt.below.2500"  "mother.age"          "mother.weight"
##  [4] "race"                "mother.smokes"       "previous.prem.labor"
##  [7] "hypertension"        "uterine.irr"         "physician.visits"
## [10] "birthwt.grams"
```

Note that in this command we didn't rename the race variable because it already had a good name.

Factors

All the factors are currently represented as integers. Let's use the `mutate()`, `mutate_at()` and `?recode_factor()` functions

> `mutate()` → adds new variables and preserves existing ones `transmutate()` → adds new variables and drops all existing ones `mutate_at()` → apply same transformation to multiple variables `recode_factor()` → replace `numeric` values based on position or name and `characters` or `factors` based on names `%>%` pipe or chain increases code readability - represents a sequence of operations e.g.

```r
years <- factor(2008:2012)
# nesting
as.numeric(as.character(years))
```

```
## [1] 2008 2009 2010 2011 2012
```

```r
# piping
years %>% as.character %>% as.numeric
```

```
## [1] 2008 2009 2010 2011 2012
```

```

Converting variables to factors and giving the factors more meaningful levels

```
birthwt <- birthwt %>%
  mutate(race = recode_factor(race, `1` = "white", `2` = "black", `3` = "other")) %>%
  mutate_at(c("mother.smokes", "hypertension", "uterine.irr", "birthwt.below.2500"),
            ~ recode_factor(.x, `0` = "no", `1` = "yes"))

birthwt
```

```
## # A tibble: 189 x 10
##    birthwt.below.2~ mother.age mother.weight race  mother.smokes
##    <fct>                <int>         <int> <fct> <fct>
##  1 no                      19           182 black no
##  2 no                      33           155 other no
##  3 no                      20           105 white yes
##  4 no                      21           108 white yes
##  5 no                      18           107 white yes
##  6 no                      21           124 other no
##  7 no                      22           118 white no
##  8 no                      17           103 other no
##  9 no                      29           123 white yes
## 10 no                      26           113 white yes
## # ... with 179 more rows, and 5 more variables: previous.prem.labor <int>,
## #   hypertension <fct>, uterine.irr <fct>, physician.visits <int>,
## #   birthwt.grams <int>
```

Recall that the syntax `~ recode_factor(.x, ...)` defines an anonymous function that will be applied to every column specfied in the first part of the `mutate_at()` call. In this case, all of the specified variables are binary 0/1 coded, and are being recoded to no/yes.

Now that things are coded correctly, we can look at an overall summary:

```
summary(birthwt)
```

```
##  birthwt.below.2500   mother.age    mother.weight      race    mother.smokes
##  no :130            Min.   :14.00   Min.   : 80.0   white:96   no :115
##  yes: 59            1st Qu.:19.00   1st Qu.:110.0   black:26   yes: 74
##                     Median :23.00   Median :121.0   other:67
##                     Mean   :23.24   Mean   :129.8
##                     3rd Qu.:26.00   3rd Qu.:140.0
##                     Max.   :45.00   Max.   :250.0
##  previous.prem.labor hypertension uterine.irr physician.visits birthwt.grams
##  Min.   :0.0000      no :177      no :161     Min.   :0.0000   Min.   : 709
##  1st Qu.:0.0000      yes: 12      yes: 28     1st Qu.:0.0000   1st Qu.:2414
##  Median :0.0000                               Median :0.0000   Median :2977
##  Mean   :0.1958                               Mean   :0.7937   Mean   :2945
##  3rd Qu.:0.0000                               3rd Qu.:1.0000   3rd Qu.:3487
##  Max.   :3.0000                               Max.   :6.0000   Max.   :4990
```
```

Revisit Statistics

Standard deviation: A non-negative number to measure of dispersement i.e. spread of data around the `mean` or `average` e.g. how close to the average is your score?

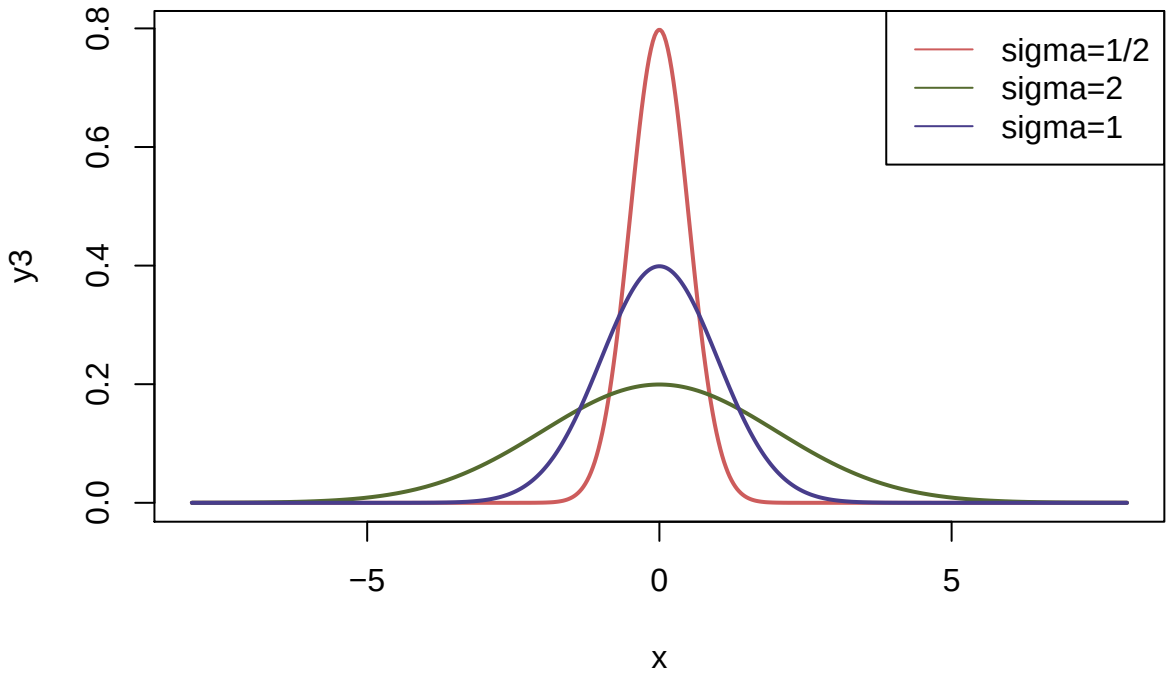
$$S_x = \sqrt{\frac{\sum (x - \overline{x})^2}{n - 1}}$$

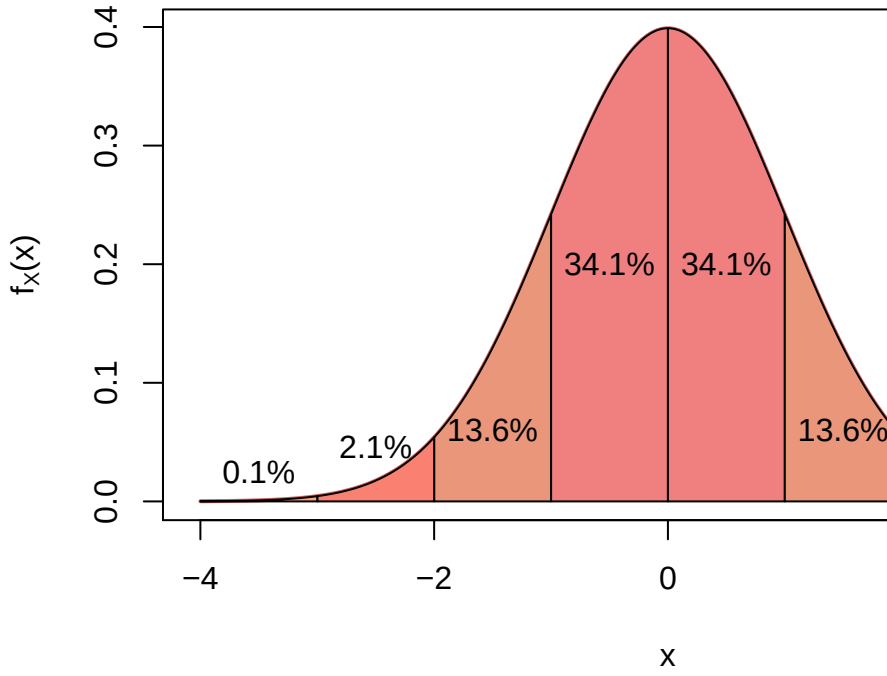
where $\overline{x} = \frac{1}{n} \sum x$ is the sample mean and n is the size of the sample.

Normal Distribution: Bell curve also called "normal distribution" models most of the naturally occuring process. The expression for $\mathcal{N}(\mu, \sigma)$ is:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where μ and σ represent the mean and standard deviation of the distribution. Note: σ^2 is known as *variance*.





It is also a tool to understand standard deviation.

Self Study Topics

- Mean, Median, Mode (measure of central tendency)
- Standard Deviation and Variance (measure of dispersion)
- Standard Normal Distribution
- Random Variables
- Expectation

Step into `Analysis`

A simple table

Question 01: What is the average birth weight of the newborns with regards to smoker and non-smoker mothers of different races?

- Consider the following functions: > - `group_by()`: We can group different variables on which same data operations are to be performed. `ungroup()` removes grouping.
  - `summarize()`: Gives a summary of the variables in a dataframe based on operation specified as argument. It is very effective when used with `group_by()` as demonstrated above. The syntax is as under:

```
summarise(df, variable_name=condition)
arguments:
  - `df`: Dataset used to construct the summary statistics
  - `variable_name=condition`: Formula to create the new variable
```

- Let's use the `summarize()` and `group_by()` functions to see what the average birthweight looks like when broken down by race and smoking status. We'll round to the nearest gram to make the printout nicer.

