Tutorial 5 - Data summaries and standard graphics

Content

- Summaries with the aggregate() function
- Standard graphics

Getting started: birthwt data set

- 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

```
library(MASS)
str(birthwt)
```

```
## 'data.frame':    189 obs. of  10 variables:
##  $ low  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ age  : int  19 33 20 21 18 21 22 17 29 26 ...
##  $ lwt  : int  182 155 105 108 107 124 118 103 123 113 ...
##  $ race : int  2 3 1 1 1 3 1 3 1 1 ...
##  $ smoke: int  0 0 1 1 1 0 0 0 1 1 ...
##  $ ptl  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ht   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ui   : int  1 0 0 1 1 0 0 0 0 0 ...
##  $ ftv  : int  0 3 1 2 0 0 1 1 1 0 ...
##  $ bwt  : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

Renaming the variables

- 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)

```
colnames(birthwt)
```

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

```
# The default names are not very descriptive

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

# Better names!
```

Renaming the factors

- All the factors are currently represented as integers
- Let's use the `transform()` and `mapvalues()` functions to convert variables to factors and give the factors more meaningful levels

```
library(plyr)
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                             c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                             c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                             c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                             c(0,1), c("no", "yes"))),
            birthwt.below.2500 = as.factor(mapvalues(birthwt.below.2500,
                             c(0,1), c("no", "yes")))
            )
```

Summary of the data

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

```
summary(birthwt)
```

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

A simple table

- Let's use the `tapply()` function to see what the average birthweight looks like when broken down by race and smoking status

```
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean))
```

```
##             no      yes
## black 2854.500 2504.000
## other 2815.782 2757.167
## white 3428.750 2826.846
```

What if we wanted nicer looking output?

- Let's use the header `{r, results='asis'}`, along with the `kable()` function from the `knitr` library

```
library(knitr)
bwt.tbl <- with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean))
kable(bwt.tbl, format = "markdown")
```

	no	yes
black	2854.500	2504.000
other	2815.782	2757.167
white	3428.750	2826.846

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

aggregate() function

- Let's first recall what `tapply()` does
- Command: `tapply(X, INDEX, FUN)`
  - Applies `FUN` to `X` grouped by factors in `INDEX`
- **`aggregate()`** performs a similar operation, but presents the results in a form that is at times more convenient
- There are many ways to call the `aggregate()` function
- Analog of `tapply` call: `aggregate(X, by, FUN)`
  - Here, `by` is exactly like `INDEX`

Example: tapply vs aggregate

```
library(MASS)
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean)) # tapply
```

```
##             no      yes
## black 2854.500 2504.000
## other 2815.782 2757.167
## white 3428.750 2826.846
```

```
with(birthwt, aggregate(birthwt.grams, by = list(race, mother.smokes), FUN = mean)) # aggregate
```

```
##   Group.1 Group.2        x
## 1   black      no 2854.500
## 2   other      no 2815.782
## 3   white      no 3428.750
## 4   black     yes 2504.000
## 5   other     yes 2757.167
## 6   white     yes 2826.846
```

Example: different syntax

- Here's a convenient alternative way to call `aggregate`
- It uses the R `formula` syntax, which we'll learn more about when we discuss regression

```
aggregate(birthwt.grams ~ race + mother.smokes, FUN=mean, data=birthwt)
```

```
##    race mother.smokes birthwt.grams
## 1 black            no      2854.500
## 2 other            no      2815.782
## 3 white            no      3428.750
## 4 black           yes      2504.000
## 5 other           yes      2757.167
## 6 white           yes      2826.846
```

- We'll see later that `aggregate` output can be more convenient for plotting

A closer look at low birth weight

```
weight.smoke.tbl <- with(birthwt, table(birthwt.below.2500, mother.smokes))
weight.smoke.tbl
```

```
##                   mother.smokes
## birthwt.below.2500 no yes
##                no  86  44
##                yes 29  30
```

- The odds of low bwt among non-smoking mothers is

```
or.smoke.bwt <- (weight.smoke.tbl[2,2] / weight.smoke.tbl[1,2]) / (weight.smoke.tbl[2,1] / weigh
t.smoke.tbl[1,1])
or.smoke.bwt
```

```
## [1] 2.021944
```

- So the odds of low birth weight are 2 times higher when the mother smokes

continuted…

- Is the mother's age correlated with birth weight?

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

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

- Does this change when we account for smoking status?

```
with(birthwt, cor(birthwt.grams[mother.smokes == "yes"], mother.age[mother.smokes == "yes"]))
```

```
## [1] -0.1441649
```

```
with(birthwt, cor(birthwt.grams[mother.smokes == "no"], mother.age[mother.smokes == "no"]))
```

```
## [1] 0.2014558
```

Faster way: by() function

- Think of the `by(data, INDICES, FUN)` function as a `tapply()` function that operates on data frames instead of just vectors
- When using `tapply(X, INDEX, FUN)`, `X` is generally a numeric vector
- To calculate correlations, we need to allow `X` to be a data frame or matrix

```
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

```
## mother.smokes: no
## [1] 0.2014558
## ------------------------------------------------------------
## mother.smokes: yes
## [1] -0.1441649
```

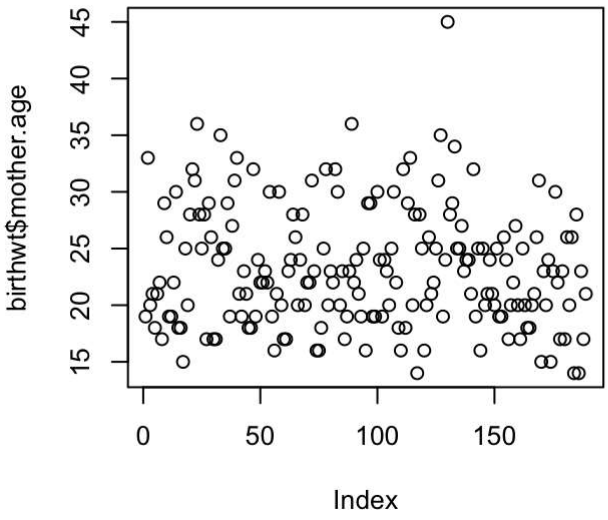
Standard graphics in R

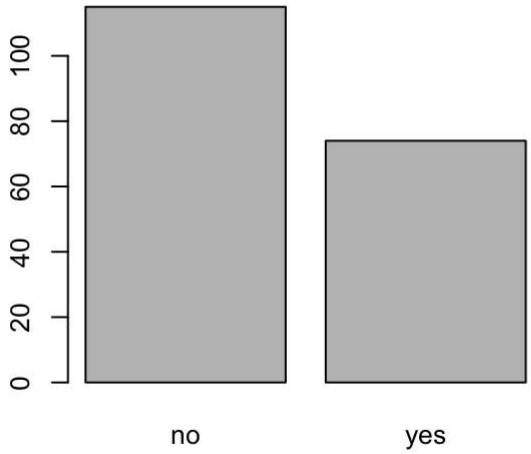
Single-variable plots

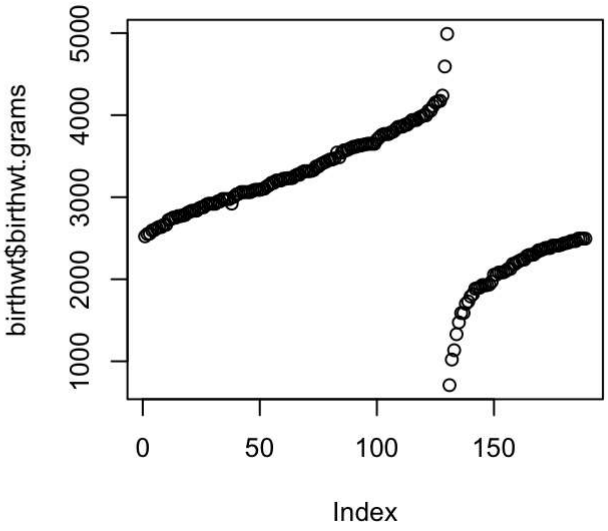
Let's continue with the `birthwt` data from the `MASS` library.

