---

title: “ggplot2” author: “Veerasak Kritsanapraphan” date: “January 28, 2016” output: html_document: fig_height: 5 fig_width: 5 toc: yes toc_depth: 5 pdf_document: toc: yes word_document: default —

Exploratory Data Analysis

Let’s begin by loading the packages we’ll use this class

```
library(ggplot2)
library(MASS)  # Useful for data sets
library(plyr)  # We'll need mapvalues
```

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.

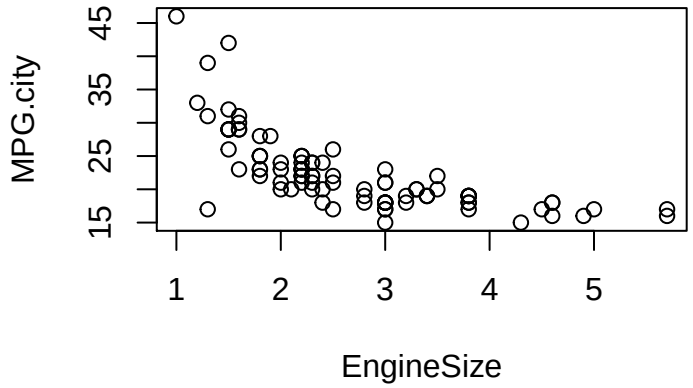
There are two basic calls in ggplot:

- `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.

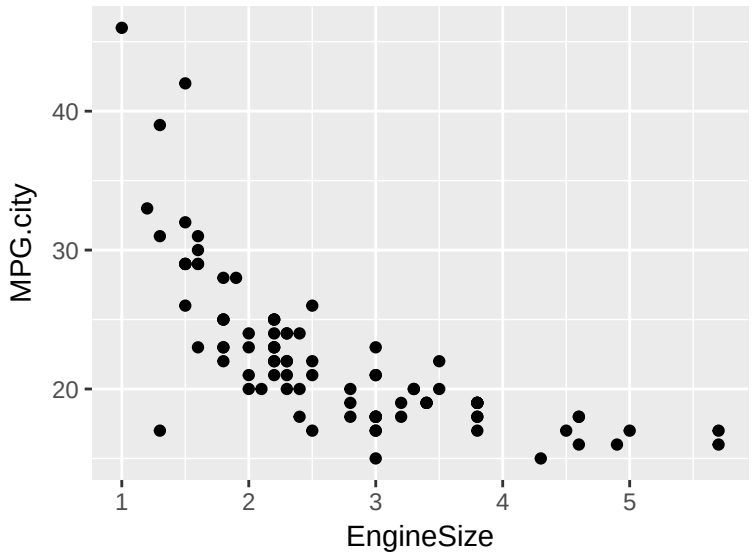
plot vs qplot

Here’s how the default scatterplots look in ggplot compared to the base graphics. We’ll illustrate things using the Cars93 data from the `MASS` library.

```
with(Cars93, plot(EngineSize, MPG.city))  # Base graphics
```

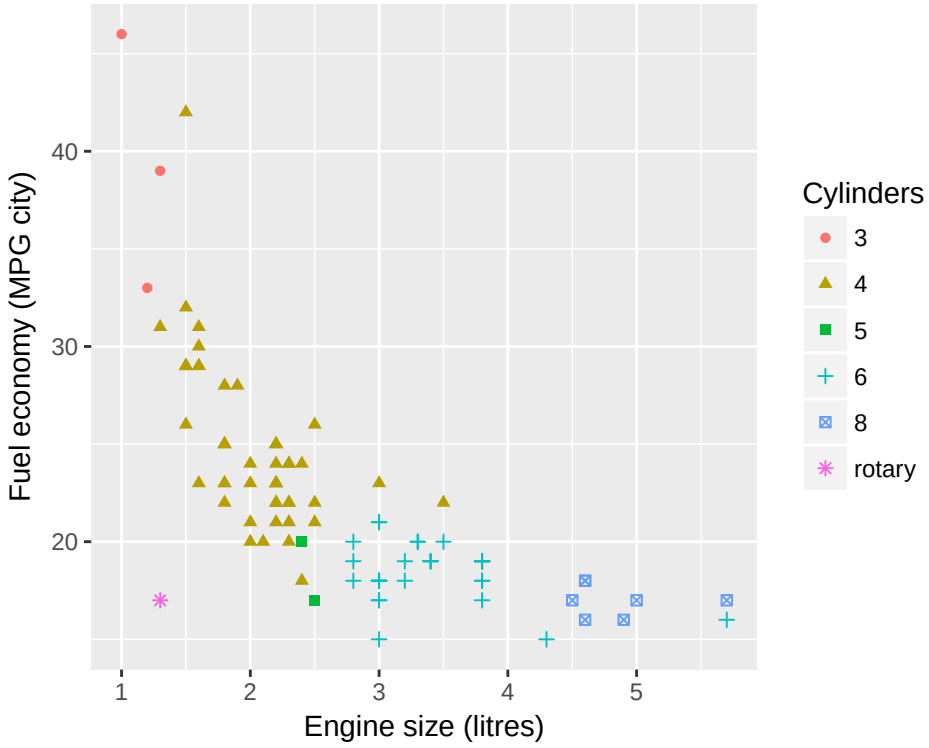



```
qplot(x=EngineSize, y=MPG.city, data=Cars93)  # using qplot from ggplot2
```



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=EngineSize, y=MPG.city, data=Cars93,
      colour=Cylinders,
      shape=Cylinders,
      xlab = "Engine size (litres)",
      ylab = "Fuel economy (MPG city)"
      )
```



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
```

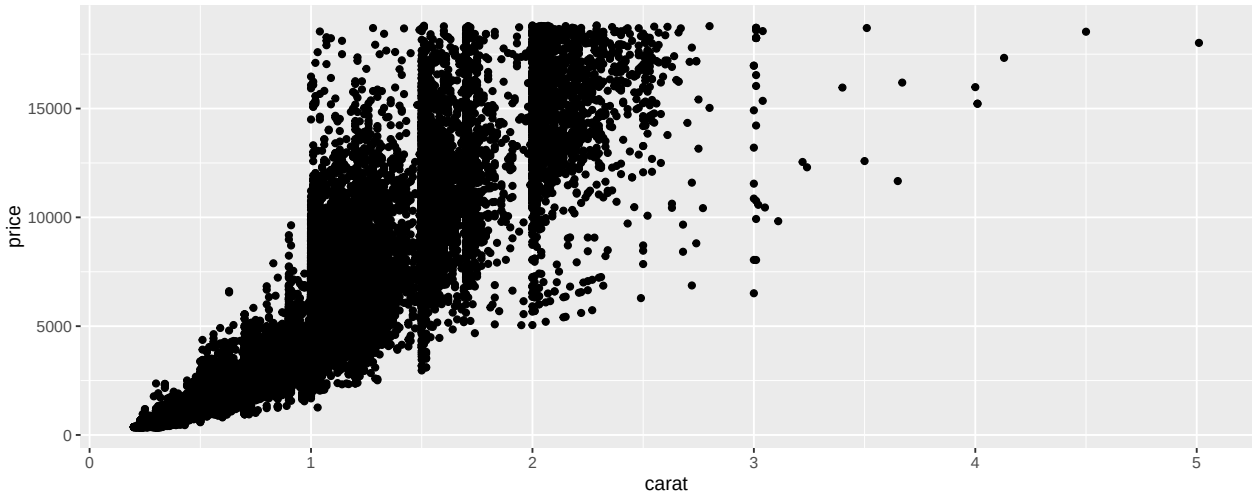
```
head(diamonds)
```

```
## # 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 0.230 Ideal     E     SI2      61.5   55.   326  3.95  3.98  2.43
## 2 0.210 Premium   E     SI1      59.8   61.   326  3.89  3.84  2.31
## 3 0.230 Good      E     VS1      56.9   65.   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4   58.   334  4.20  4.23  2.63
## 5 0.310 Good      J     SI2      63.3   58.   335  4.34  4.35  2.75
## 6 0.240 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.

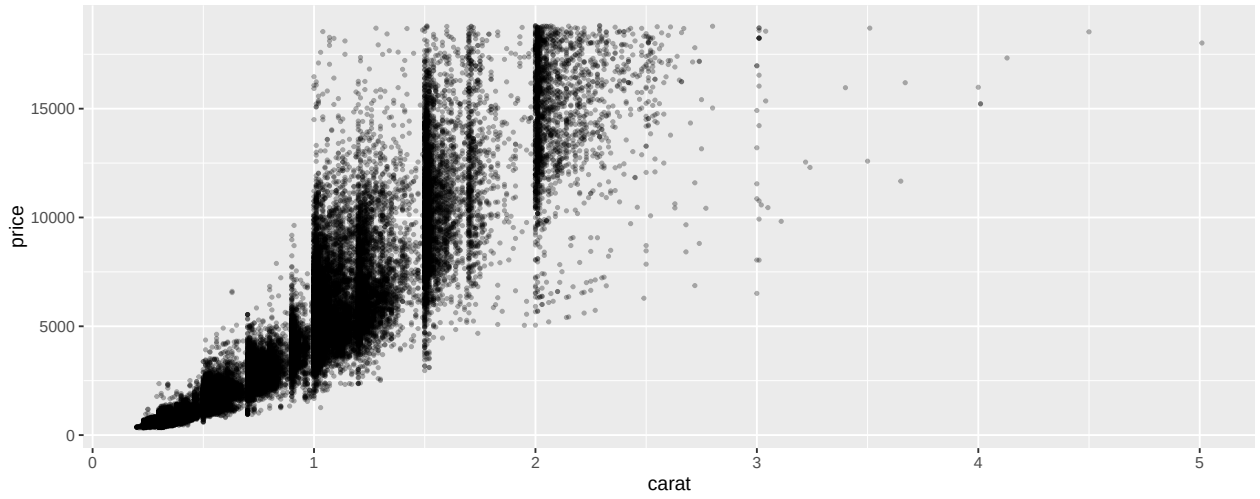
2) We then called `diamond.plot + geom_point()` to get a scatterplot.

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.

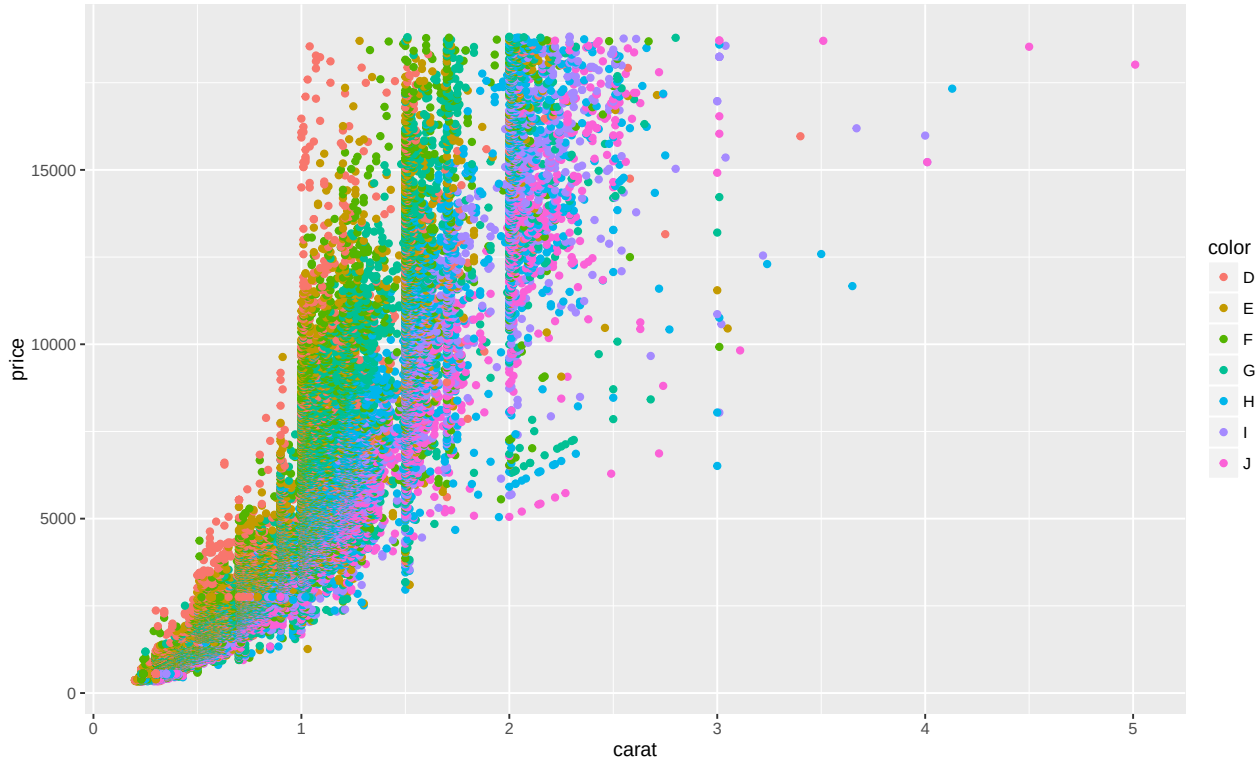
Here we shrink the points to a smaller size, and use the `alpha` argument to make the points transparent.