```
tbl.mean.bwt <- birthwt %>%
  group_by(race, mother.smokes) %>%
  summarize(mean.birthwt = round(mean(birthwt.grams), 0))
tbl.mean.bwt
```

```
## # A tibble: 6 x 3
## # Groups:   race [3]
##   race  mother.smokes mean.birthwt
##   <fct> <fct>                <dbl>
## 1 white no                    3429
## 2 white yes                   2827
## 3 black no                    2854
## 4 black yes                   2504
## 5 other no                    2816
## 6 other yes                   2757
```

A Simple Reshape

Question 02: Does smoking status appear to have an effect on birth weight?

We can approach the answer if the data is in a wider rather than a long format. Here's how we can do that with the `spread()` function from `tidyr` > Basic `spread()` call is `spread(data, key, value)`

```
tbl.mean.bwt %>% spread(mother.smokes, mean.birthwt)
```

```
## # A tibble: 3 x 3
## # Groups:   race [3]
##   race     no   yes
##   <fct> <dbl> <dbl>
## 1 white  3429  2827
## 2 black  2854  2504
## 3 other  2816  2757
```

Giving the output a nicer looking look? - Let's use the header `{r, results='asis'}`, along with the `kable()` function from the `knitr` library

```
# Save the table from before as a
# Print nicely
kable(spread(tbl.mean.bwt, mother.smokes, mean.birthwt),
      format = "markdown")
```

race	no	yes
white	3429	2827
black	2854	2504
other	2816	2757

- `kable()` outputs the table in a way that Markdown can read and nicely display
- Note: changing the CSS changes the table appearance

Correlation

Correlation coefficient is used to find relationship between data components. Different formulas can be used to return a value between -1 and 1, where:

- 1 → Strong Positive Relationship.
- -1 → Strong Negative Relationship.
- 0 → No relationship (Orthogonal).

Pearson's Correlation Coefficient

$$\rho = \frac{n\sum xy - \sum x \sum y}{\sqrt{\left((n\sum x^2) - (\sum x)^2\right)\left((n\sum y^2) - (\sum y)^2\right)}}$$

where x and y are two variables from the data set.

Sample Correlation Coefficient

$$r_{xy} = \frac{S_{xy}}{S_x S_y}$$

S_x and S_y are the sample standard deviations, and S_{xy} is the *sample covariance*.

Population Correlation Coefficient

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

where σ_{xy} is the *population covariance*.

Covariance Covariance is a measure of how much two random variables vary together. It's similar to variance, but where variance tells you how a single variable varies, covariance tells you how two variables vary together.

$$\begin{aligned} \operatorname{Cov}(X,Y) &= \operatorname{E}((X-\mu_X)(Y-\mu_Y)) \\ &= \tfrac{1}{n-1}\sum (X-\mu_X)(Y-\mu_Y) \end{aligned}$$

Question 3: What is the association between mother's age and birth weight?

```
cor(birthwt$mother.age, birthwt$birthwt.grams)  # Calculate correlation
```

```
## [1] 0.09031781
```

Question that you should try to answer:

Does the correlation vary with smoking status?

Data Visualization

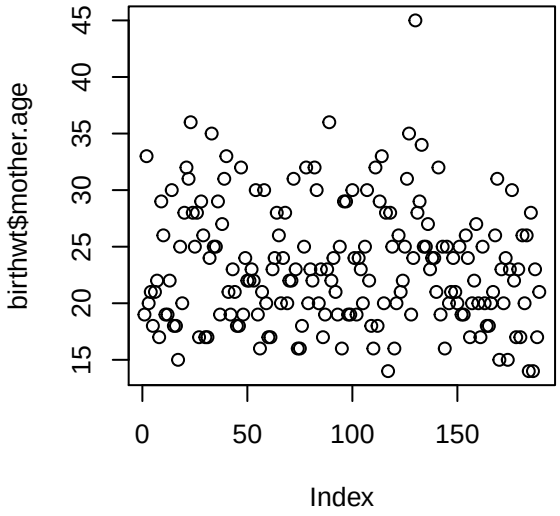
We now know a lot about how to tabulate data. It's often easier to look at plots instead of tables. We'll now talk about some of the standard plotting options.

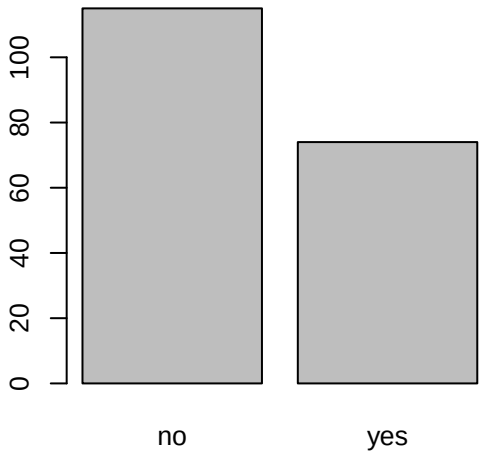
Standard graphics

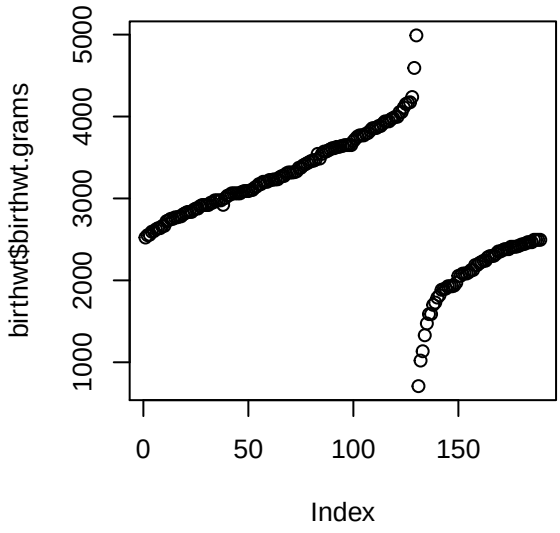
Single-variable plots

Let's continue with the `birthwt` data from the `MASS` library. Here are some basic single-variable plots.

```
par(mfrow = c(2,2)) # Display plots in a single 2 x 2 figure
plot(birthwt$mother.age)
with(birthwt, hist(mother.age))
plot(birthwt$mother.smokes)
plot(birthwt$birthwt.grams)
```



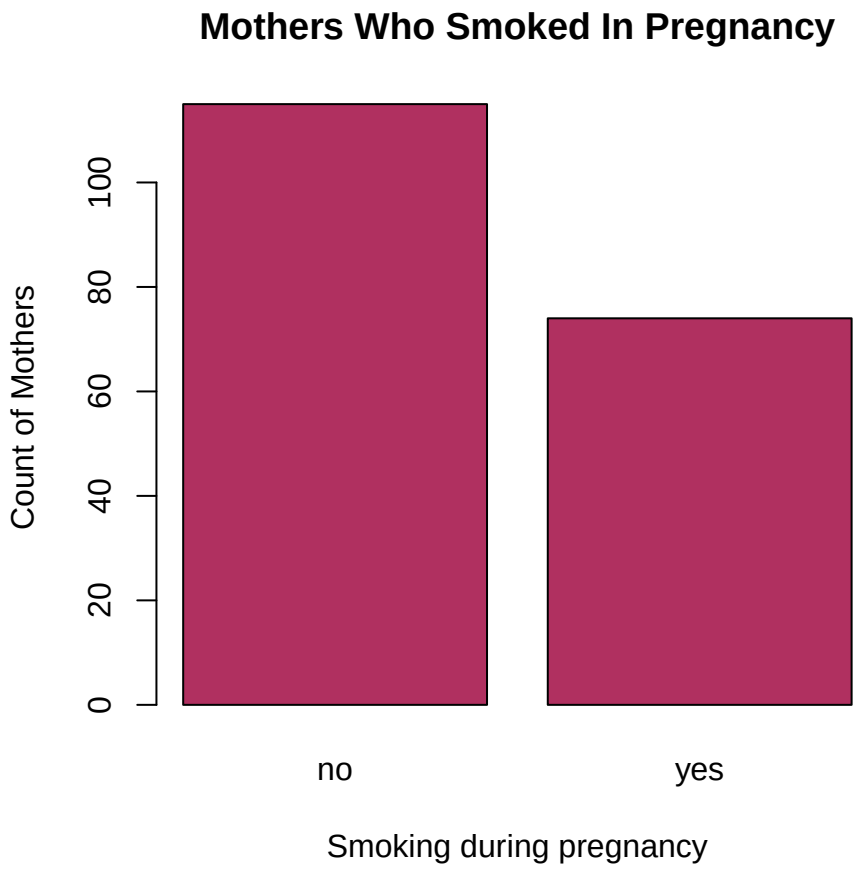




Note that the result of calling `plot(x, ...)` varies depending on what `x` is. - When `x` is *numeric*, you get a plot showing the value of `x` at every `index`.
- When `x` is a *factor*, you get a bar plot of counts for every `level`

Let's add more information to the smoking bar plot, and also change the color by setting the `col` option.