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)
```



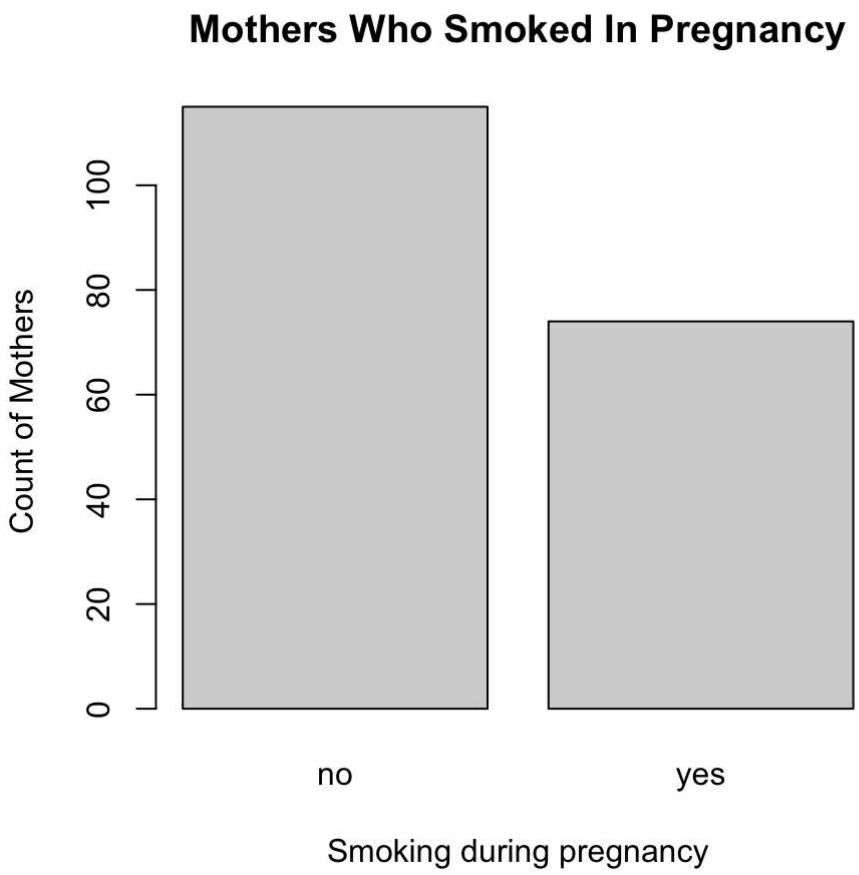




Note that the result of calling `plot(x, ...)` varies depending on what `x` is.
- When `x` is *numeric*, you get a plot showing the value of `x` at every index.
- When `x` is a *factor*, you get a bar plot of counts for every level

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 = "lightgrey")
```



(much) better graphics with ggplot2

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:

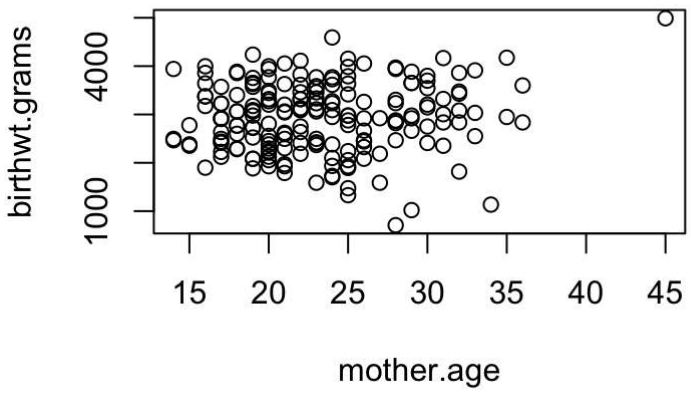
- `qplot(x, y, ..., data)` : a "quick-plot" routine, which essentially replaces the base `plot()`
- `ggplot(data, aes(x, y, ...), ...)` : defines a graphics object from which plots can be generated, along with *aesthetic mappings* that specify how variables are mapped to visual properties.

```
library(ggplot2)
```

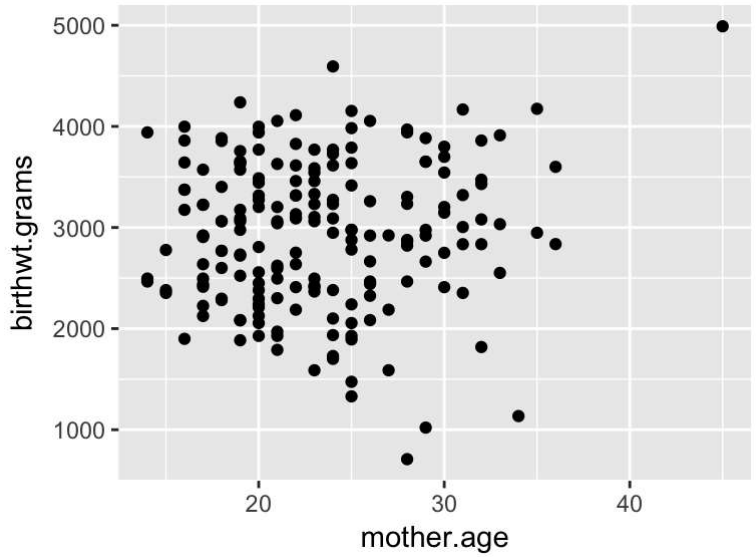
plot vs qplot

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
```

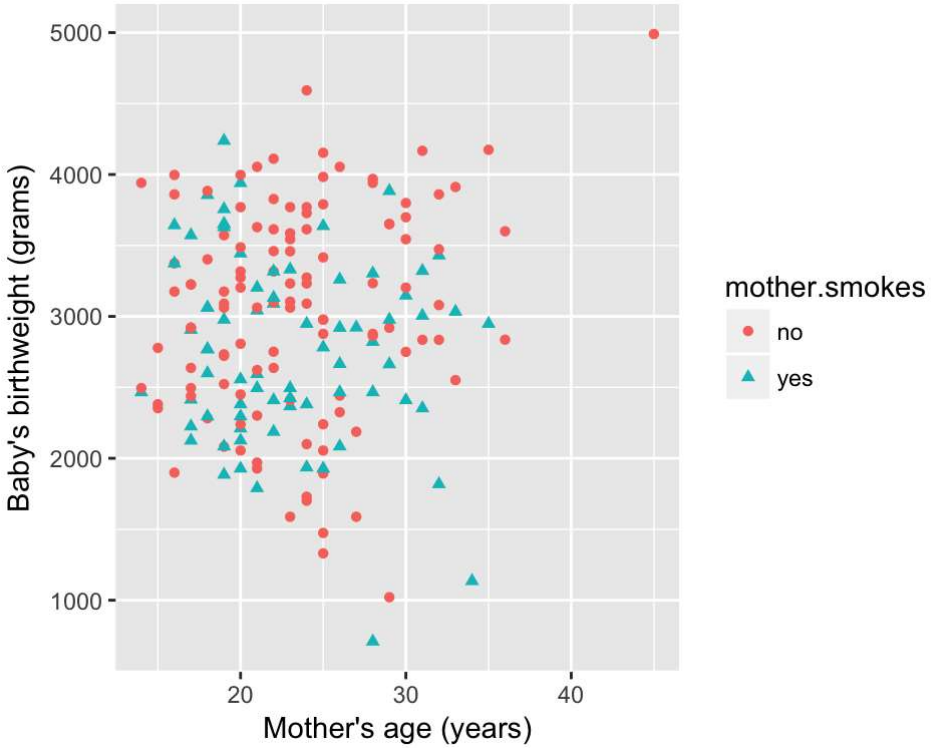



```
qplot(x=mother.age, y=birthwt.grams, data=birthwt)  # using qplot from ggplot2
```



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
```

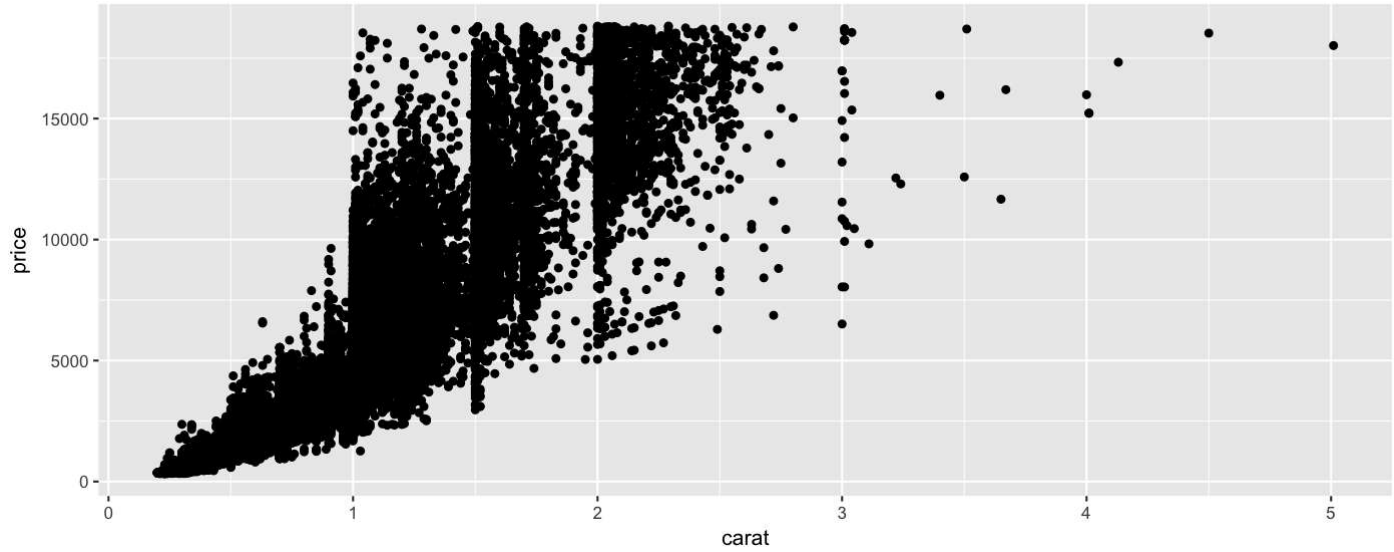
```
head(diamonds)
```

```
## # 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  0.29   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
```

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.