```
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.

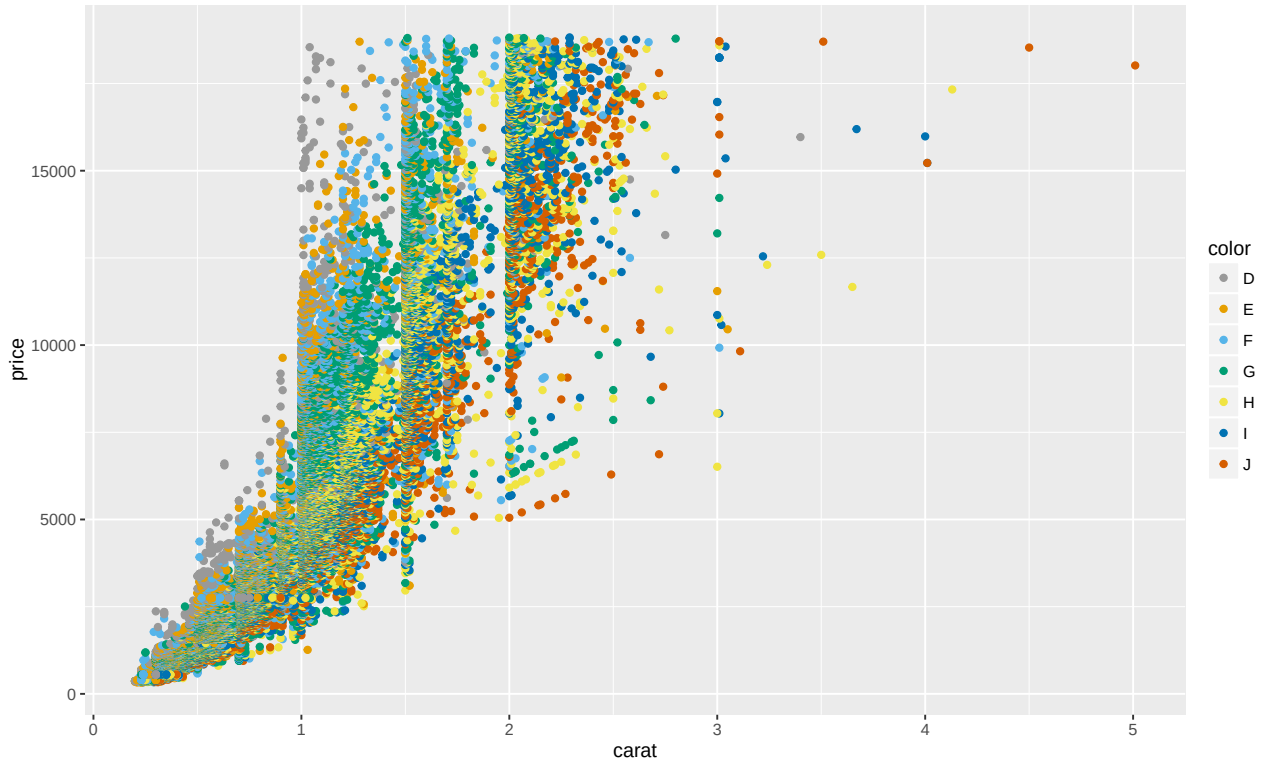
```
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point()
```



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.

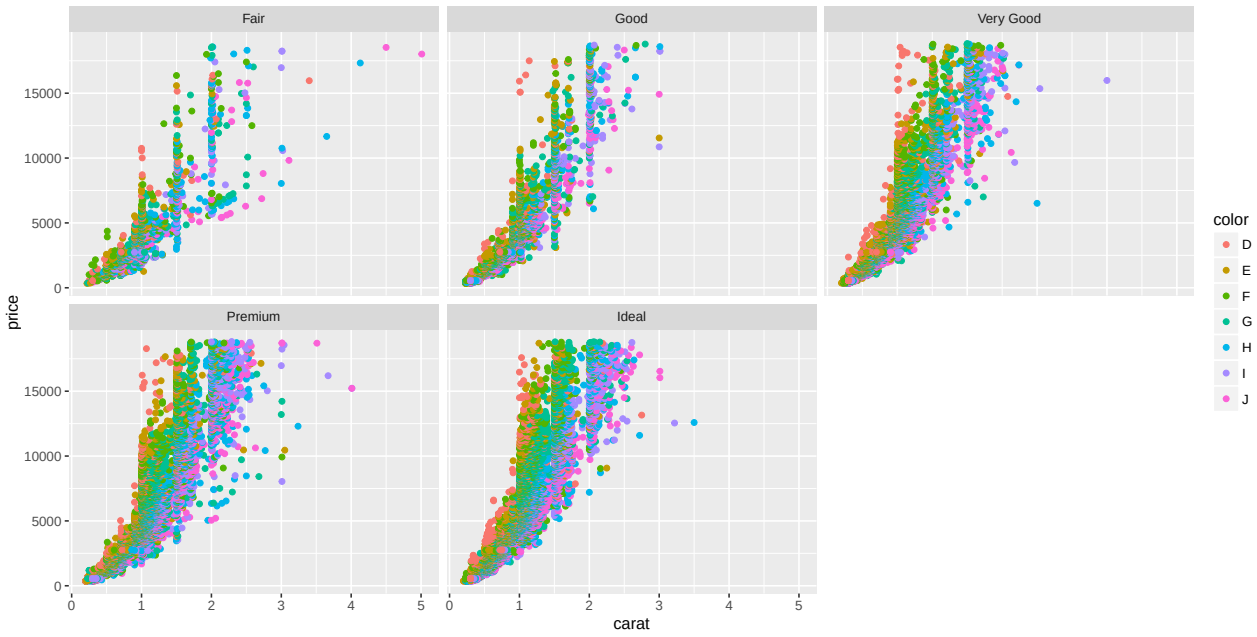
We can change colors by specifying a different color palette. Here's how we can switch to the `cbPalette` we saw last class.

```
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)
```