```
par(mfrow = c(1,1))
plot(birthwt$mother.smokes,
     main = "Mothers Who Smoked In Pregnancy",
     xlab = "Smoking during pregnancy",
     ylab = "Count of Mothers",
     col = "maroon")
```



Better Graphics

Introduction to `ggplot2`

`ggplot2` has a slightly steeper learning curve than the base graphics functions, but it also generally produces far better and more easily customizable graphics.

There are two basic calls in ggplot:

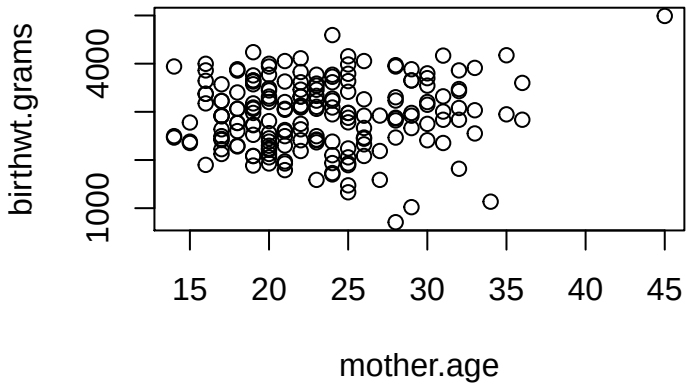
- `qplot(x, y, ..., data)`: a "quick-plot" routine, which essentially replaces the base `plot()`
- `ggplot(data, aes(x, y, ...), ...)`: defines a graphics object from which plots can be generated, along with *aesthetic mappings* that specify how variables are mapped to visual properties.

```
library(ggplot2)
```

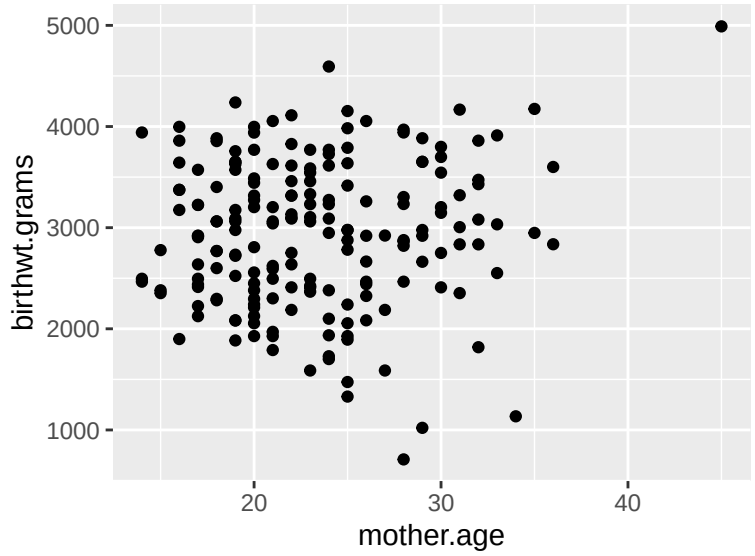
plot vs qplot

Here's how the default scatterplots look in `ggplot` compared to the base graphics. We'll illustrate things by continuing to use the `birthwt` data from the `MASS` library.

```
with(birthwt, plot(mother.age, birthwt.grams))  # Base graphics
```




```
qplot(mother.age, birthwt.grams, data=birthwt)  # using qplot from ggplot2
```

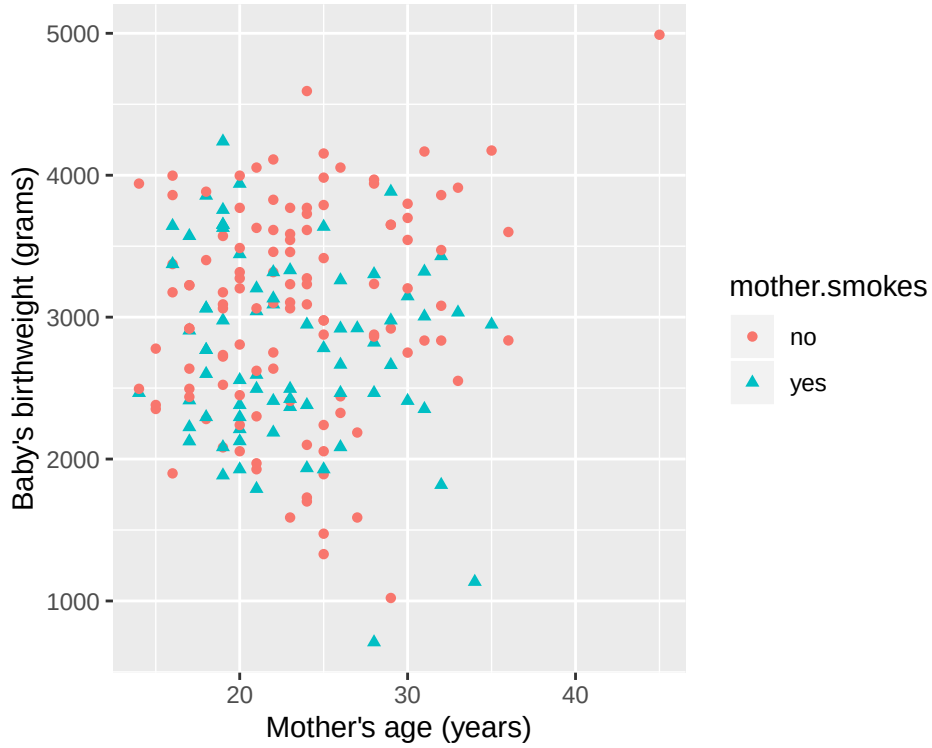


I've snuck the `with()` command into this example. `with()` allows you to use the variables in a data frame directly in evaluating the expression in the second argument.

Remember how it took us some effort last time to add color coding, use different plotting characters, and add a legend? Here's the `qplot` call that does it all in one simple line.

```
qplot(x=mother.age, y=birthwt.grams, data=birthwt,
      color = mother.smokes,
```

```
      shape = mother.smokes,
      xlab = "Mother's age (years)",
      ylab = "Baby's birthweight (grams)")
```



This way you won't run into problems of accidentally producing the wrong legend. The legend is produced based on the `colour` and `shape` argument that you pass in. (Note: `color` and `colour` have the same effect.)

ggplot function

The `ggplot2` library comes with a dataset called `diamonds`. Let's look at it

```
dim(diamonds)
```

```
## [1] 53940    10
```

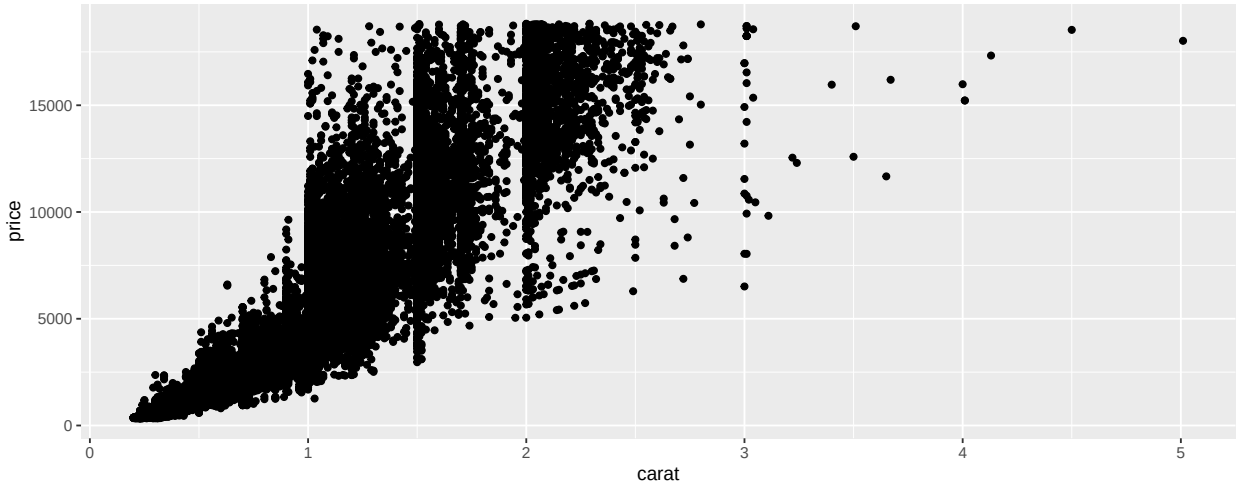
```
head(diamonds)
```

```
## # A tibble: 6 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 0.290 Premium   I     VS2      62.4    58   334  4.2   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
```

It is a data frame of 53,940 diamonds, recording their attributes such as carat, cut, color, clarity, and price.

We will make a scatterplot showing the price as a function of the carat (size). (The data set is large so the plot may take a few moments to generate.)