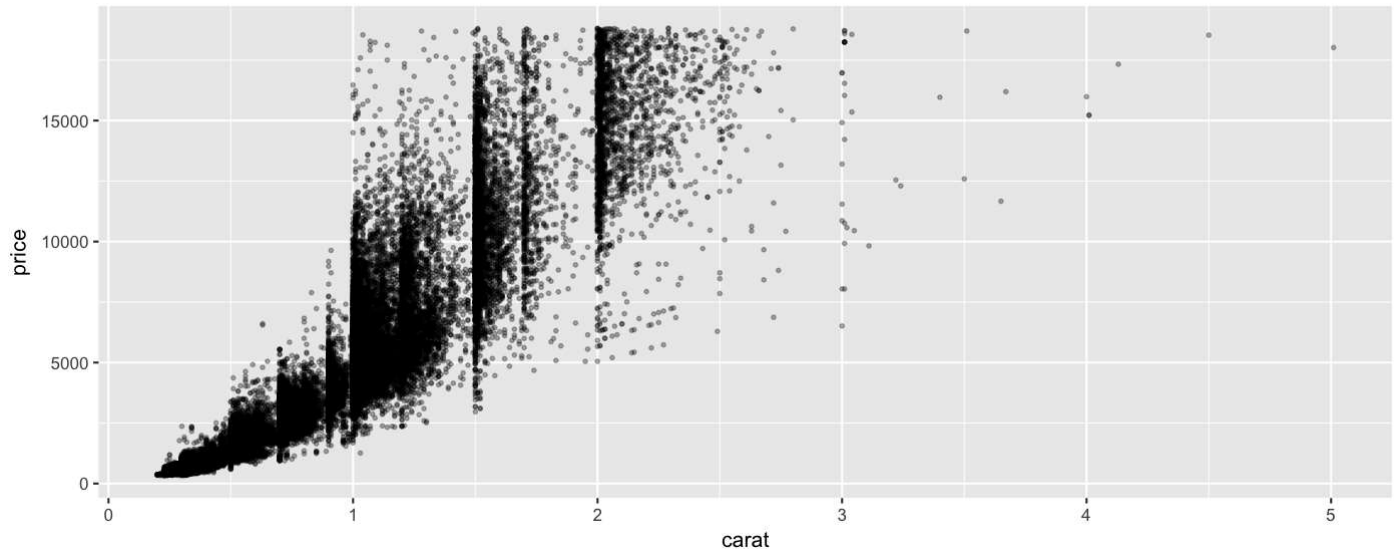
1. The first thing we did was to define a graphics object, `diamond.plot`. This definition told R that we're using the `diamonds` data, and that we want to display `carat` on the x-axis, and `price` on the y-axis.
2. We then called `diamond.plot + geom_point()` to get a scatterplot.

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.

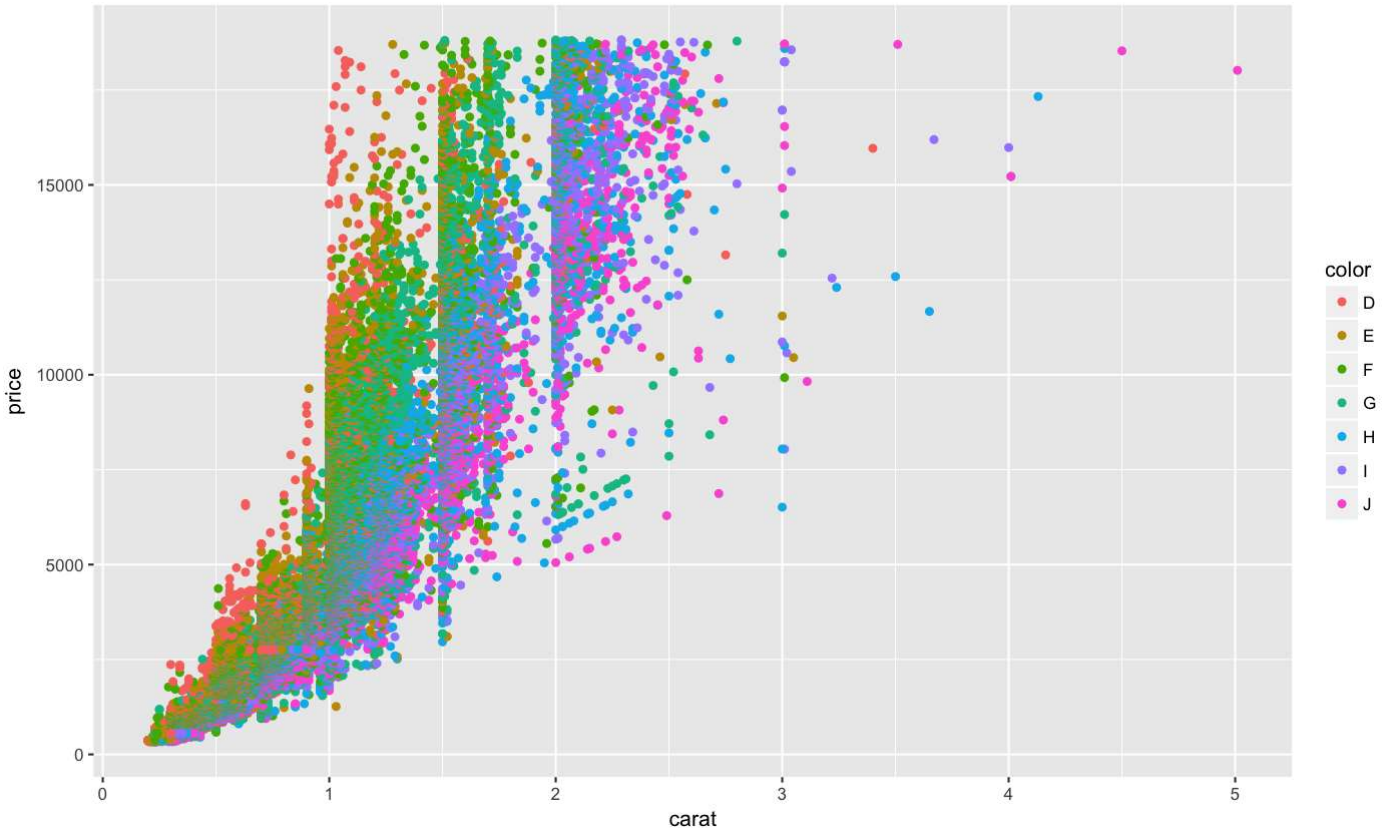
Here we shrink the points to a smaller size, and use the `alpha` argument to make the points transparent.

```
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.

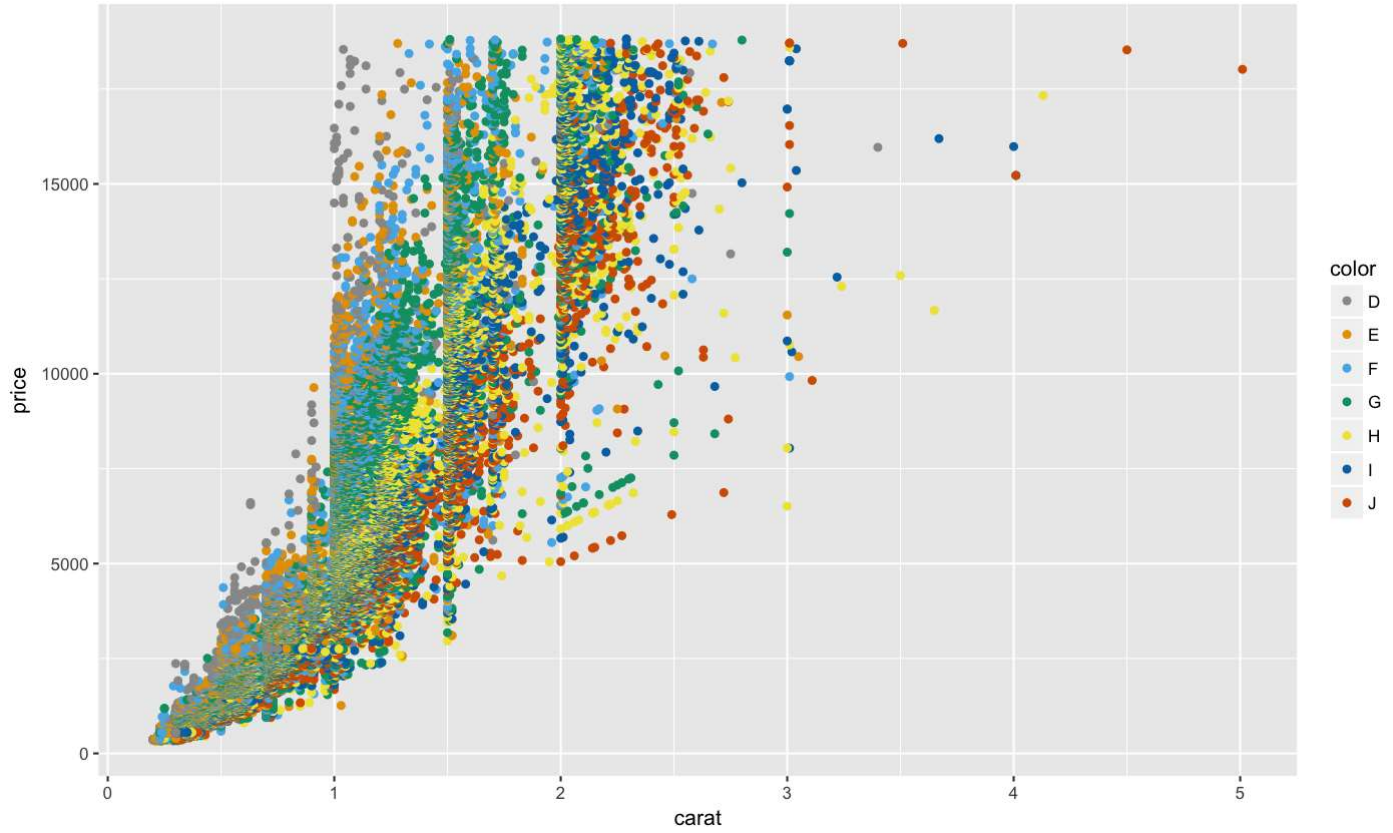
```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point()
```



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", "#CC
79A7")
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.

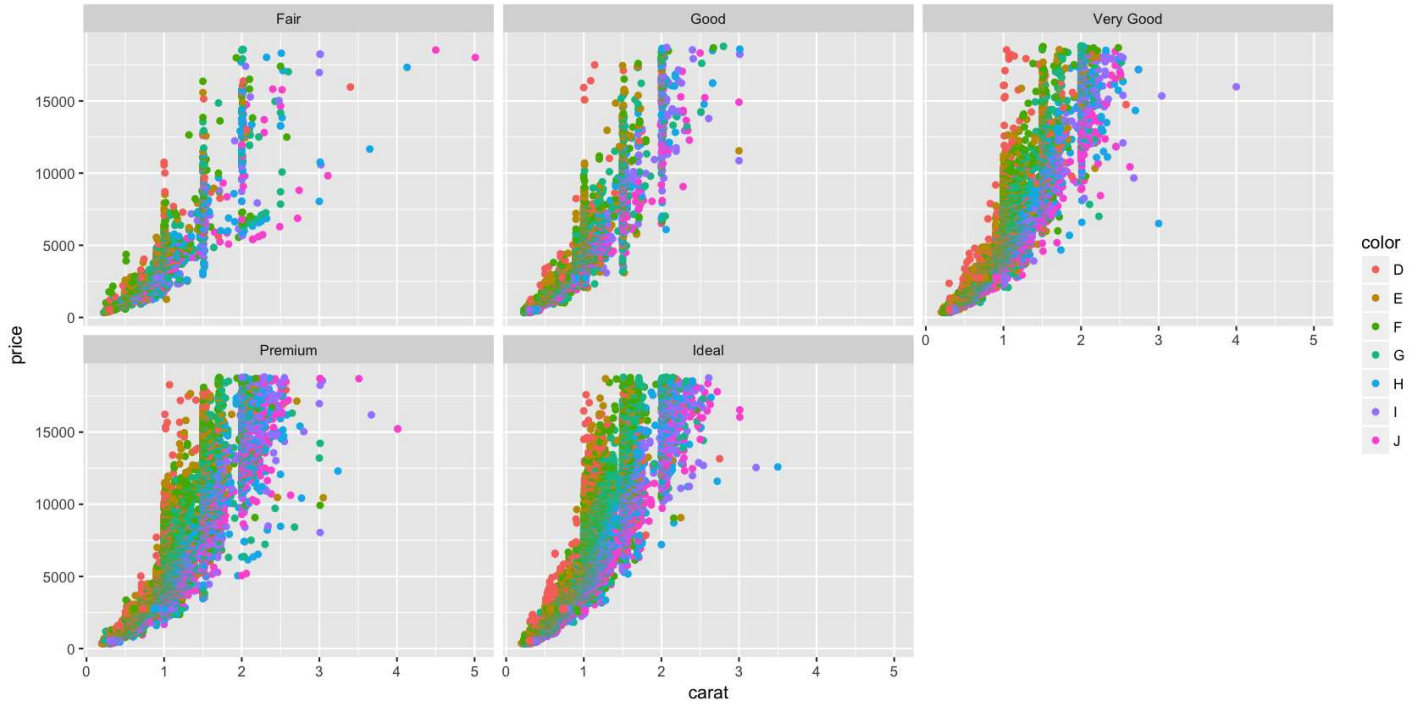
```
diamond.plot + geom_point() +
  coord_trans(x = "log10", y = "log10")
```

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.

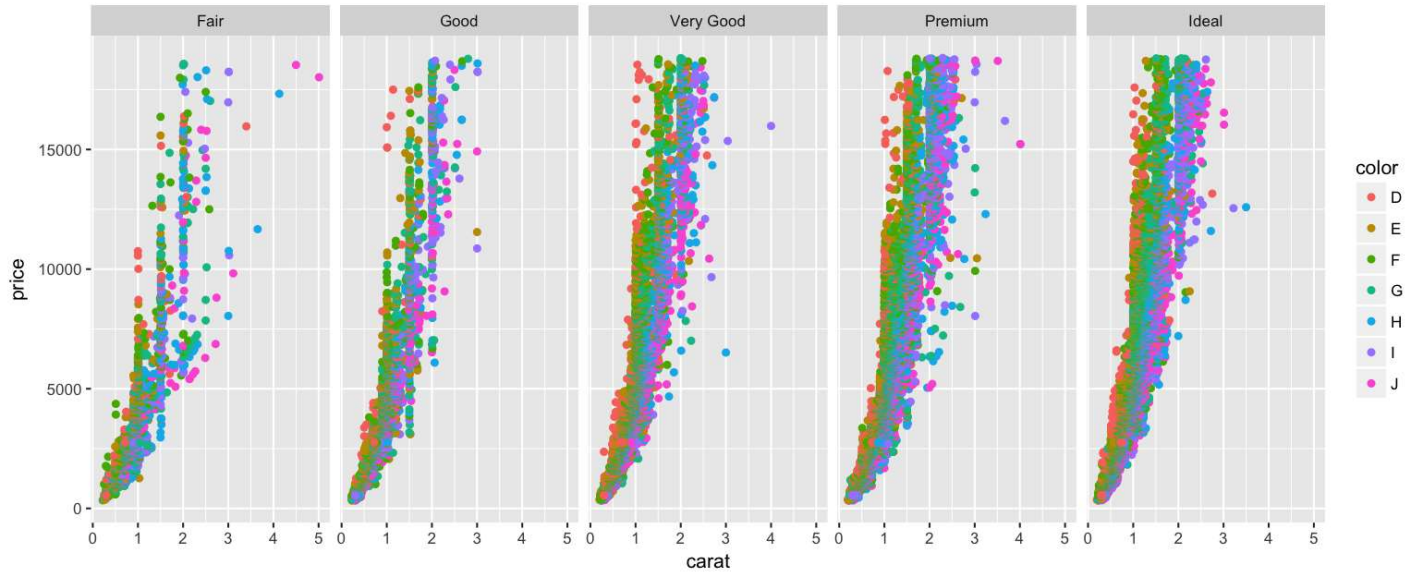
```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))