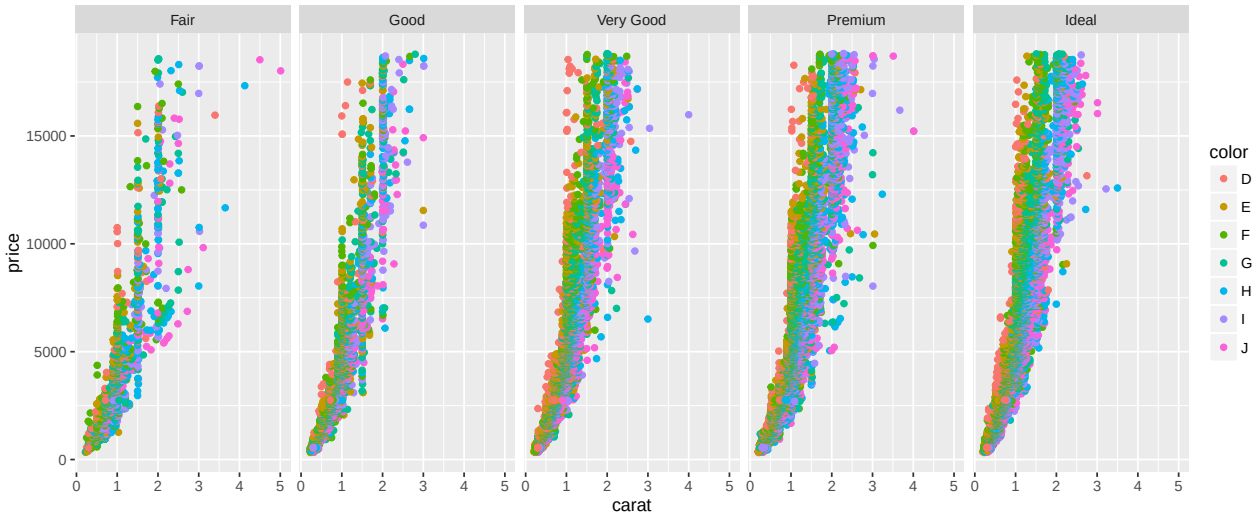
Just like in lattice, we can create plots showing the relationship between variables across different values of a factor. For instance, here's a scatterplot showing how diamond price varies with carat size, conditioned on color. It's created using the `facet_wrap(~ factor1 + factor2 + ... + factorn)`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

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

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

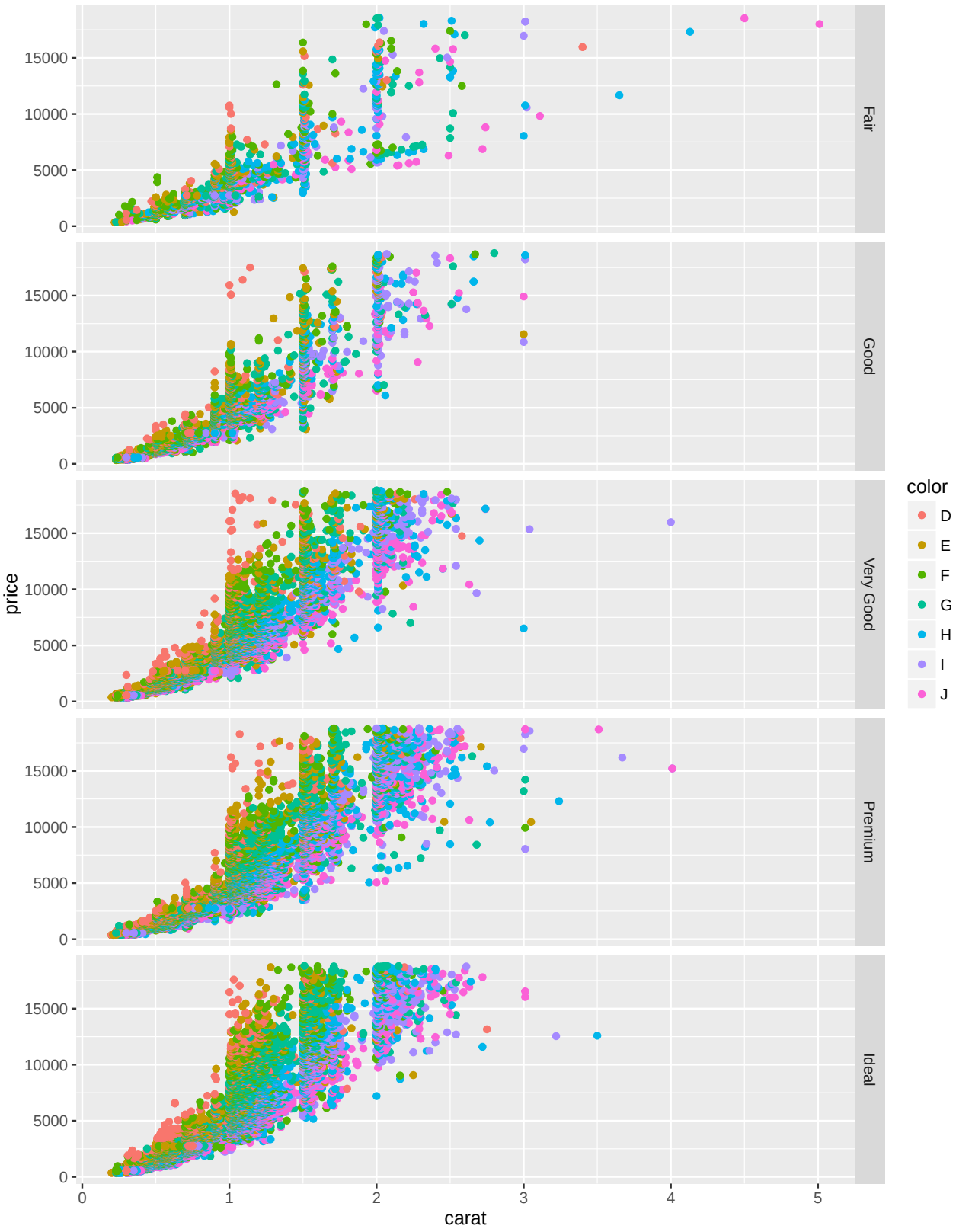


You can also use `facet_grid()` to produce this type of output.

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

A barchart example

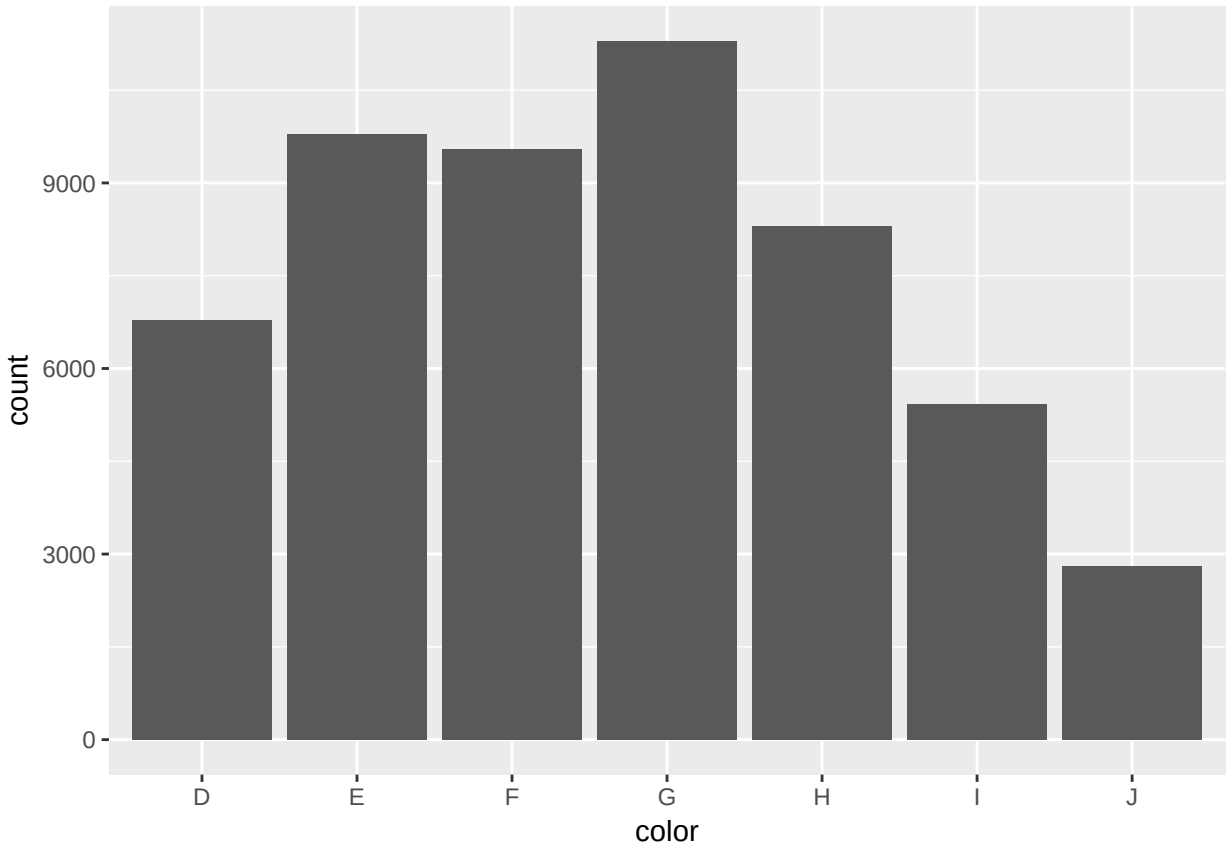
We'll use the `birthwt` data for this example, so let's start by loading an cleaning it. All of this code is borrowed from Lecture 5.

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

# Assign better labels to categorical variables
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")))
            )
```

`geom_bar()` constructs bar charts. Bar charts are useful for visualizing one-number summaries of data. E.g., counts, means, or totals, broken down by levels of a factor. Bar charts are a nice graphical accompaniment to summary tables as generated by `tapply()` or `aggregate()`.

Simple example: Bar chart of diamonds broken down by color

```
p.color <- ggplot(data = diamonds, aes(x = color))
p.color + geom_bar()
```



More complex example: Average birthweight broken down by race and smoking status

We'll start by using the aggregate function to produce a summary of the data in a form that's amenable to plotting with the ggplot2 functions.