```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price))
diamond.plot + geom_point()
```



The data set looks a little weird because a lot of diamonds are concentrated on the 1, 1.5 and 2 carat mark.

Let's take a step back and try to understand the ggplot syntax.

1) The first thing we did was to define a graphics object, `diamond.plot`. This definition told R that we're using the `diamonds` data, and that we want to display `carat` on the x-axis, and `price` on the y-axis.

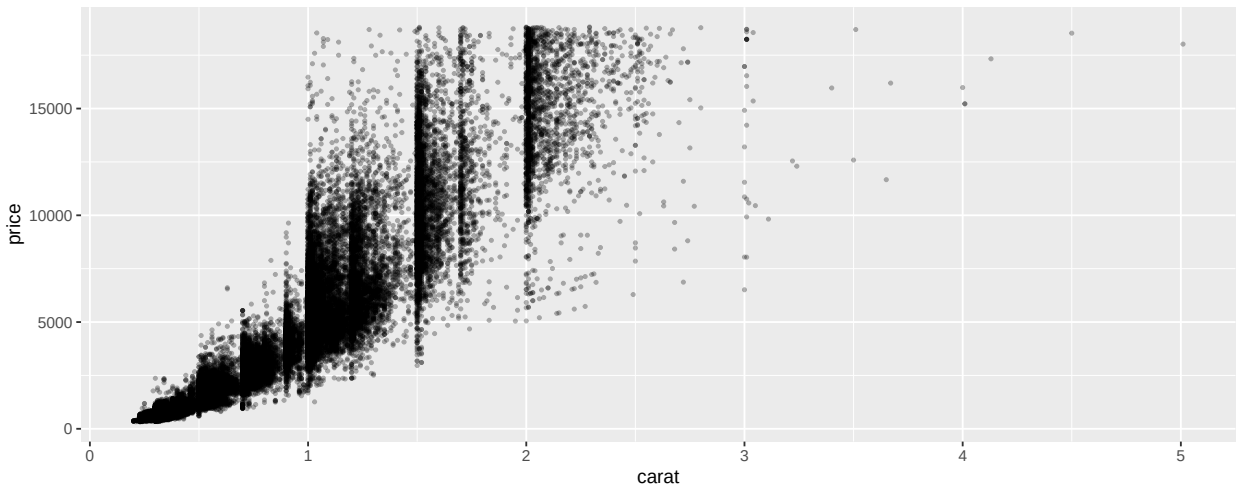
2) We then called `diamond.plot + geom_point()` to get a scatterplot.

The arguments passed to `aes()` are called **mappings**. Mappings specify what variables are used for what purpose. When you use `geom_point()` in the second line, it pulls `x`, `y`, `colour`, `size`, etc., from the **mappings** specified in the `ggplot()` command.

You can also specify some arguments to `geom_point` directly if you want to specify them for each plot separately instead of pre-specifying a default.

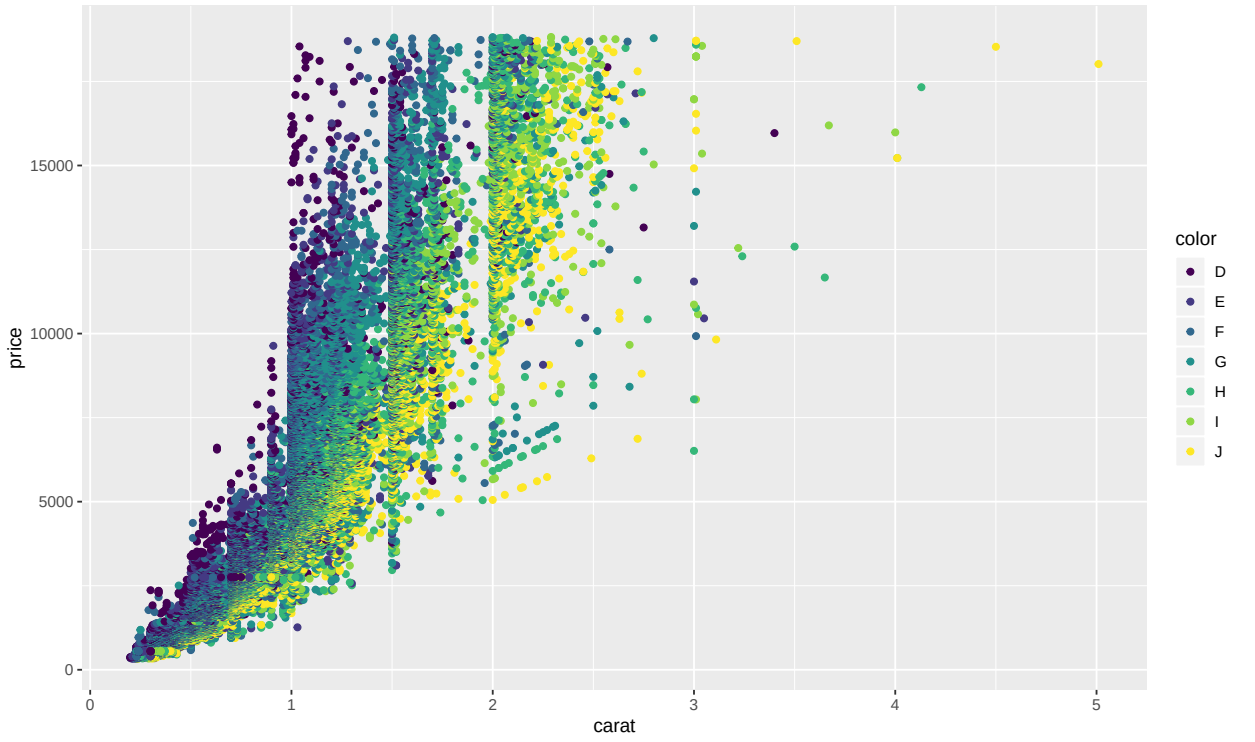
Here we shrink the points to a smaller size, and use the `alpha` argument to make the points transparent.

```
diamond.plot + geom_point(size = 0.7, alpha = 0.3)
```



If we wanted to let point color depend on the color indicator of the diamond, we could do so in the following way.

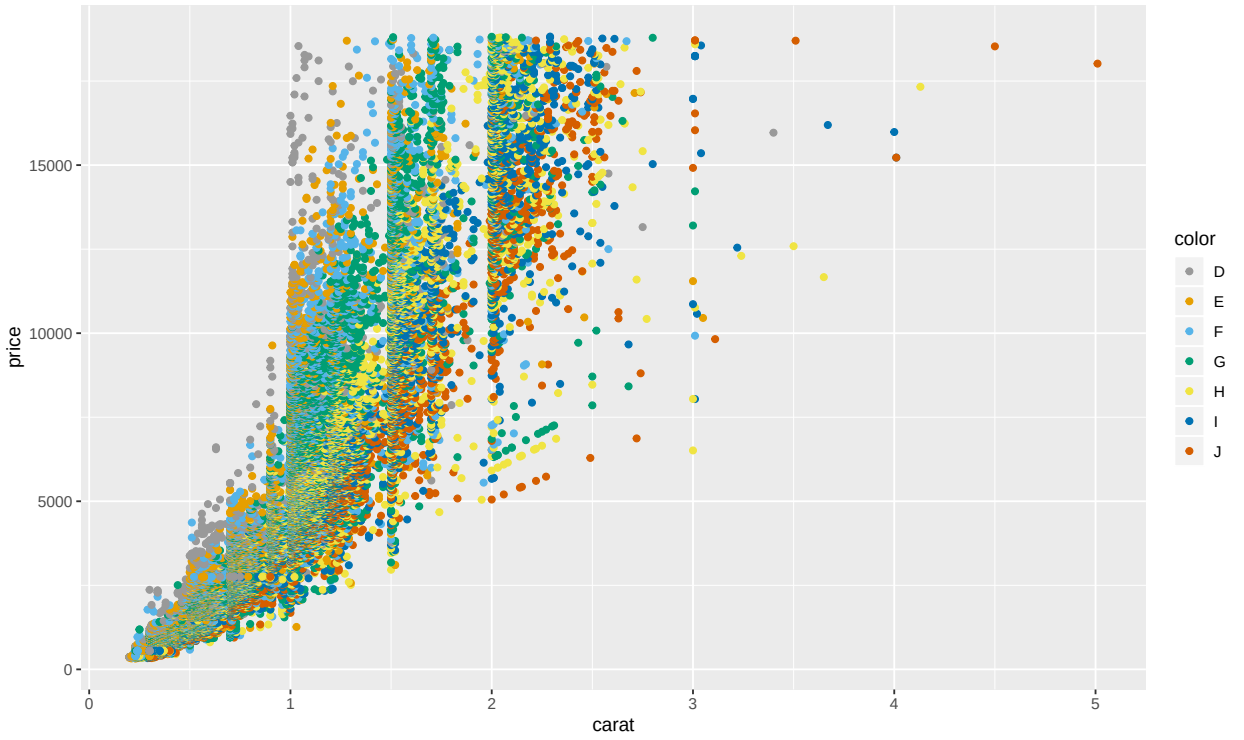
```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point()
```



If we didn't know anything about diamonds going in, this plot would indicate to us that **D** is likely the highest diamond grade, while **J** is the lowest grade.

We can change colors by specifying a different color palette. Here's how we can switch to the `cbPalette` we saw last class.

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point() + scale_colour_manual(values=cbPalette)
```



To make the scatterplot look more typical, we can switch to logarithmic coordinate axis spacing.