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

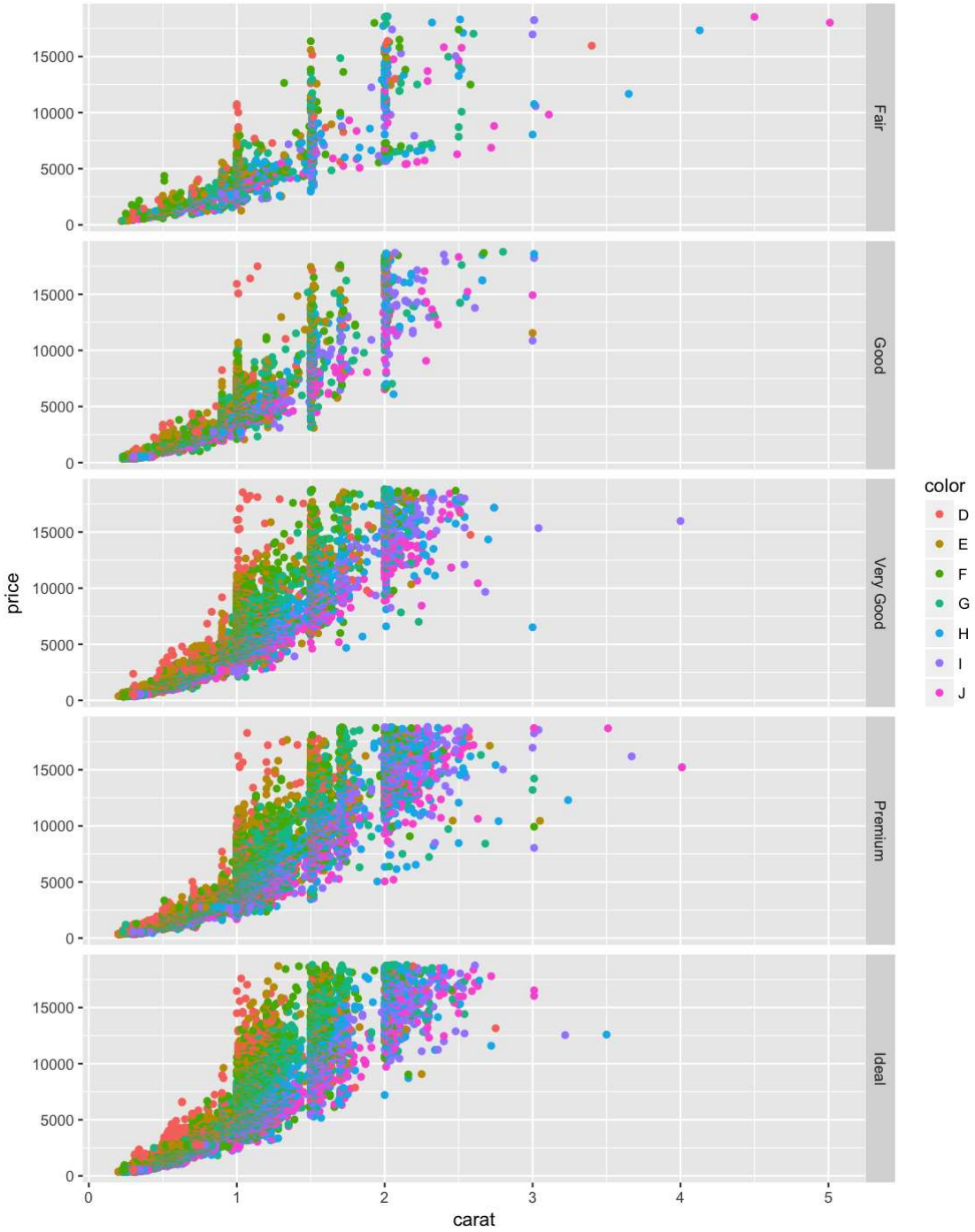


You can also use `facet_grid()` to produce this type of output.