```
# Use aggregate to get a table of values
bwt.summary <- aggregate(birthwt.grams ~ race + mother.smokes, data = birthwt, FUN = mean)
# Note that this returns a data frame
class(bwt.summary)
```

```
## [1] "data.frame"
```

Here's what that summary table looks like.

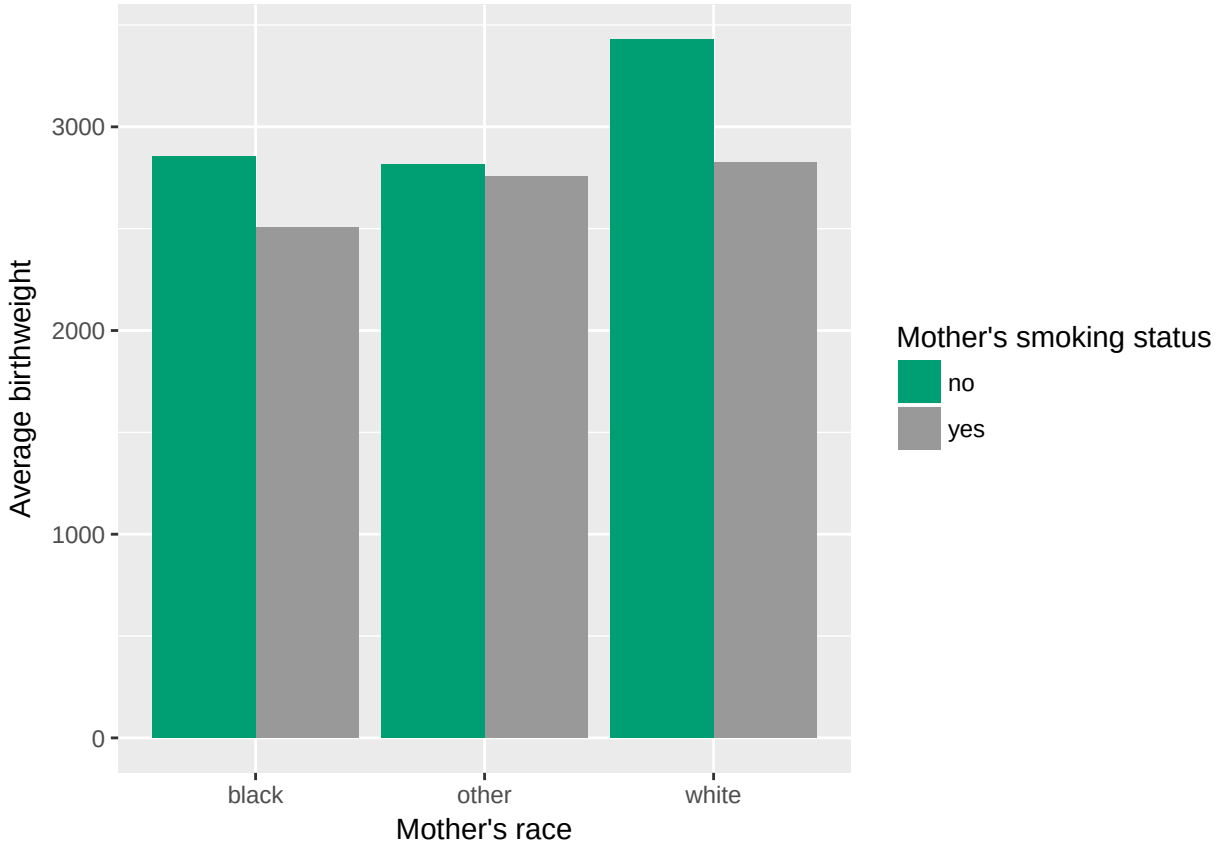
```
bwt.summary
```

```
##    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
```

While the table is informative, it takes a fair bit of effort to glean insight from it. Here's one graphical representation of the same information.

```
# 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)
```

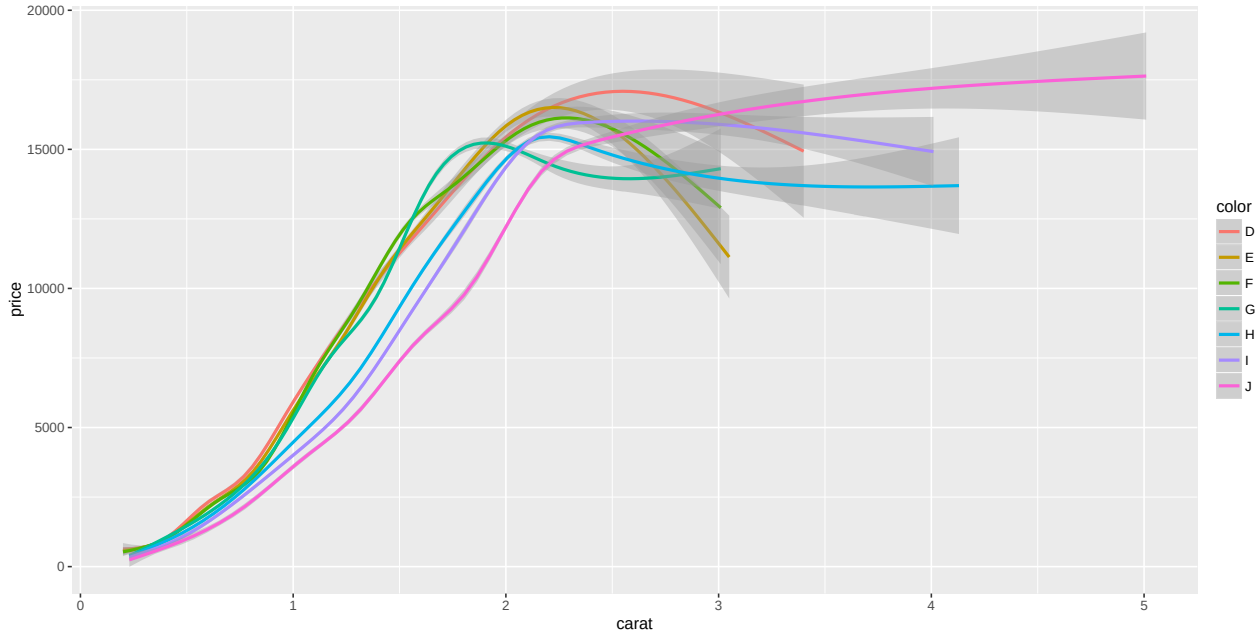


Statistical overlays in ggplot

One of the main advantages of `ggplot` is that it makes it really easy to overlay model fits, confidence bars, etc. We'll get more practice with this next week, when we talk more about how to do statistical inference in R.

For the time being, let's just display a scatterplot smoother.

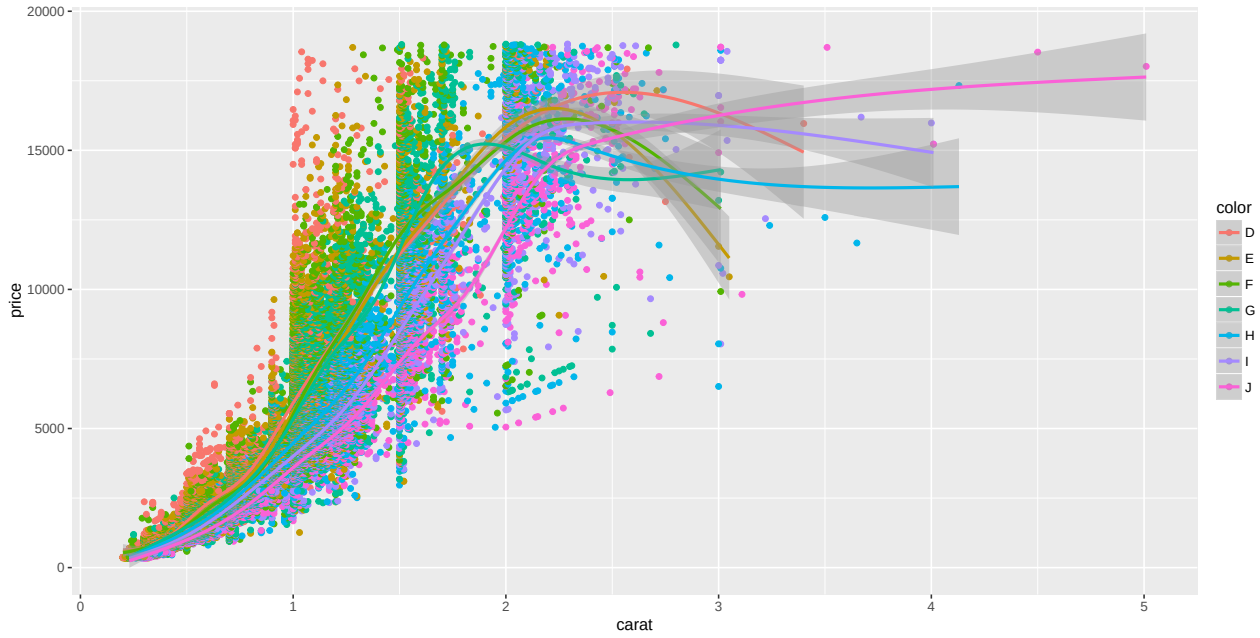
```
diamond.plot + stat_smooth()
```



This shows curves modelling the relationship between diamond size in carats and price for the different diamond colours.

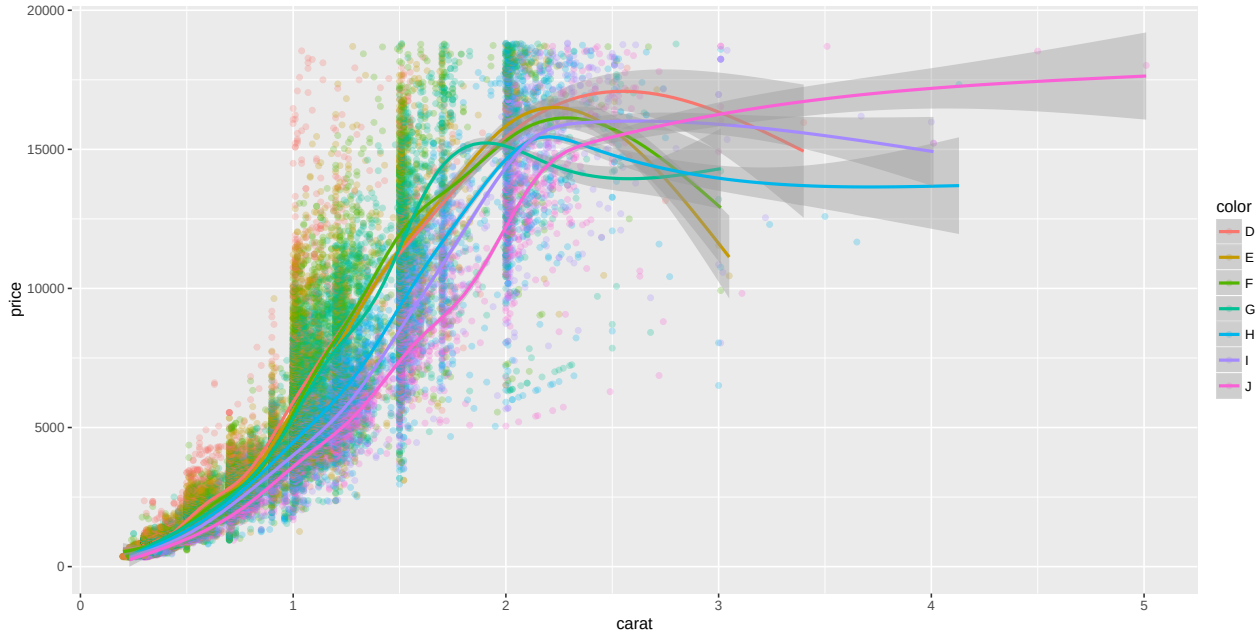
We could use the `geom_point()` command to also display the underlying points, but that would make the curves difficult to see. Let's try it anyway.