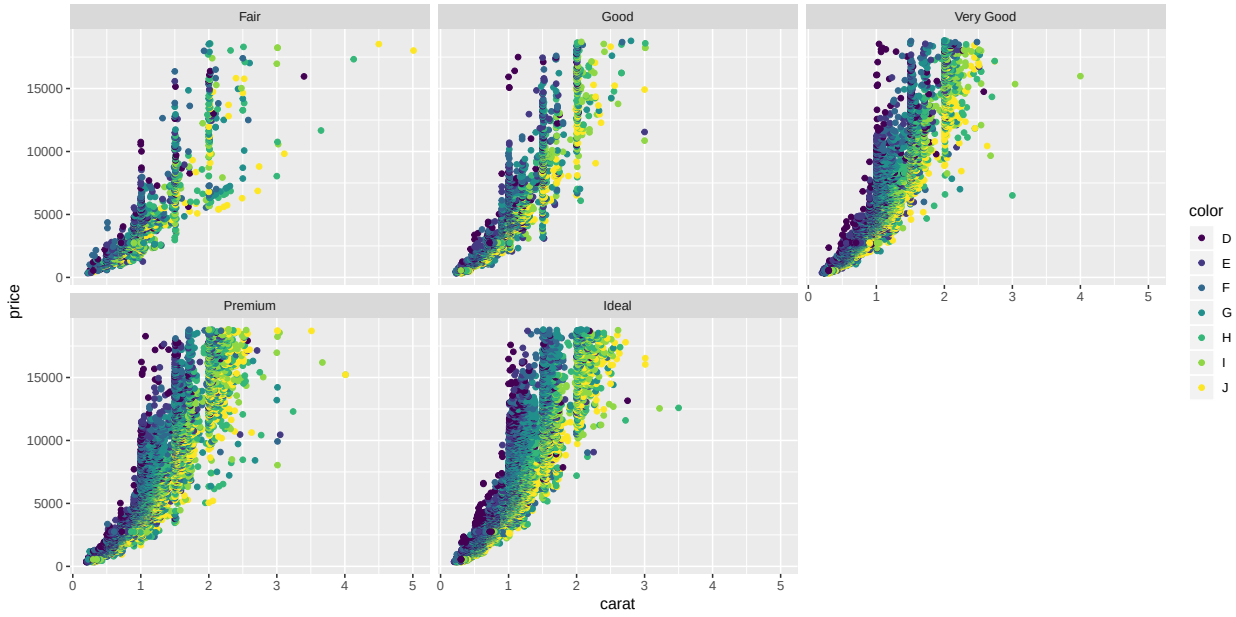
```
diamond.plot + geom_point() +
  coord_trans(x = "log10", y = "log10")
```

Conditional plots

We can create plots showing the relationship between variables across different values of a factor. For instance, here's a scatterplot showing how diamond price varies with carat size, conditioned on color. It's created using the `facet_wrap(~ factor1 + factor2 + ... + factorn)` command.

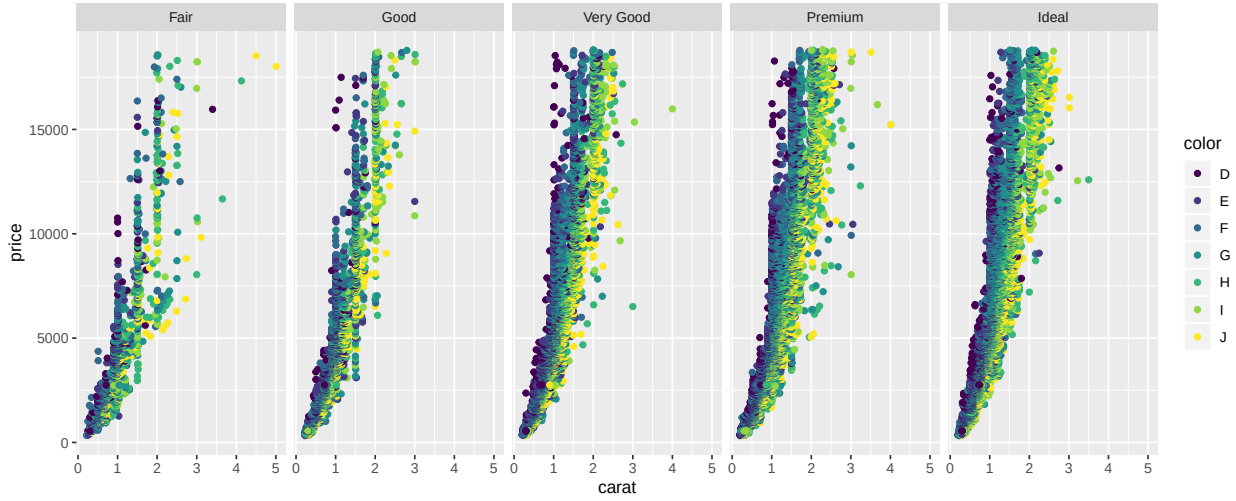
```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))

diamond.plot + geom_point() + facet_wrap(~ cut)
```

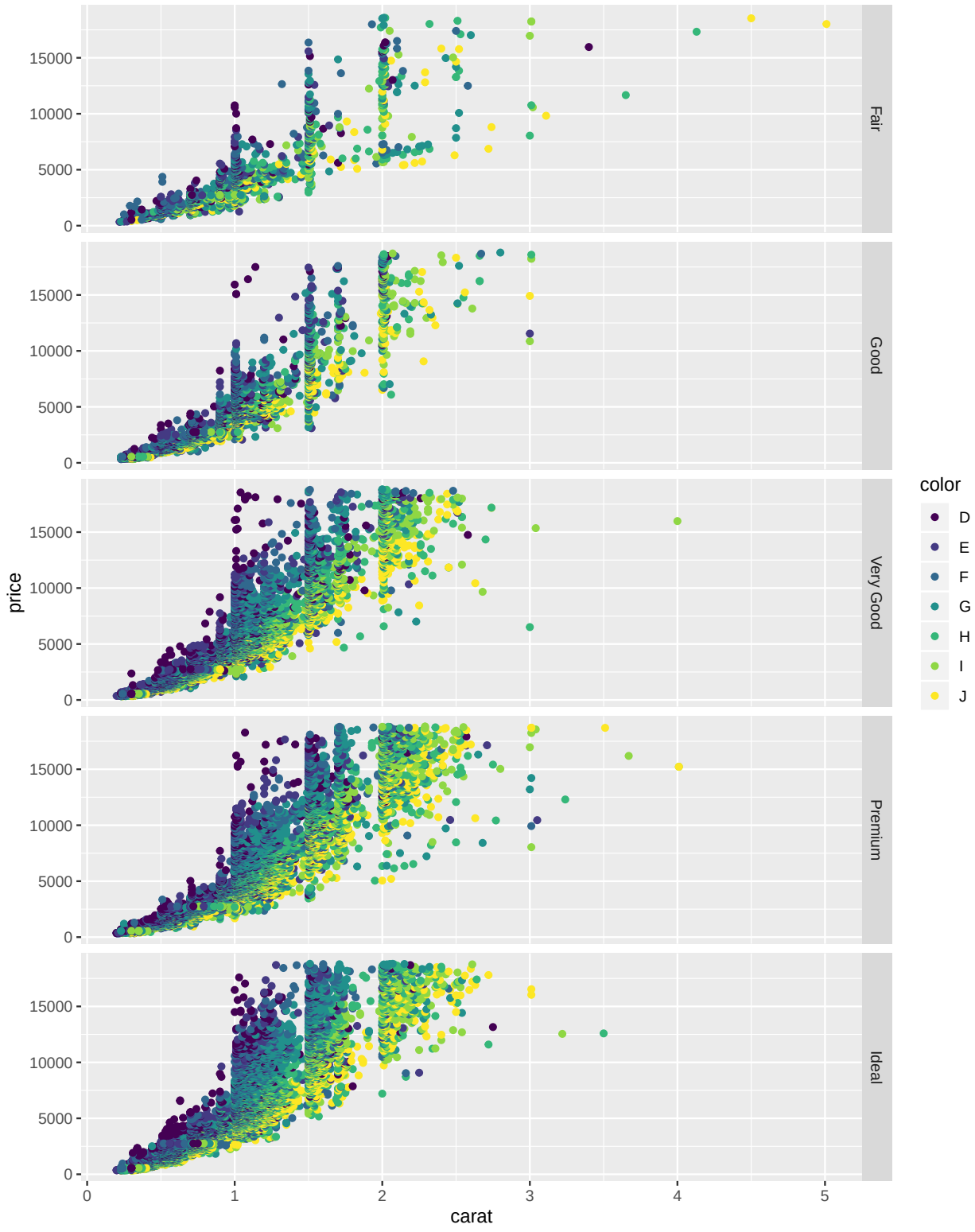


You can also use `facet_grid()` to produce this type of output.

```
diamond.plot + geom_point() + facet_grid(. ~ cut)
```




```
diamond.plot + geom_point() + facet_grid(cut ~ .)
```



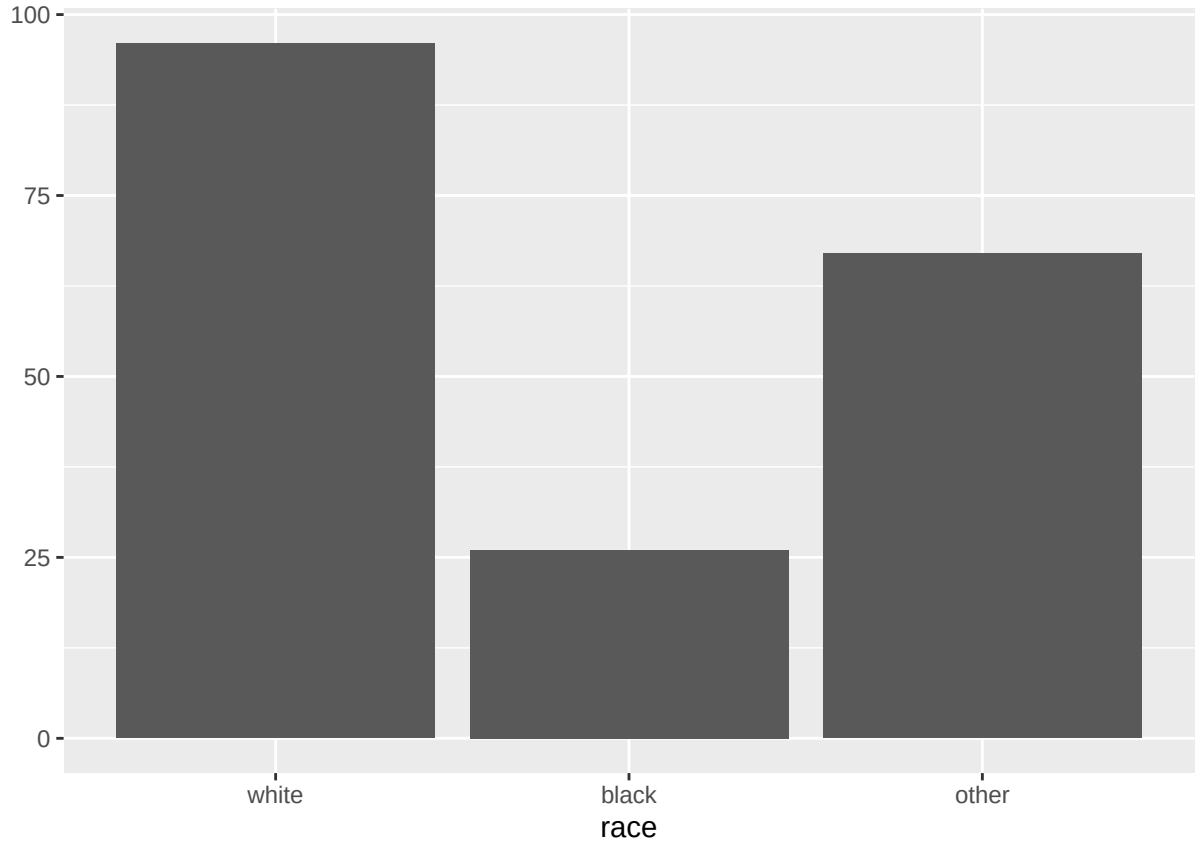
`ggplot` can create a lot of different kinds of plots, just like lattice. Here are some examples.