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




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



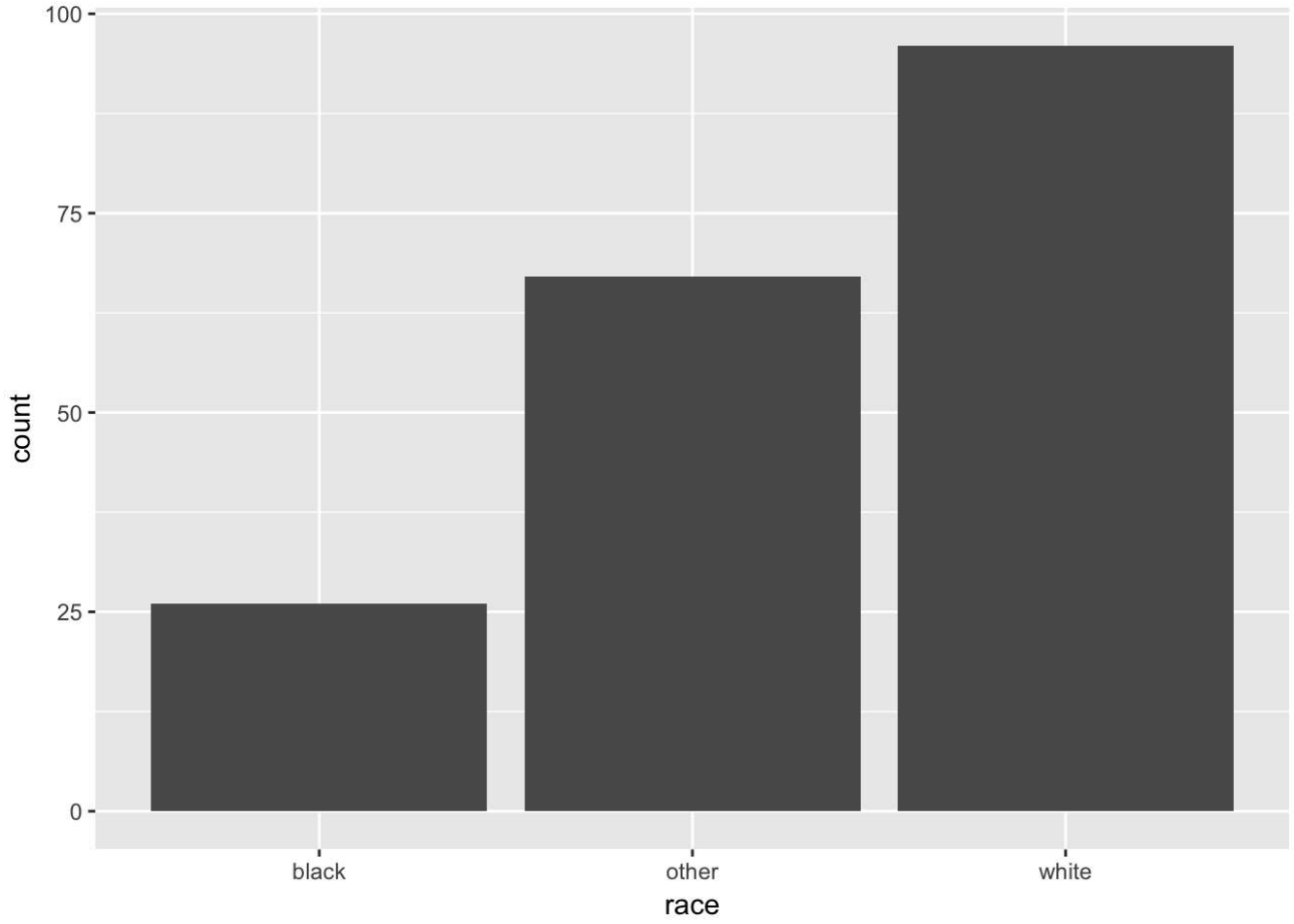
`ggplot` can create a lot of different kinds of plots, just like lattice. Here are some examples.

Function	Description
`geom_point(...)`	Points, i.e., scatterplot
`geom_bar(...)`	Bar chart
`geom_line(...)`	Line chart
`geom_boxplot(...)`	Boxplot
`geom_violin(...)`	Violin plot
`geom_density(...)`	Density plot with one variable

Function	Description
`geom_density2d(...)`	Density plot with two variables
`geom_histogram(...)`	Histogram

A bar chart

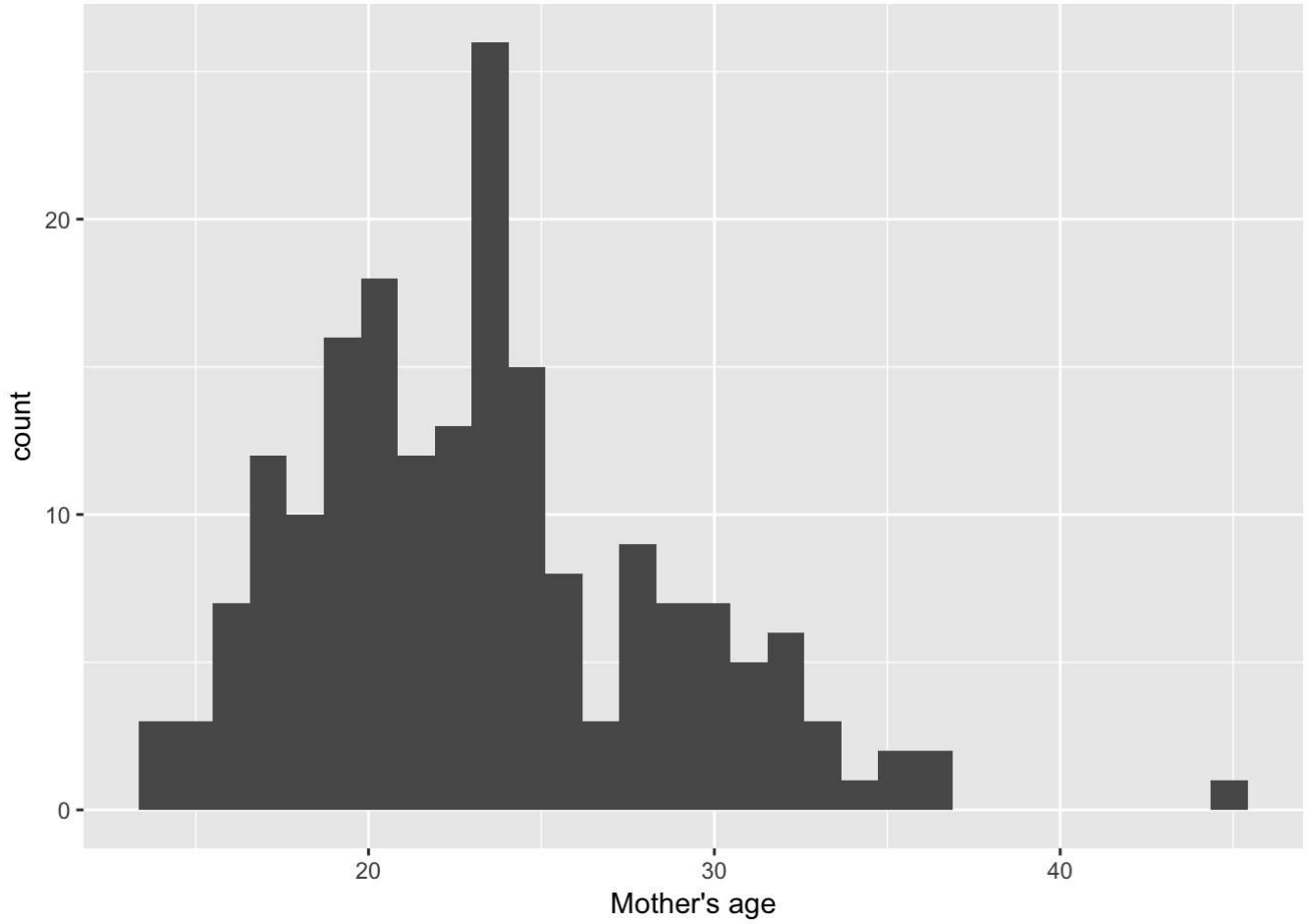
```
qplot(x = race, data = birthwt, geom = "bar")
```