```
diamond.plot + geom_point() + stat_smooth()
```



We can make this look better by decreasing the point opacity so that the trend curves aren't so obscured.

```
diamond.plot + geom_point(size = 1.5, alpha = 0.25) + stat_smooth()
```

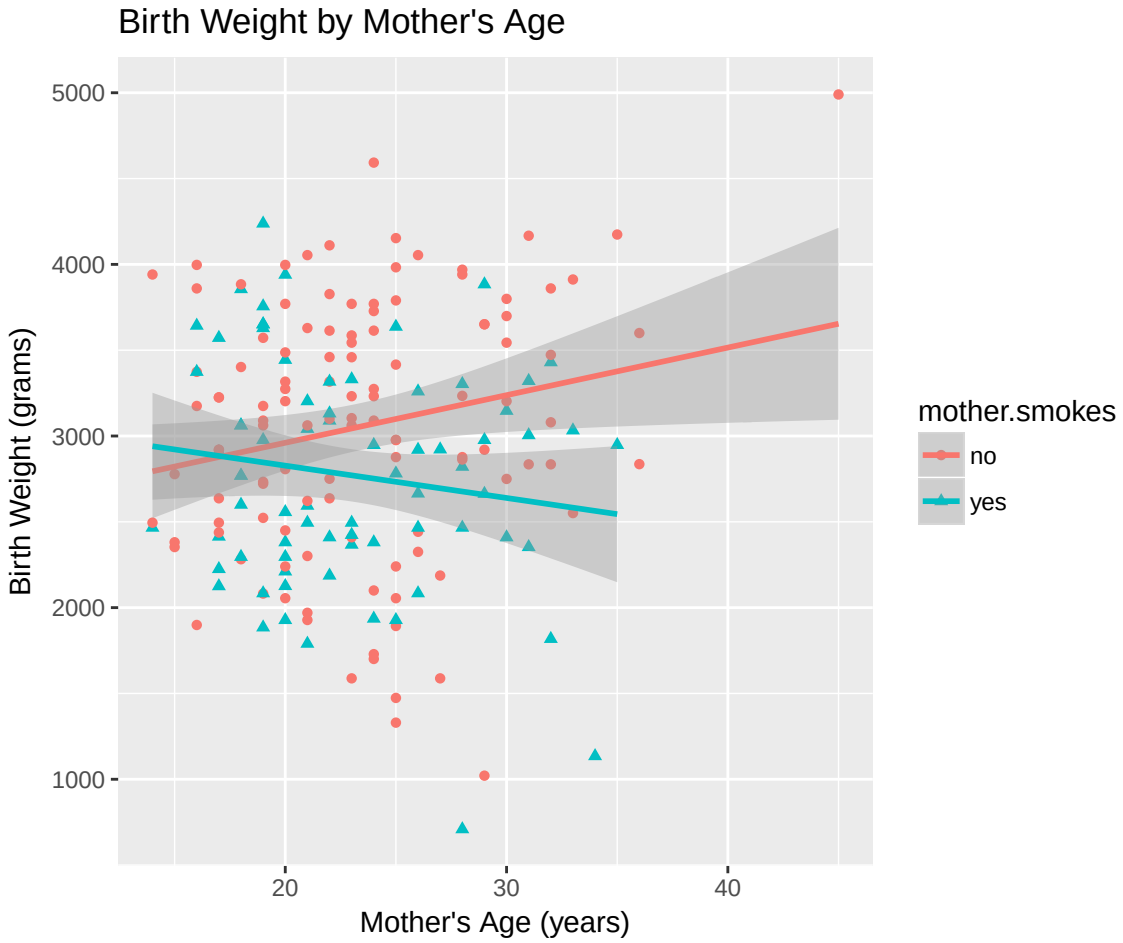


Birthweight regression overlay

In addition to the more flexible smoother in the example above, we can also overlay more structured summaries such as regression curves.

Here's the plot that we saw at the end of Lecture 5, which makes good use of regression line overlays.

```
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
```



Recall that when we calculated correlations between birth weight and mother's age, we got the following results.

```
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
```

This tells us that the association between mother's age and the baby's birthweight seems to depend on the mother's smoking status. Among mothers that smoke, this association is negative (older mothers tend to give birth to lower weight babies), while among mothers that don't smoke, the association is positive.

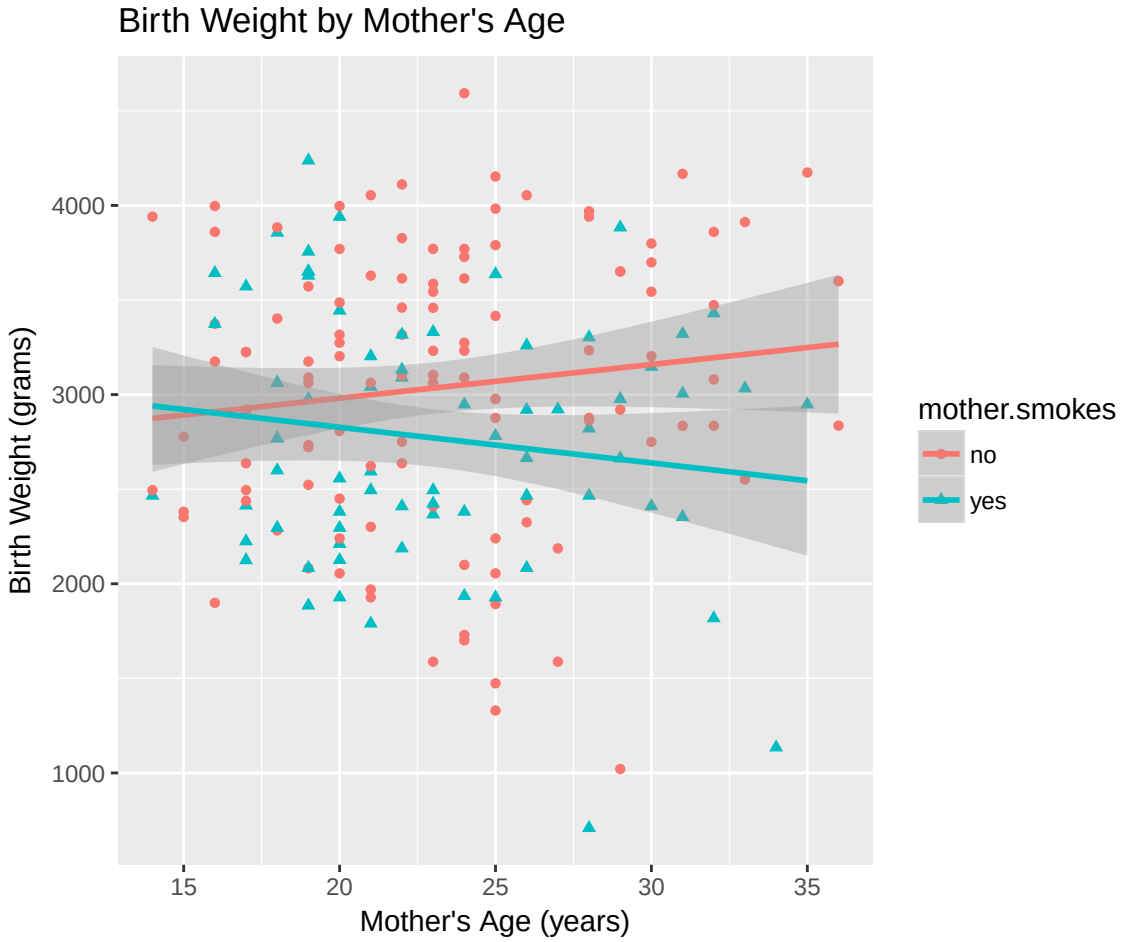
We can read off the same conclusions more simply from the plot. The regression lines capture the association between birthweight and mother's age. The fact that the observed points are very spread out around the regression lines visually conveys the fact that the correlation between birthweight and mother's age is not very large.

Is that an outlier?

The plot shows us something that the correlation calculation cannot: there's a non-smoking mother who was 45 years old when she gave birth, and the combination of her age and the baby's weight (around 5000g) are

well outside the range of all other data points. Let's see what happens when we remove this observation and re-plot.