Function	Description
`geom_point(...)`	Points, i.e., scatterplot

Function	Description
`geom_bar(...)`	Bar chart
`geom_line(...)`	Line chart
`geom_boxplot(...)`	Boxplot
`geom_violin(...)`	Violin plot
`geom_density(...)`	Density plot with one variable
`geom_density2d(...)`	Density plot with two variables
`geom_histogram(...)`	Histogram

A bar chart

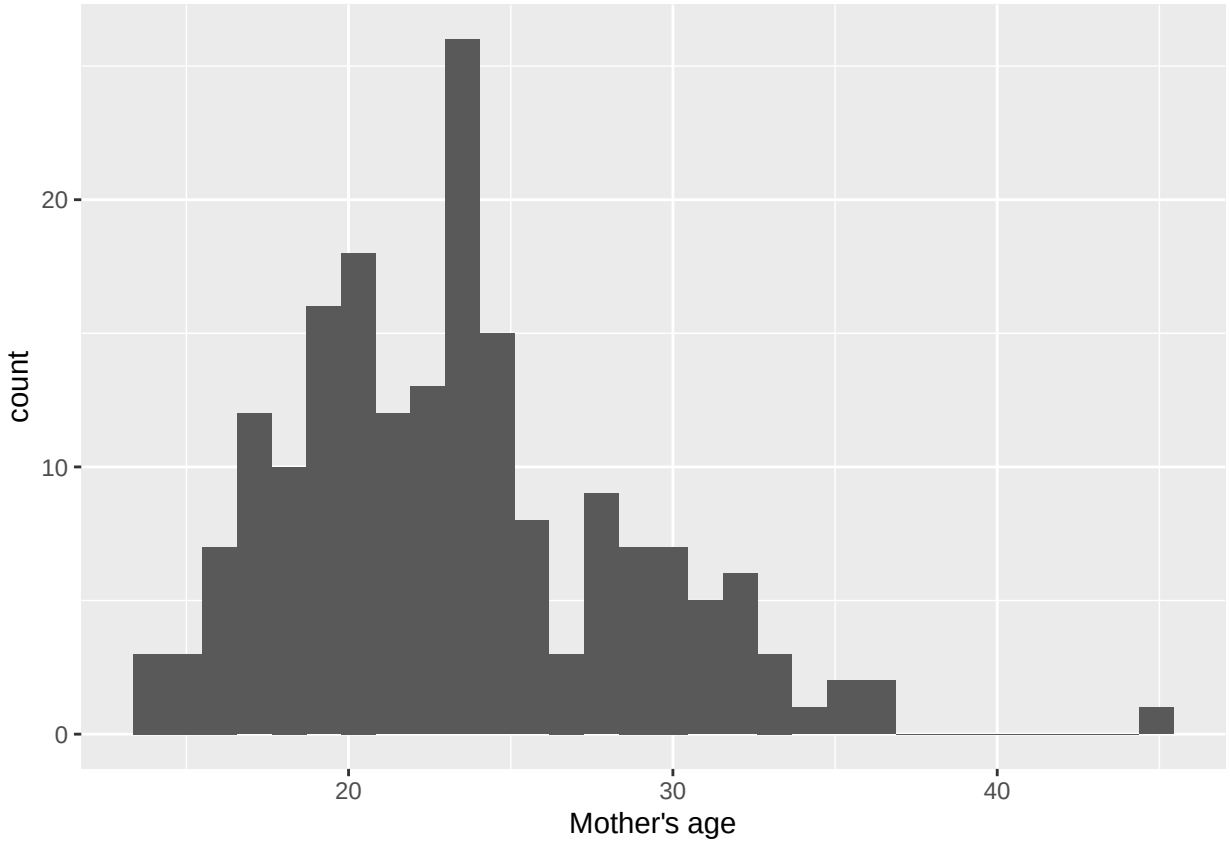
```
qplot(x = race, data = birthwt, geom = "bar")
```



Histograms and density plots

```
base.plot <- ggplot(birthwt, aes(x = mother.age)) +
  xlab("Mother's age")
base.plot + geom_histogram()
```

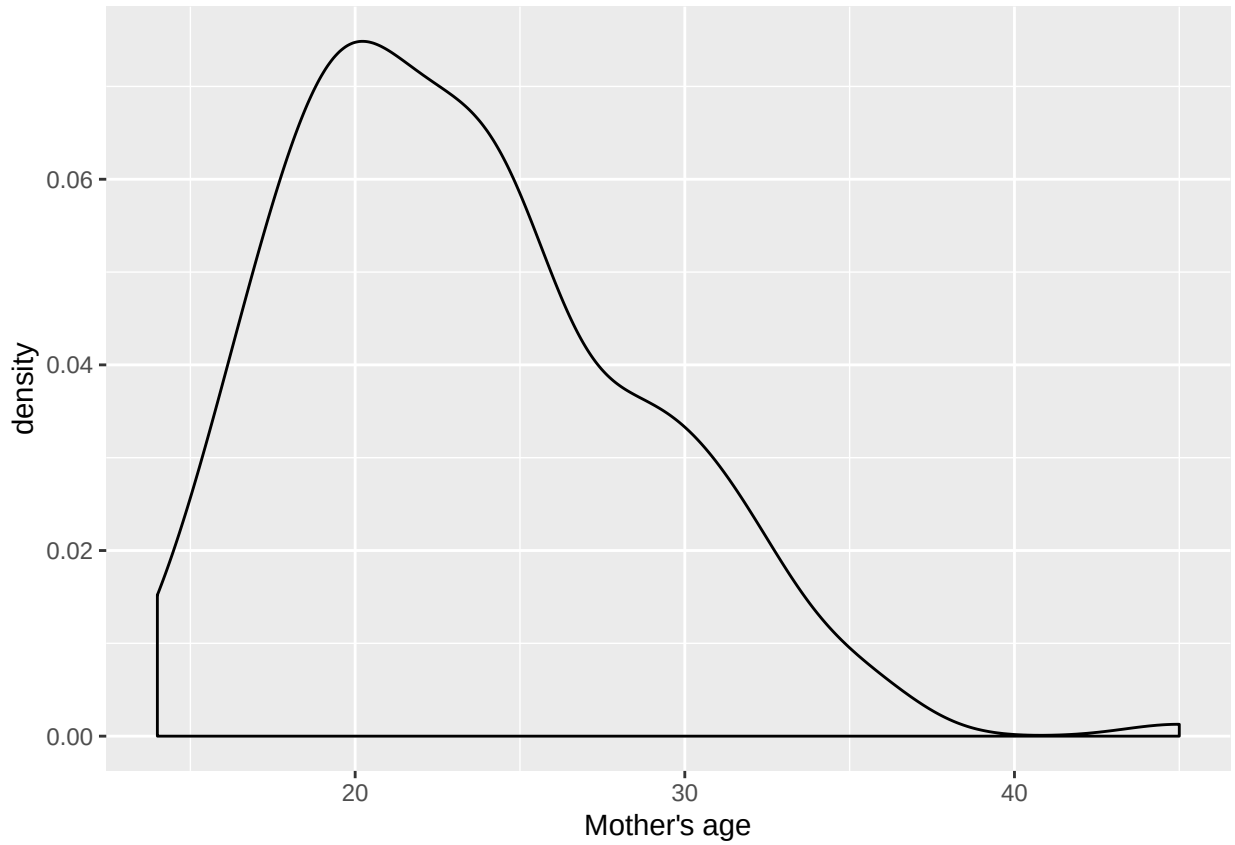
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