Histograms and density plots

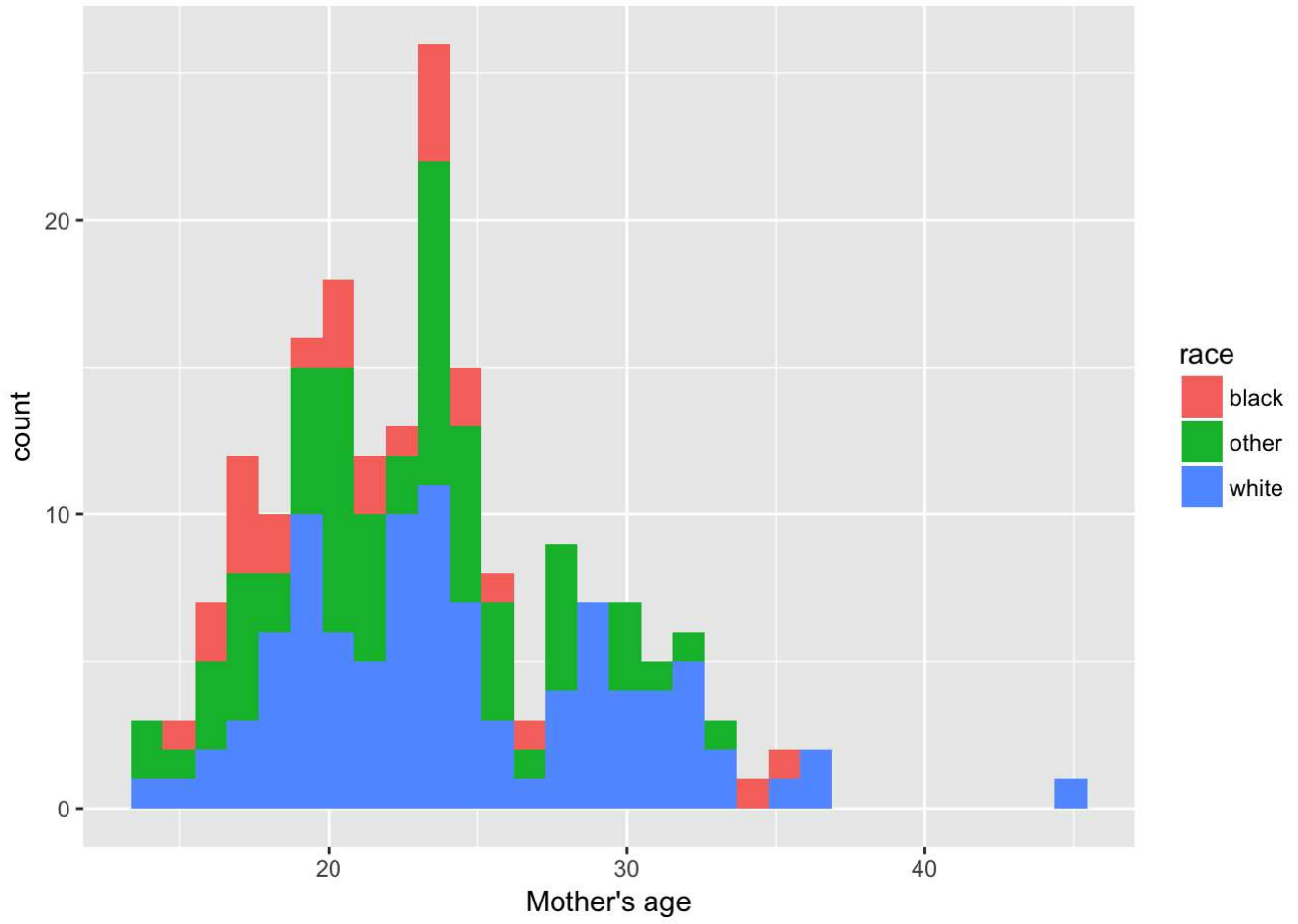
```
base.plot <- ggplot(birthwt, aes(x = mother.age)) +
  xlab("Mother's age")
base.plot + geom_histogram()
```

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

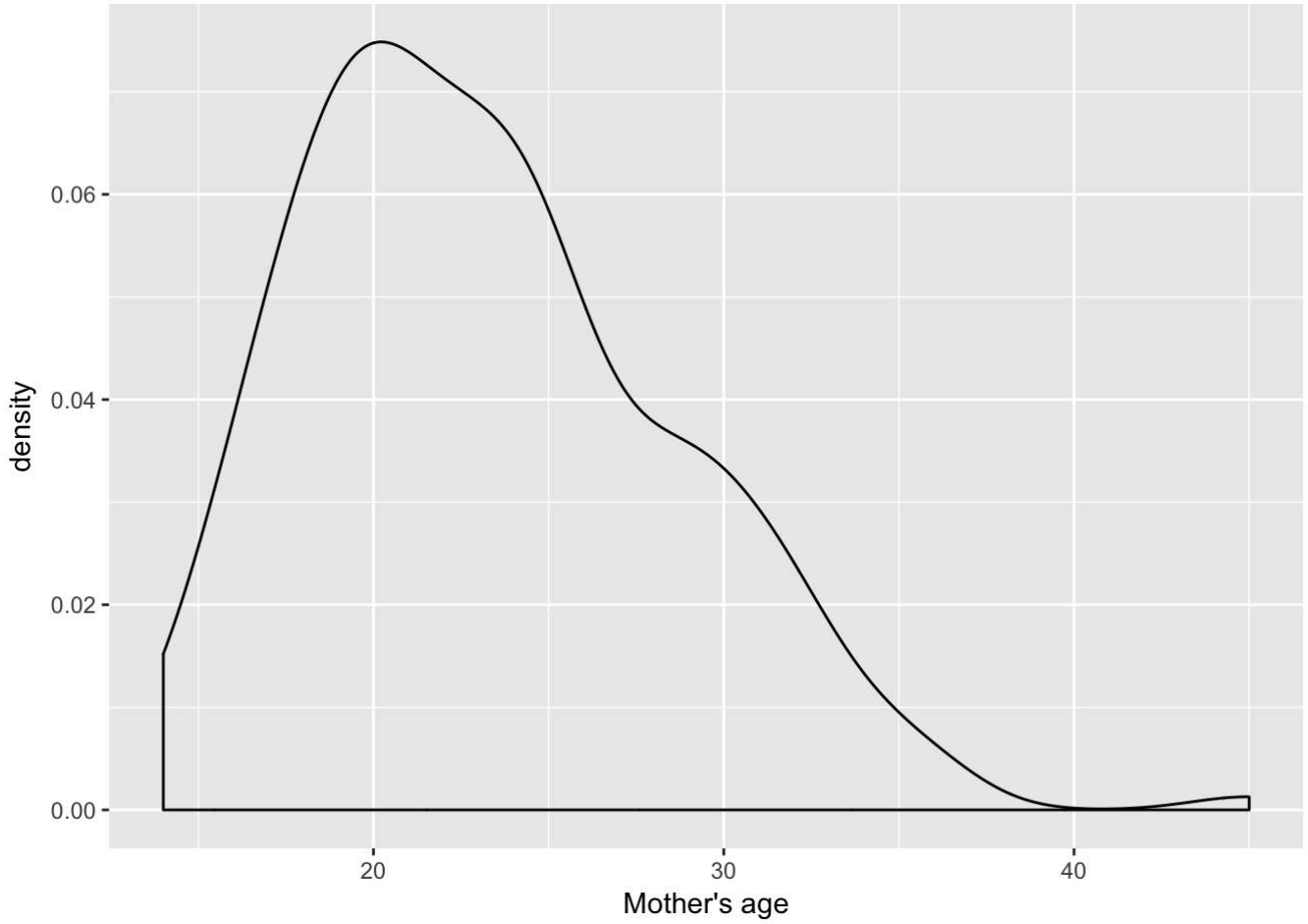



```
base.plot + geom_histogram(aes(fill = race))
```