```
# Get new data set that no longer contains the outlier
birthwt.sub <- subset(birthwt, subset = mother.age < 40)

ggplot(birthwt.sub, 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
```



The general trends appear to be qualitatively the same, but the association in the non-smoking group appears to be far less strong than we were led to believe when we still had the outlying point in our data.

A dot plot/bar chart example.

For this example we're going to use some data based on the 2011 Consumer Expenditure (CE) Survey. This data is stored in a tab-delimited file called `mtbi.txt`.

```
expenses <- read.table("mtbi.txt", sep="\t", header=TRUE)
dim(expenses)
```

```
## [1] 579   4
```

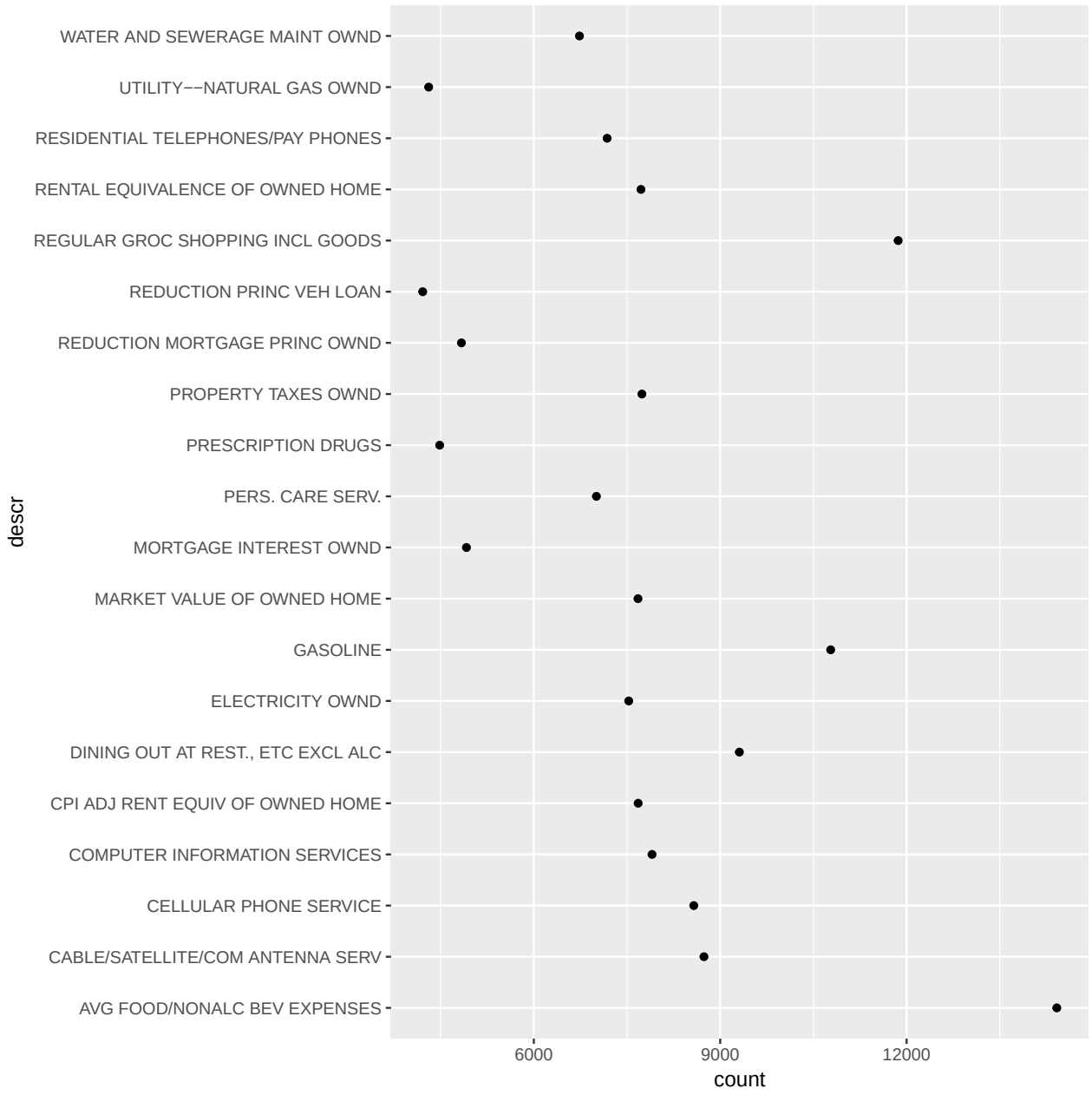
```
head(expenses, 20)
```

```
##        ucc count    avg.cost                            descr
## 480 790240 14418   335.99369      AVG FOOD/NONALC BEV EXPENSES
## 479 790210 11865   470.72061  REGULAR GROC SHOPPING INCL GOODS
## 291 470111 10779   221.60729                          GASOLINE
## 482 790410  9309   192.62423 DINING OUT AT REST., ETC EXCL ALC
## 93  270310  8739    72.29626  CABLE/SATELLITE/COM ANTENNA SERV
## 86  270102  8575    95.50787            CELLULAR PHONE SERVICE
## 460 690114  7903    38.72745     COMPUTER INFORMATION SERVICES
## 28  220211  7740   229.23689               PROPERTY TAXES OWND
## 571 910050  7725   108.06803  RENTAL EQUIVALENCE OF OWNED HOME
## 575 910104  7680  1281.71706  CPI ADJ RENT EQUIV OF OWNED HOME
## 501 800721  7677 19309.76019        MARKET VALUE OF OWNED HOME
## 78  260112  7528   136.53892                  ELECTRICITY OWND
## 85  270101  7180    53.18663 RESIDENTIAL TELEPHONES/PAY PHONES
## 434 650310  7008    39.88927                  PERS. CARE SERV.
## 90  270212  6735    44.15650     WATER AND SEWERAGE MAINT OWND
## 30  220311  4916   612.51098            MORTGAGE INTEREST OWND
## 520 830201  4835  -325.53485     REDUCTION MORTGAGE PRINC OWND
## 348 540000  4485    78.87893               PRESCRIPTION DRUGS
## 82  260212  4308    81.45265         UTILITY--NATURAL GAS OWND
## 528 850100  4212  -331.67379          REDUCTION PRINC VEH LOAN
```

The data is ordered from highest to lowest count (the count is then number of times that an item in the given ucc category was purchased). From `head()` we can see that the 20 most common expenses are things like food/drink, gas, cell phone bills, mortgage payments, etc. This is what we'd expect.

Now, let's create a dot plot showing the counts for each of the top 20 expenses.

```
expense.plot <- ggplot(data = expenses[1:20, ], mapping = aes(y = descr, x = count))
expense.plot + geom_point()
```



This doesn't look very good... What happened?

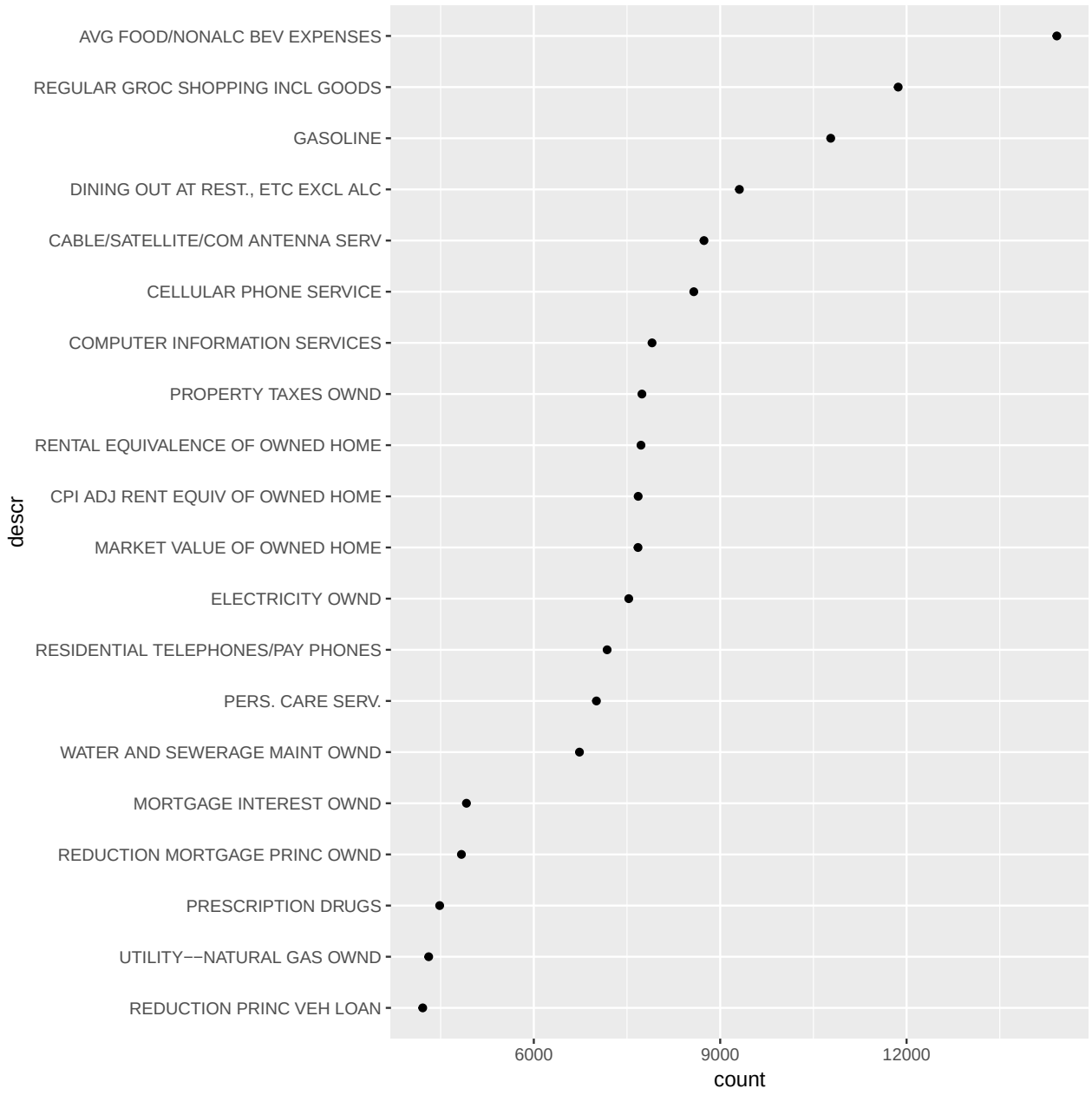
Well, R doesn't know that the `count` variable is important for the purpose of ordering. The default behaviour is to show the plot with factor levels ordered alphabetically. This makes sense in many cases, but not in ours.

To re-order the levels of a factor, we use the `reorder()` function. Here's how we re-order the levels of `descr` based on the value of `count`.