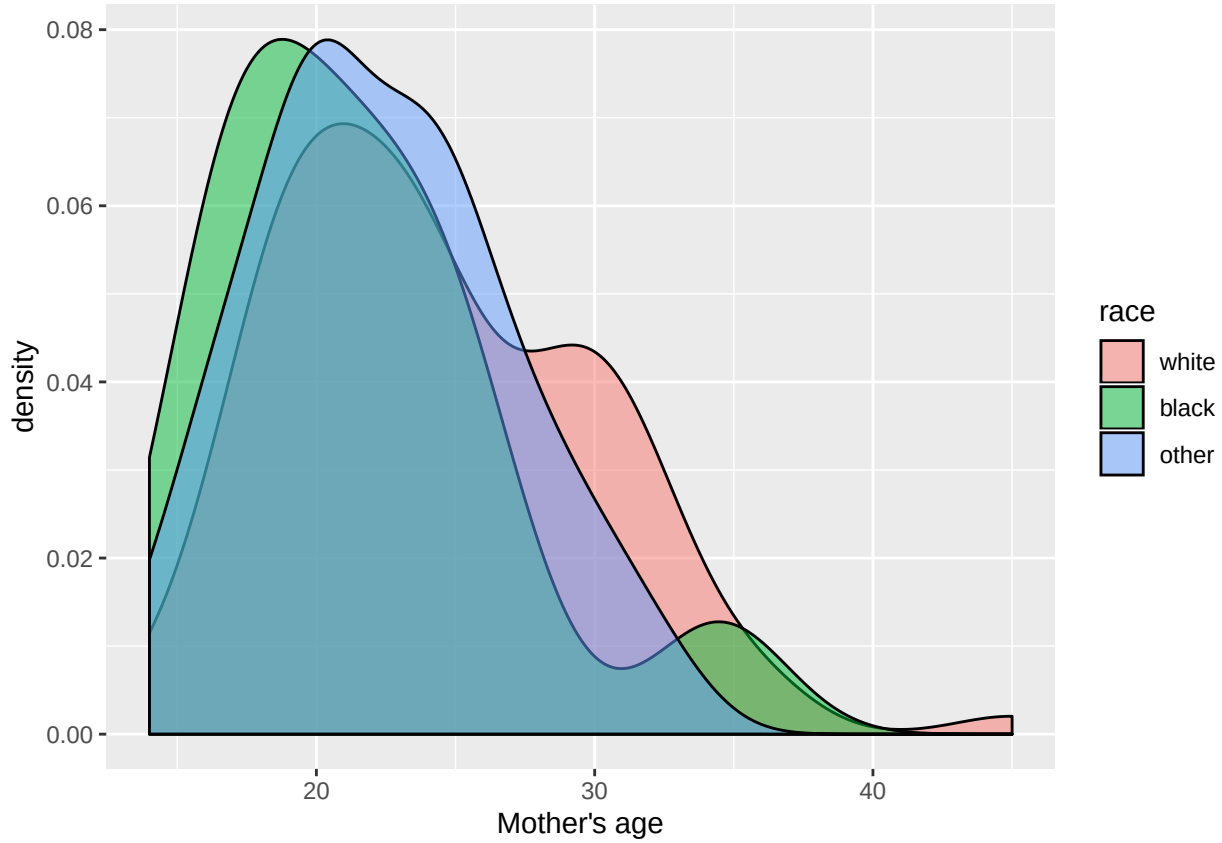

```
base.plot + geom_histogram(aes(fill = race))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


```
base.plot + geom_density()
```



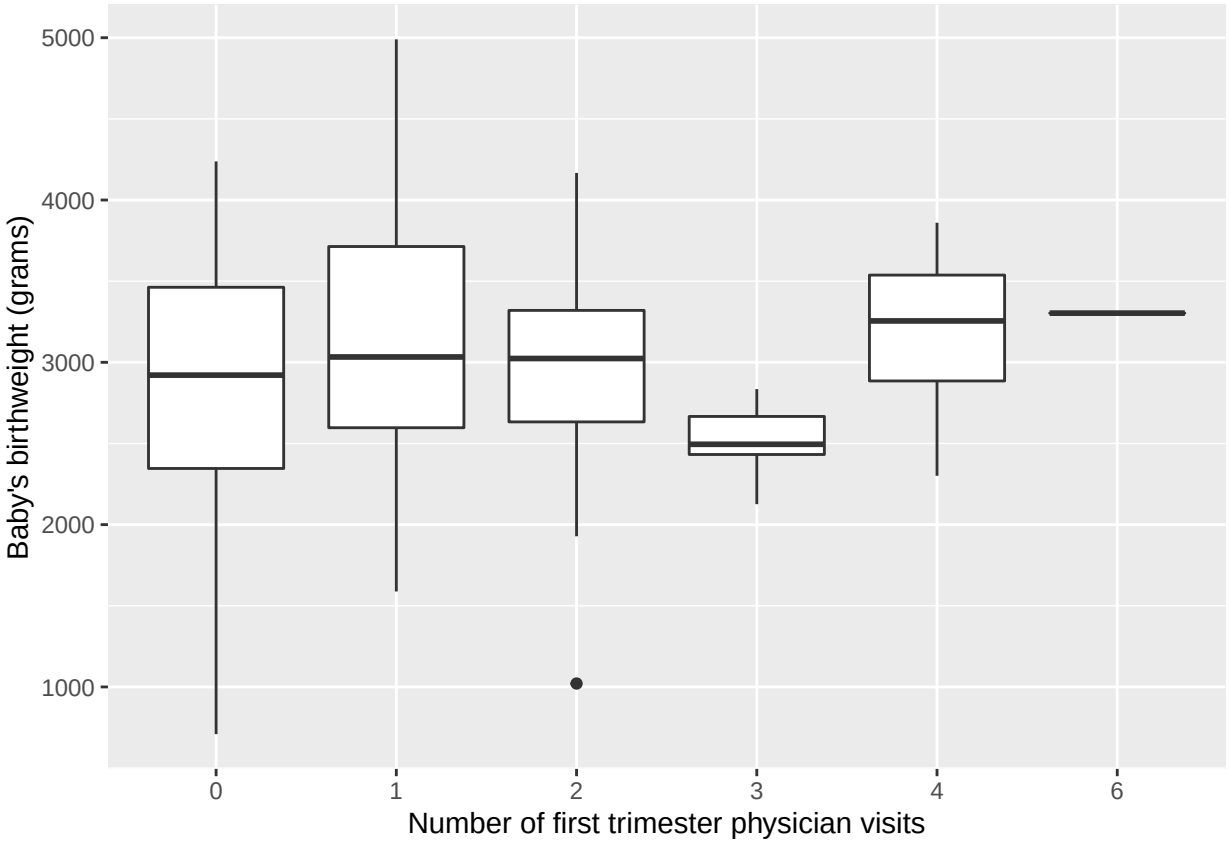

```
base.plot + geom_density(aes(fill = race), alpha = 0.5)
```



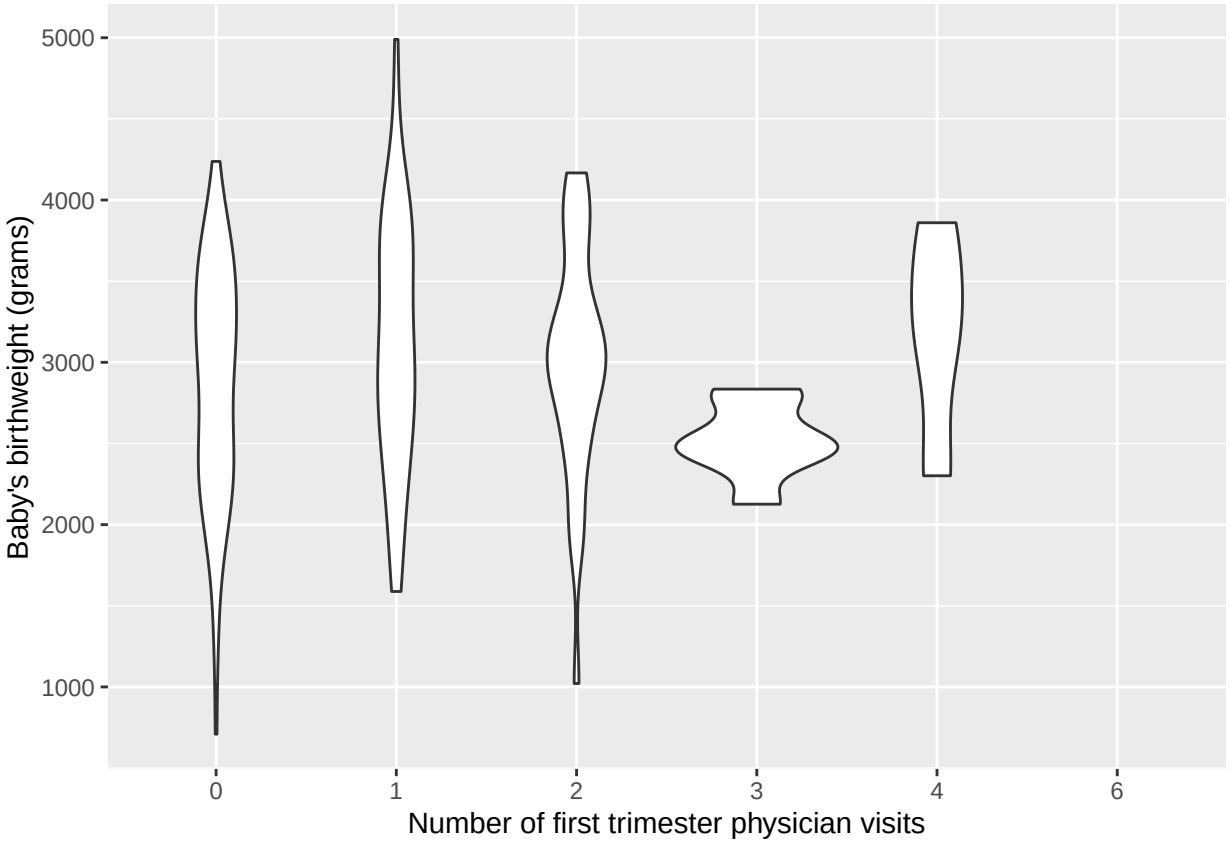
Box plots and violin plots