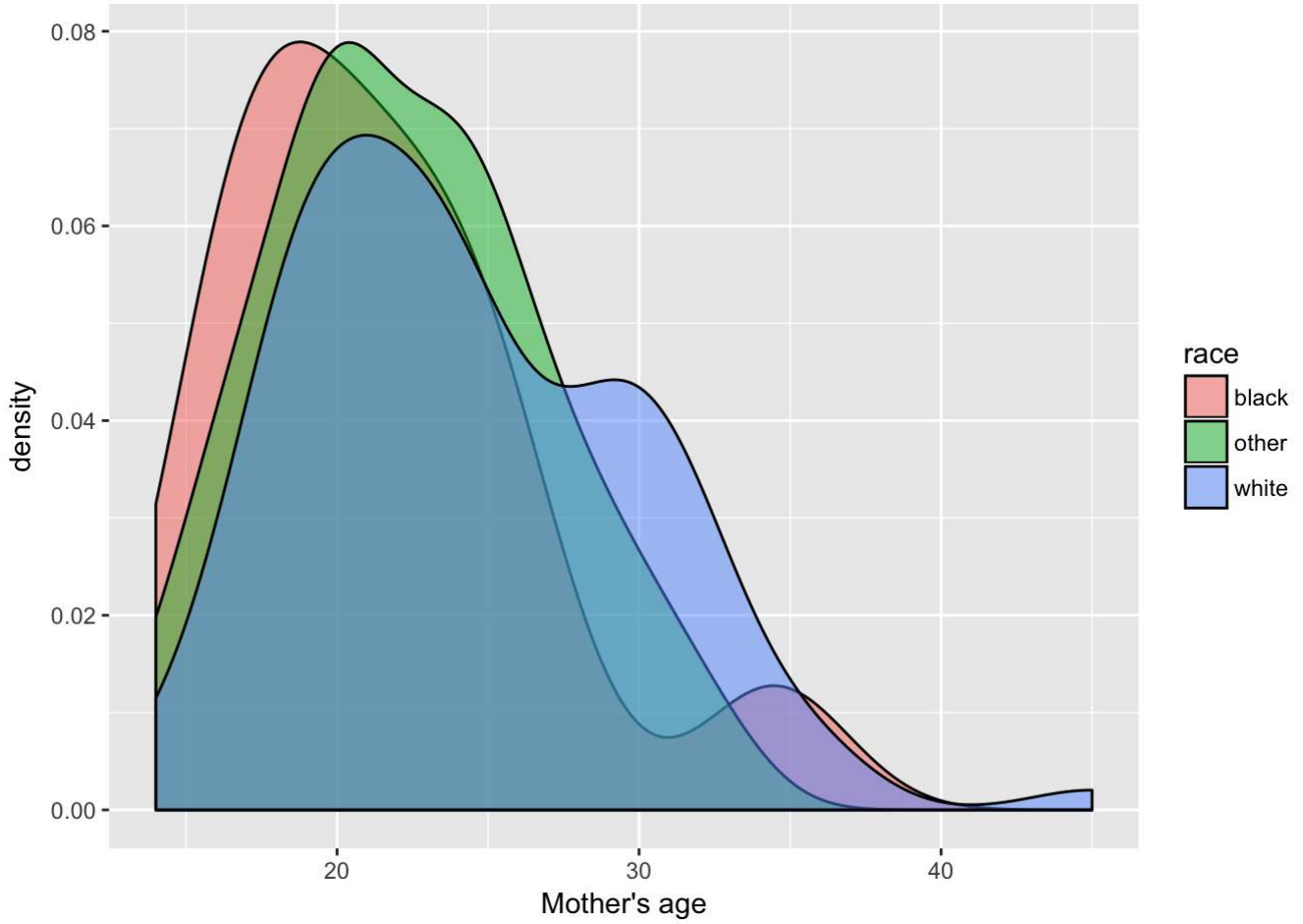
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




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



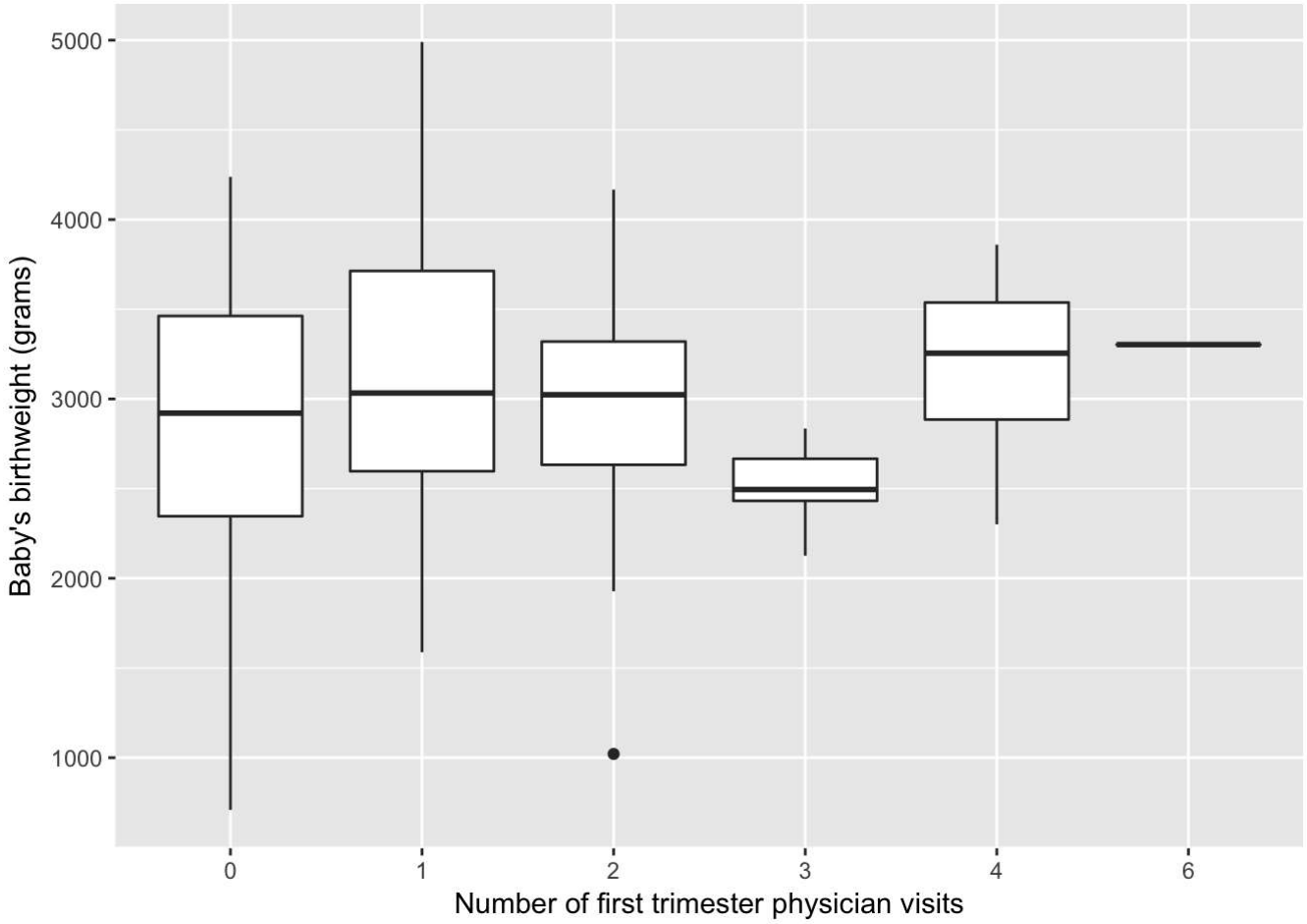

```
base.plot + geom_density(aes(fill = race), alpha = 0.5)
```



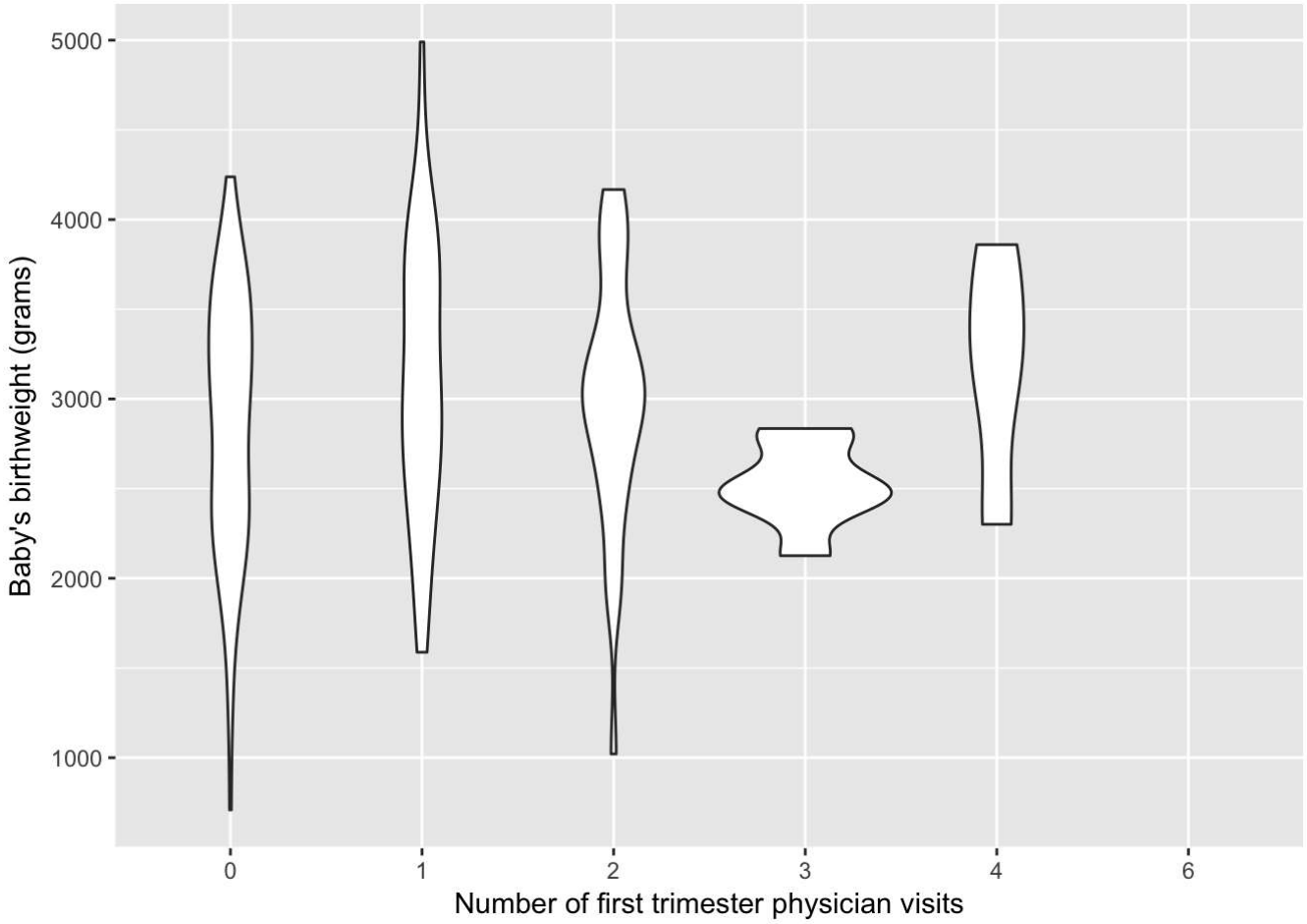
Box plots and violin plots

```
base.plot <- ggplot(birthwt, aes(x = as.factor(physician.visits), y = birthwt.grams)) +
  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:

```
bwt.summary <- aggregate(birthwt.grams ~ race + mother.smokes, data = birthwt, FUN = mean) # agg
regate
bwt.summary
```

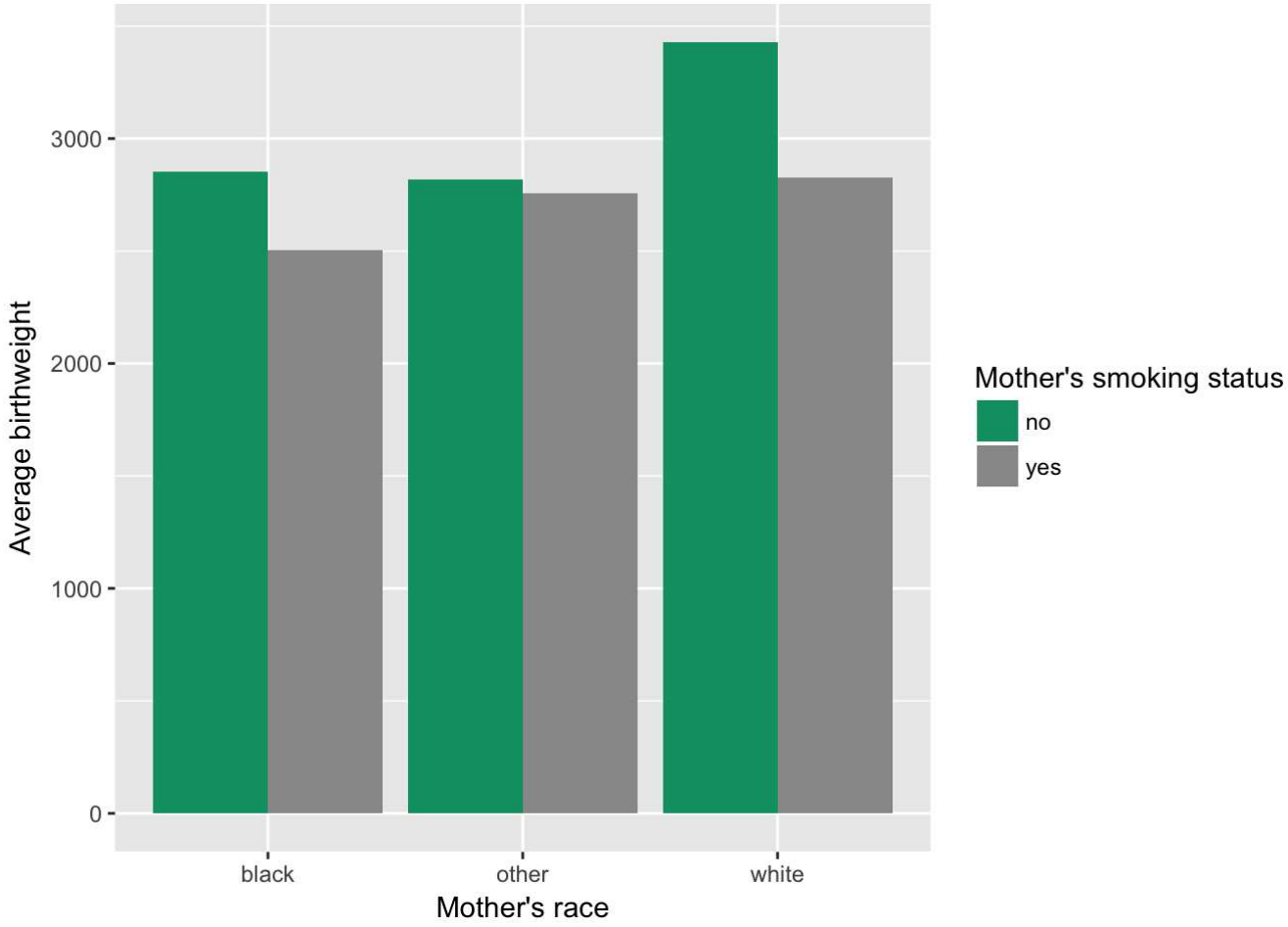
```
##    race mother.smokes birthwt.grams
## 1 black            no      2854.500
## 2 other            no      2815.782
## 3 white            no      3428.750
## 4 black           yes      2504.000
## 5 other           yes      2757.167
## 6 white           yes      2826.846
```

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

```
# Define basic aesthetic parameters
p.bwt <- ggplot(data = bwt.summary, aes(y = birthwt.grams, 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)
```



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

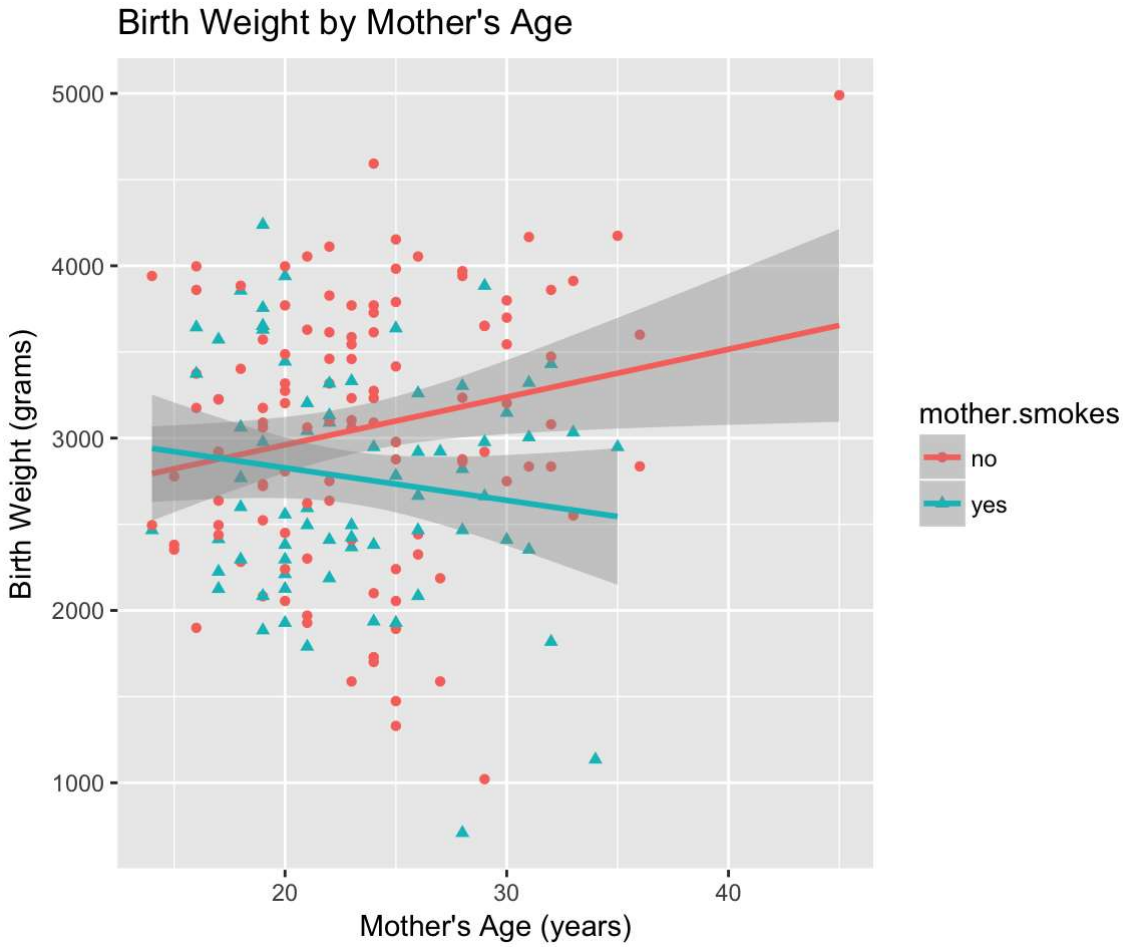
We previously ran the following command to calculate the correlation between mother's ages and baby birthweights.

```
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

```
## mother.smokes: no
## [1] 0.2014558
## ------------------------------------------------------------
## mother.smokes: yes
## [1] -0.1441649
```

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
```



Next

- Complete **Lab 5** (http://isle.heinz.cmu.edu/94-842/lab05/)