```
expenses <- transform(expenses, descr = reorder(descr, count))
```

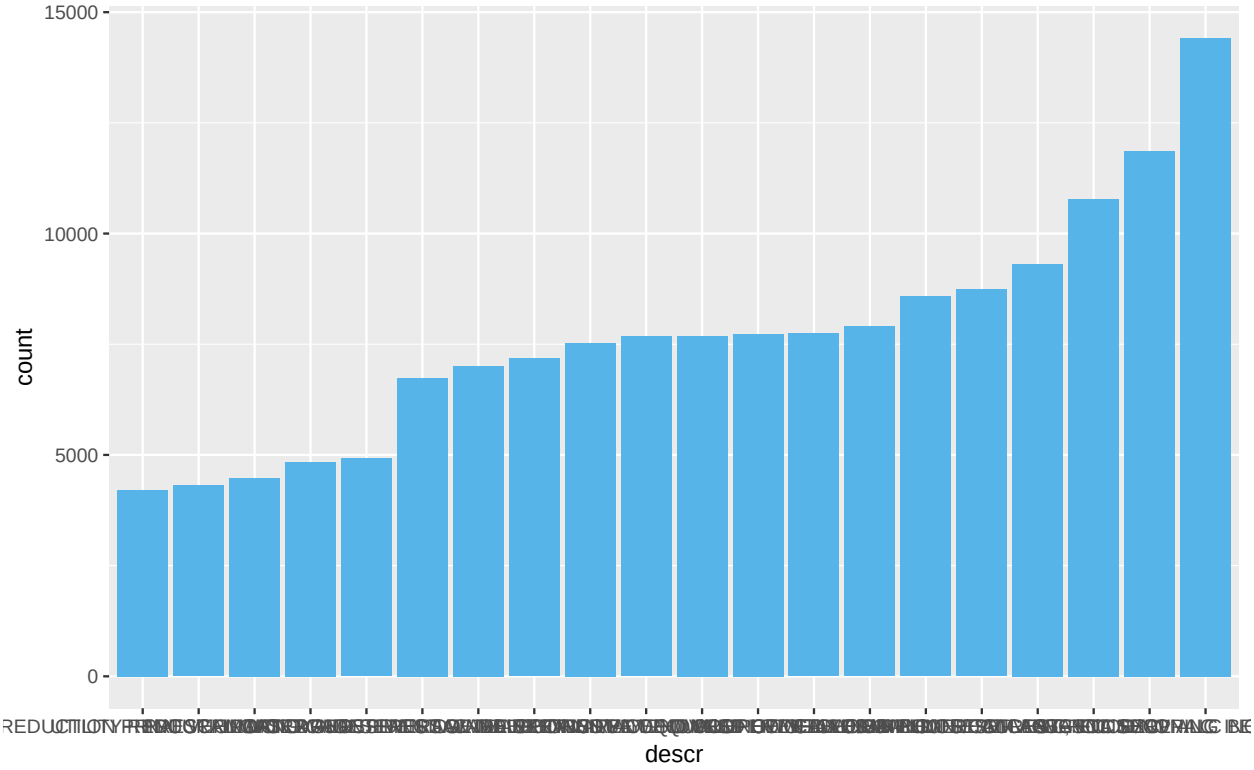
Now let's try that plotting function again...

```
expense.plot <- ggplot(data = expenses[1:20, ], mapping = aes(y = descr, x = count))
expense.plot + geom_point()
```



This kind of data is typically represented with bar charts instead of dot plots. Let's create a bar chart to capture the same information.

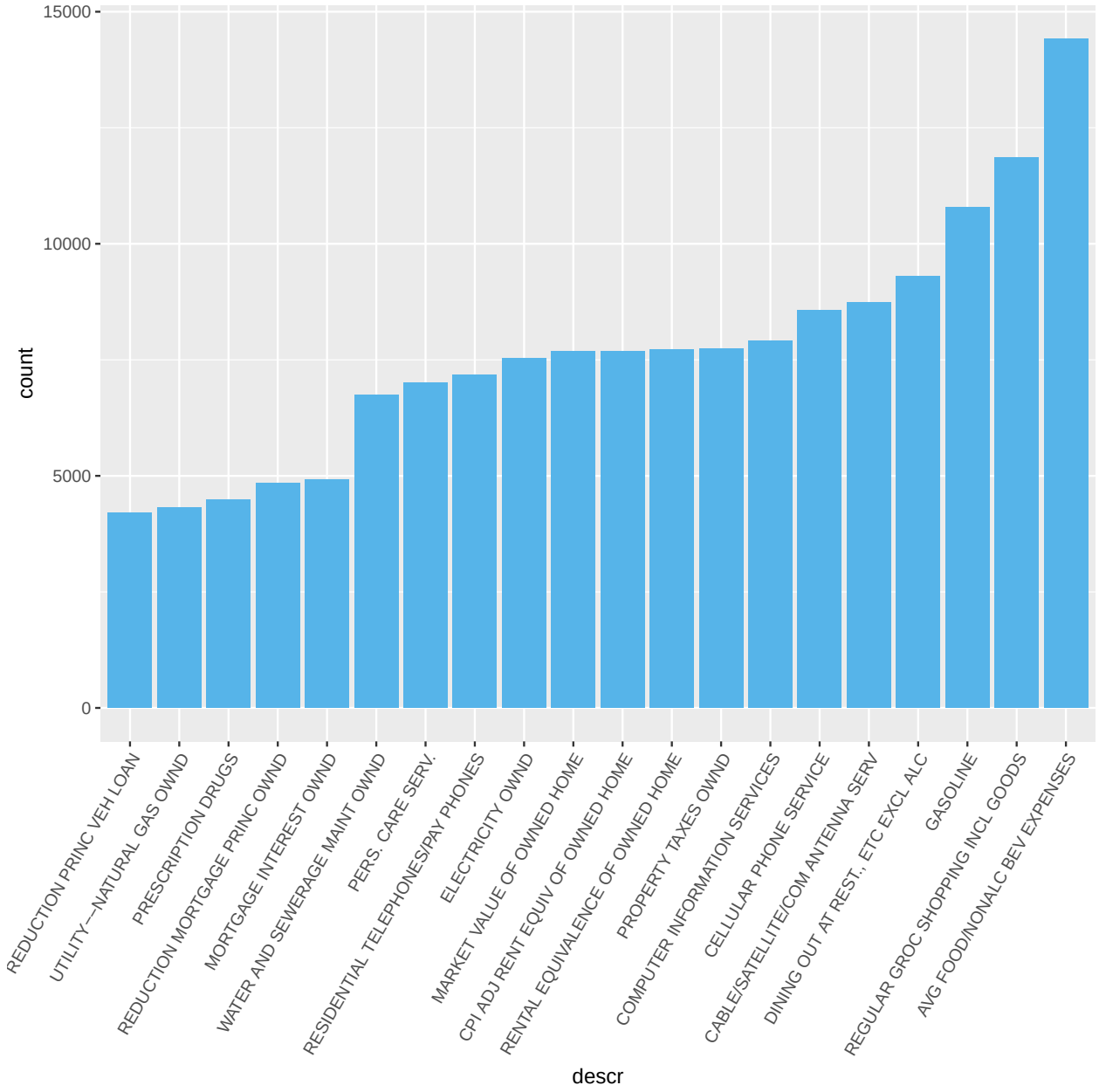
```
barchart.fig = ggplot(data = expenses[1:20, ], mapping = aes(x = descr, y = count))
barchart.fig + geom_bar(stat = "identity", fill = cbPalette[3])
```



There's an issue with this plot: the labels along the x-axis have all blended together and are incomprehensible.

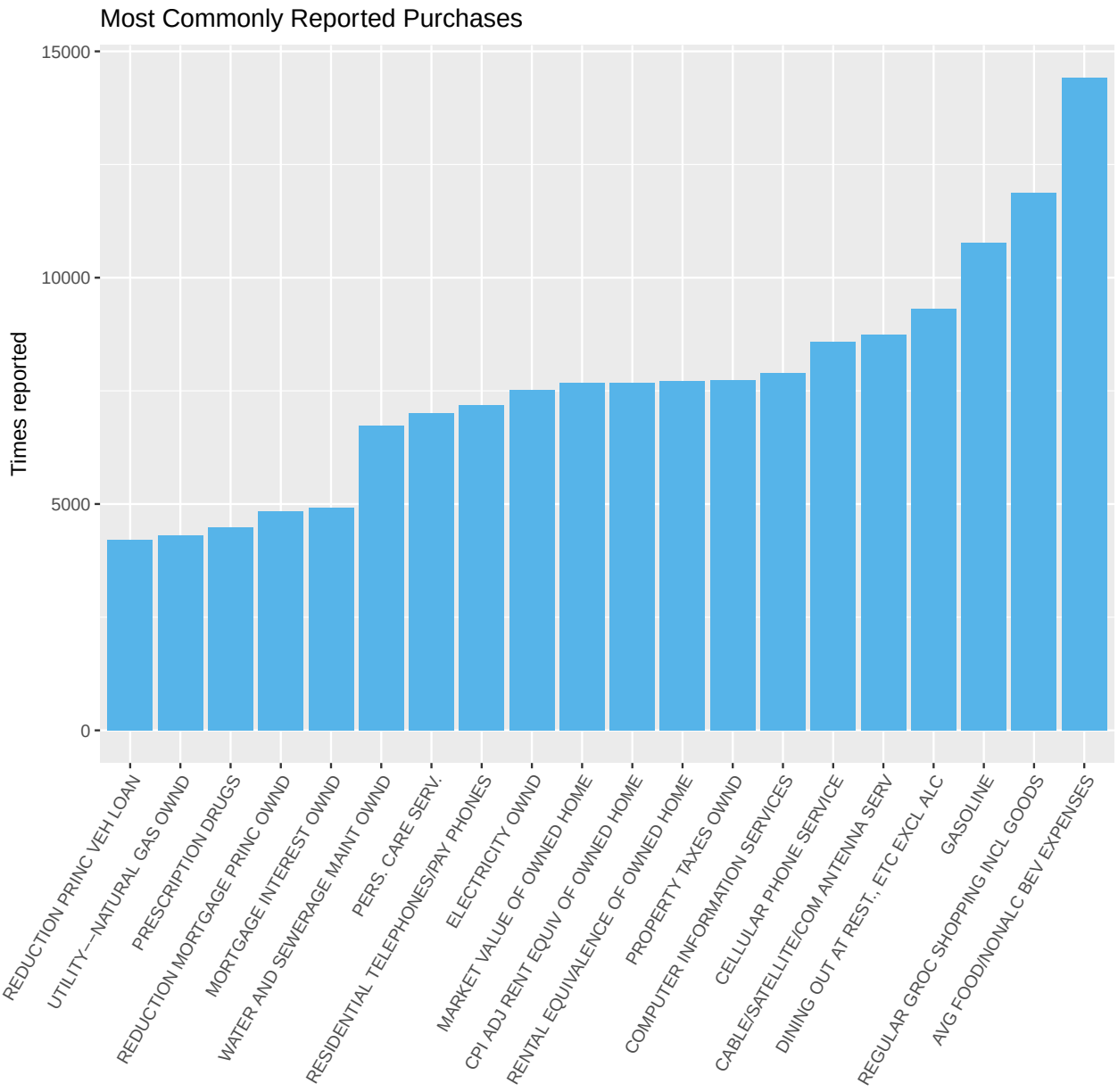
To adjust the text, we can use the `theme` command.