```
base.plot <- ggplot(birthwt, aes(x = as.factor(physician.visits), y = birthwt.grams), fill = physician.
  xlab("Number of first trimester physician visits") +
  ylab("Baby's birthweight (grams)")

# Box plot
base.plot + geom_boxplot()
```




```
# Violin plot
base.plot + geom_violin()
```



Visualizing means

Previously we calculated the following table:

```
tbl.mean.bwt <- birthwt %>%
  group_by(race, mother.smokes) %>%
  summarize(mean.birthwt = round(mean(birthwt.grams), 0))
tbl.mean.bwt
```

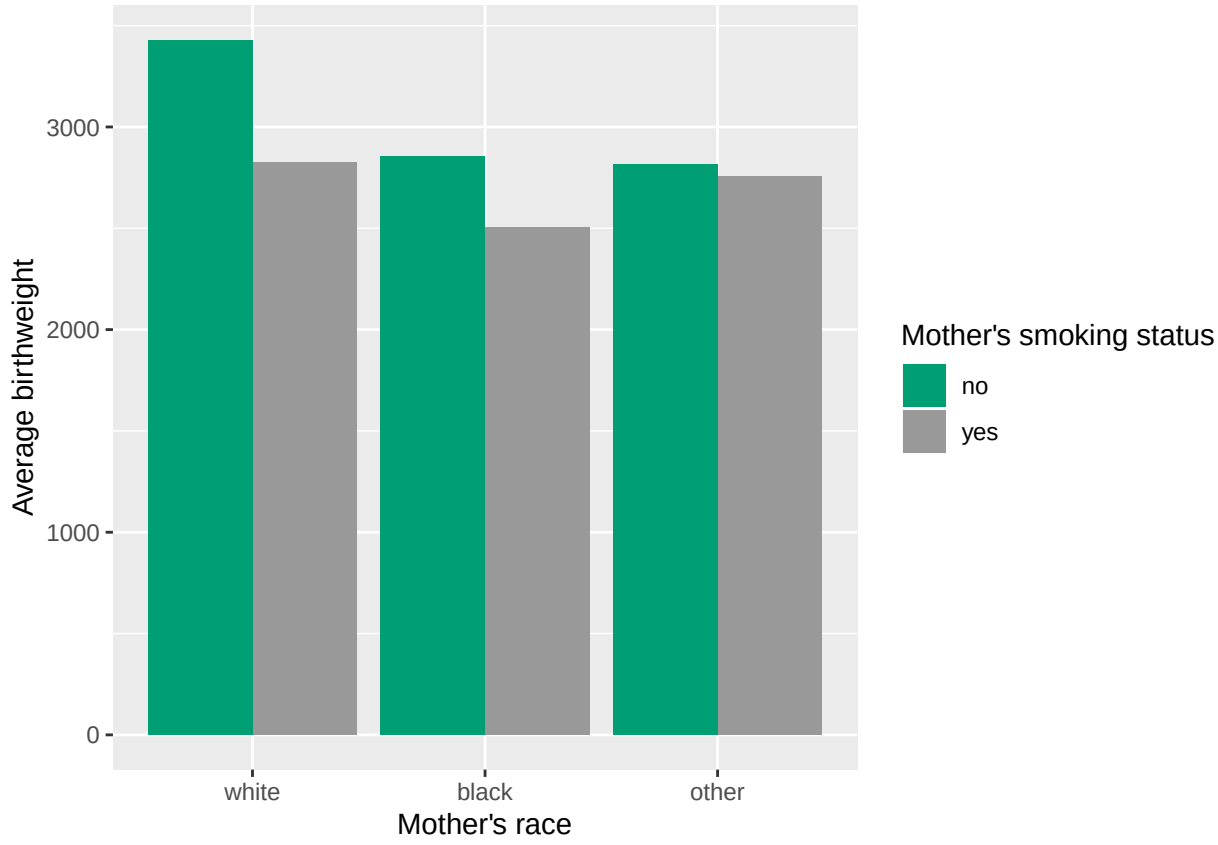
```
## # A tibble: 6 x 3
## # Groups:   race [3]
##   race  mother.smokes mean.birthwt
##   <fct> <fct>                <dbl>
## 1 white no                  3429
## 2 white yes                 2827
## 3 black no                  2854
## 4 black yes                 2504
## 5 other no                  2816
## 6 other yes                 2757
```

We can plot this table in a nice bar chart as follows:

```
# Define basic aesthetic parameters
p.bwt <- ggplot(data = tbl.mean.bwt,
                aes(y = mean.birthwt, x = race, fill = mother.smokes))

# Pick colors for the bars
bwt.colors <- c("#009E73", "#999999")
```

```
# Display barchart
p.bwt + geom_bar(stat = "identity", position = "dodge") +
  ylab("Average birthweight") +
  xlab("Mother's race") +
  guides(fill = guide_legend(title = "Mother's smoking status")) +
  scale_fill_manual(values=bwt.colors)
```



Question Does the association between birthweight and mother's age depend on smoking status?

We previously ran the following command to calculate the correlation between mother's ages and baby birthweights broken down by the mother's smoking status.

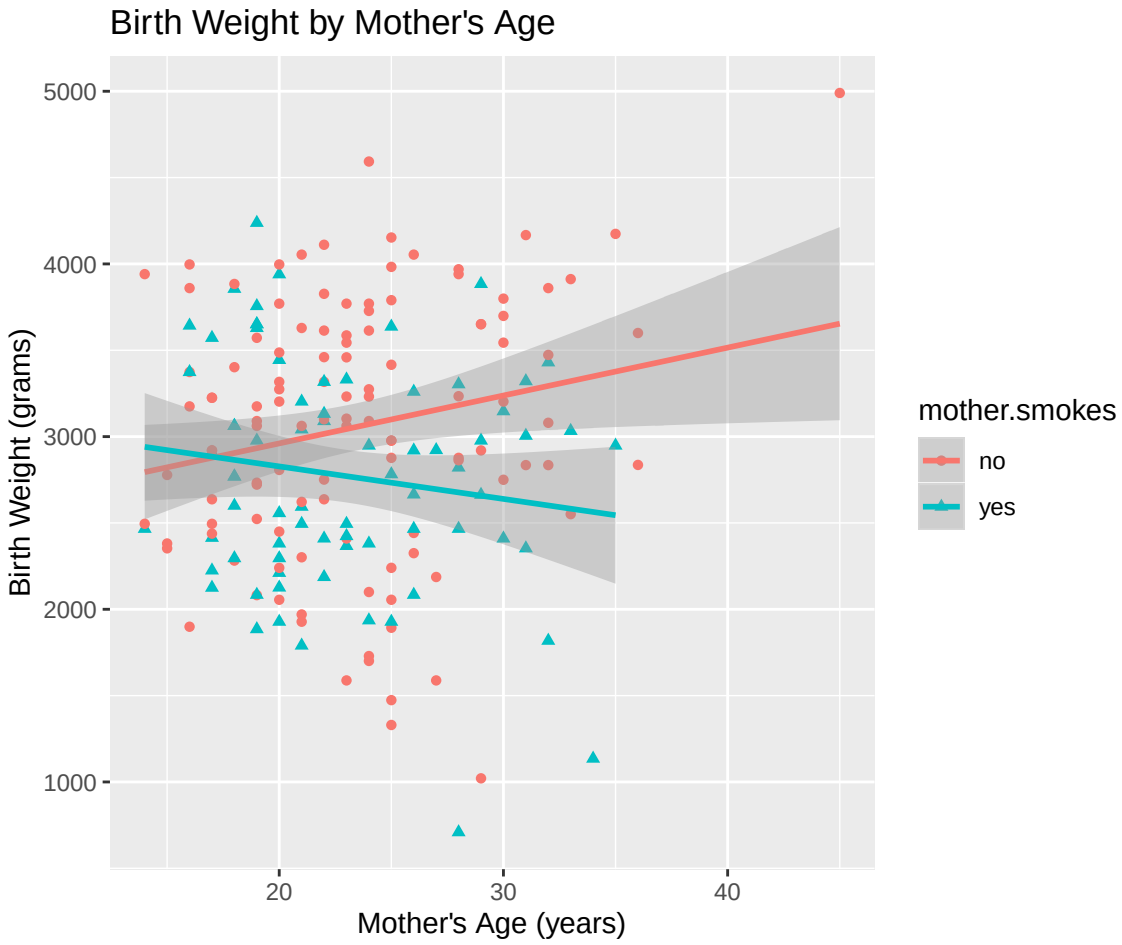
```
birthwt %>%
  group_by(mother.smokes) %>%
  summarize(cor_bwt_age = cor(birthwt.grams, mother.age))
```

```
## # A tibble: 2 x 2
##   mother.smokes cor_bwt_age
##   <fct>               <dbl>
## 1 no                  0.201
## 2 yes                -0.144
```

Here's a visualization of our data that allows us to see what's going on.

```
ggplot(birthwt,
       aes(x=mother.age, y=birthwt.grams, shape=mother.smokes, color=mother.smokes)) +
  geom_point() + # Adds points (scatterplot)
  geom_smooth(method = "lm") + # Adds regression lines
```

```
  ylab("Birth Weight (grams)") + # Changes y-axis label
  xlab("Mother's Age (years)") + # Changes x-axis label
  ggtitle("Birth Weight by Mother's Age") # Changes plot title
```



Reminders

- Homework 2 has been uploaded on shared folder and **due date is 4 March**.
- Lab 4 (uploaded on shared folder) - **due on Wednesday, 26 Februray 2020**
- If you have questions, feel free to post on the course group