```
barchart.fig + geom_bar(stat = "identity", fill = cbPalette[3]) +
               theme(axis.text.x = element_text(angle = 60, vjust = 1, hjust = 1))
```



We should probably also change the axis labels and title to something more meaningful

```
barchart.fig + geom_bar(stat = "identity", fill = cbPalette[3]) +
               theme(axis.text.x = element_text(angle = 60, vjust = 1, hjust = 1)) +
               labs(x = "", y = "Times reported", title = "Most Commonly Reported Purchases")
```



ggplot2 also does maps

We'll need the `maps` library for this example.

```
library(maps)
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:plyr':
##
##     ozone
```

One of the data sets that comes with R is the `USArrests` data

```
head(USArrests)
```

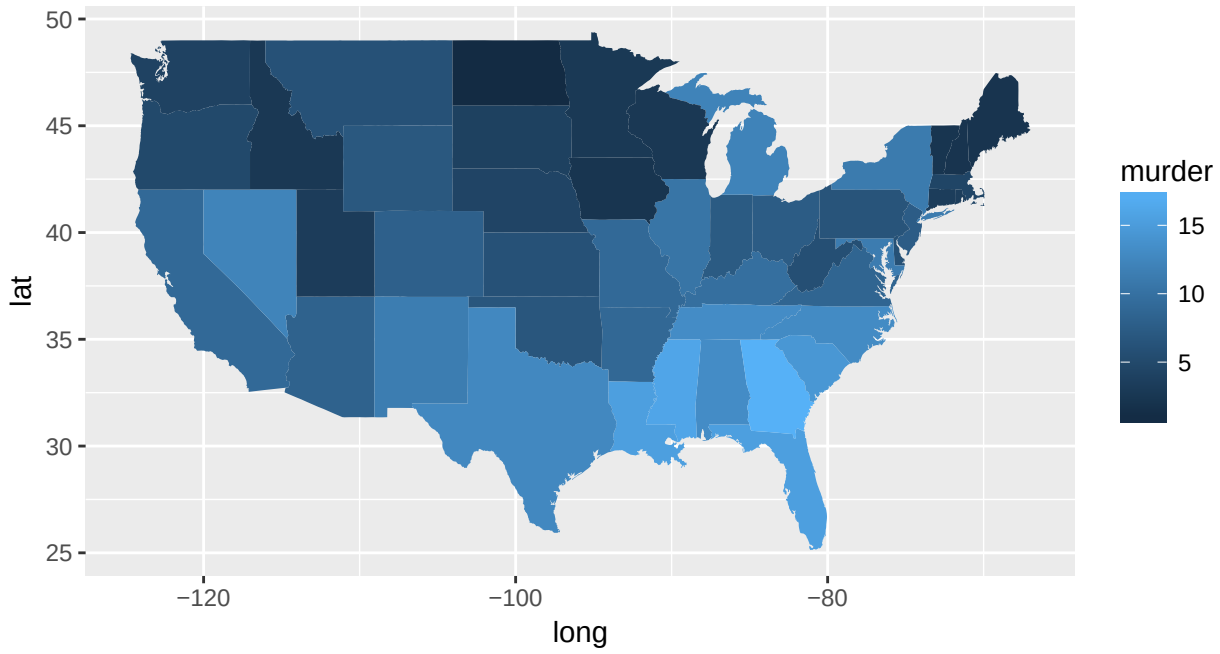
```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Here's how we can get a headmap of Murder rates (per 100,000 population) on a map of the US.

```
# Create data frame for map data (US states)
states <- map_data("state")

# Here's what the states data frame looks like
str(states)
```

```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long     : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat      : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "alabama" "alabama" "alabama" "alabama" ...
##  $ subregion: chr  NA NA NA NA ...
```

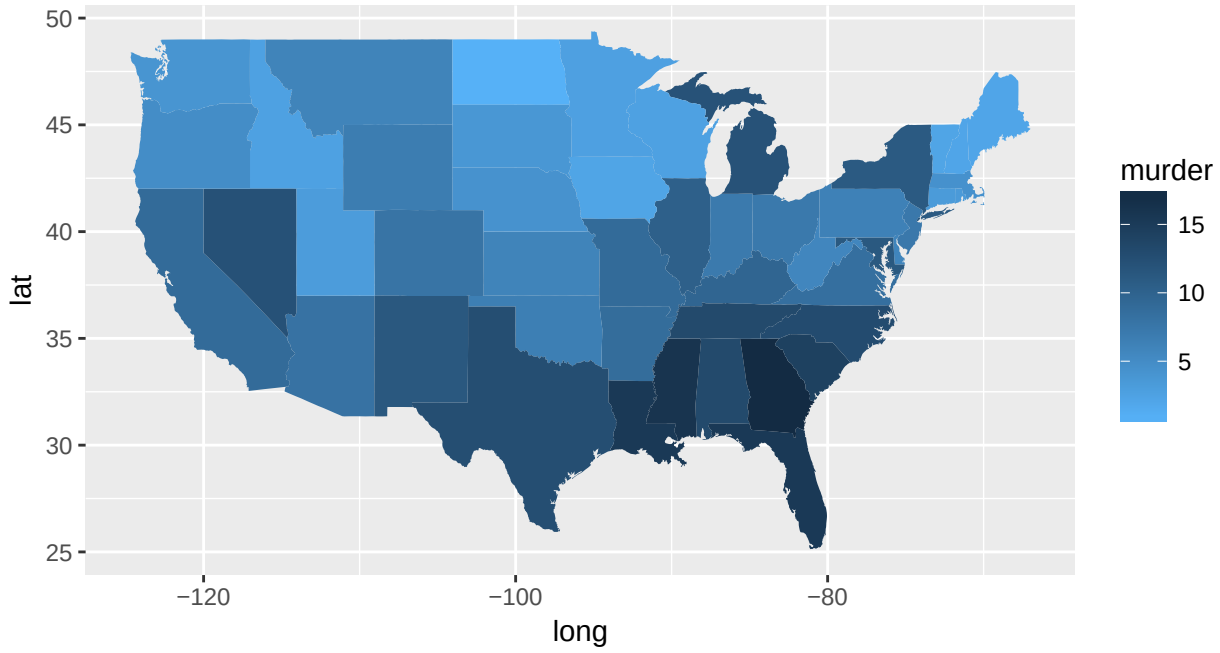
```
# Make a copy of the data frame to manipulate
arrests <- USArrests

# Convert everything to lower case
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))

# Merge the map data with the arrests data based on region
choro <- merge(states, arrests, sort = FALSE, by = "region")
choro <- choro[order(choro$order), ]

# Plot a map, filling in the states based on murder rate
qplot(long, lat, data = choro, group = group, fill = murder,
  geom = "polygon")
```



Essentially what's happening here is that the map data (here called `states`) includes the latitute and longitude coordinates for the boundaries of each state. Specifying `geom = "polygon"` in the `qplot` function results in the states being traced out based on the lat/long coordinates. The `arrests` data provides crime rates for all the states, which allows us to color each polygon (state) based on the murder rate.

It may be counter-intuitive that light blue indicates areas with high murder rates while dark blue indicates areas with low murder rates. We can fix this by specifying a different gradient using the `scale_colour_gradient()` command. Here's an example.

```
qplot(long, lat, data = choro, group = group, fill = murder,
  geom = "polygon") + scale_fill_gradient(low = "#56B1F7", high = "#132B43")
```



All we've done here is swapped the defaults for `low` and `high`.