DSCI354-451 practicum: 06b, Summary Statistics and Visualization

Profs: R. H. French, L. S. Bruckman, P. Leu, K. Davis, S. Cirlos

TAs: W. Oltjen, K. Hernandez, M. Li, M. Li, D. Colvin

11 October, 2022

Contents

```
options("digits" = 5)
options("digits.secs" = 3)
library(learningr)
library(plyr)
library(dplyr)
```

6.2.5.1 setup for r-code chunks

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(reshape2)
```

6.2.5.2 Assignments and Comments on Code?

- We've been reading Hadley's R for Data Science

Now that we've cleaned and have dataframes

- Lets get some summary statistics
- And do some data visualization

6.2.5.3 Useful Tidyverse and Other Packages for Manipulation

6.2.5.3.1 Subsetting and data.table

- http://adv-r.had.co.nz/Subsetting.html
- http://educate-r.org/2014/01/06/FavDataTable/

6.2.5.3.2 plyr and dplyr

- http://cran.r-project.org/web/packages/plyr/index.html
- http://cran.r-project.org/web/packages/dplyr/index.html
- http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html
- http://blog.rstudio.org/2014/01/17/introducing-dplyr/
- http://educate-r.org/2014/01/24/usePlyr/

6.2.5.4 Today's goals

- Be able to calculate a range of summary statistics on numeric data
- Be able to draw standard plots in R's two plotting systems
- Be able to manipulate those plots in simple ways

6.2.5.4.1 Summary Statistics

- We've already come across many of the functions
  - for calculating summary statistics,
  - so this section is partly a recap.

Most are fairly obvious in their naming and their usage; for example,

- `mean` and `median` calculate their respective measures of location.

There isn't a function for the mode,

- but it can be calculated from the results
  - of the table function,
  - which gives counts of each element.

In the following examples, the obama_vs_mccain dataset,

- from the Learning R package,
- contains the fractions of people voting for Obama and McCain
- in the 2008 US presidential elections,
- along with some contextual background information on demographics:

```
data(obama_vs_mccain, package = "learningr")
obama <- obama_vs_mccain$Obama
mean(obama)
```

6.2.5.4.2 `mean`, `median`

```
## [1] 51.289
```

```
median(obama)
```

```
## [1] 51.38
```

6.2.5.4.3 table and cut

- The table function `table` doesn't make a great deal of sense
  - for the obama variable (or many numeric variables)
  - since each value is unique.

```
help(table)
```

```
## Help on topic 'table' was found in the following packages:
##
##   Package               Library
##   base                  /usr/lib/R/library
##   vctrs                 /usr/lib/R/site-library
##
##
## Using the first match ...
```

By combining `table` function with `cut`,

- we can see how many values fall into different bins:

```
help(cut)
```

```
table(cut(obama, seq.int(0, 100, 10)))
```

```
##
##   (0,10]  (10,20]  (20,30]  (30,40]  (40,50]  (50,60]  (60,70]  (70,80]
##        0        0        0        8       16       16        9        1
##  (80,90] (90,100]
##        0        1
```

6.2.5.4.4 `var`, `sd` and `mad`

- `var` and `sd` calculate
  - the variance and standard deviation, respectively.

Slightly less common is the `mad` function

- for calculating the mean absolute deviation:

```
var(obama)
```

```
## [1] 123.07
```

```
sd(obama)
```

```
## [1] 11.094
```

```
mad(obama)
```

```
## [1] 11.49
```

6.2.5.4.5 `min`, `max` and `pmin` and `pmax`, and `range`

- There are several functions for getting the extremes of numeric data.
  - `min` and `max` are the most obvious,
  - giving the smallest and largest values
    - of all their inputs, respectively.

`pmin` and `pmax` (the "parallel" equivalents)

- calculate the smallest and largest values at each point
  - across several vectors of the same length.

Meanwhile, the `range` function

- gives the minimum and maximum in a single function call:

```
min(obama)
```

```
## [1] 32.54
```

```
with(obama_vs_mccain, pmin(Obama, McCain))
```

```
##  [1] 38.74 37.89 44.91 38.86 36.91 44.71 38.22  6.53 36.93 48.10 46.90 26.58
## [13] 35.91 36.74 48.82 44.39 41.55 41.15 39.93 40.38 36.47 35.99 40.89 43.82
## [25] 43.00 49.23 47.11 41.60 42.65 44.52 41.61 41.78 36.03 49.38 44.50 46.80
## [37] 34.35 40.40 44.15 35.06 44.90 44.75 41.79 43.63 34.22 30.45 46.33 40.26
## [49] 42.51 42.31 32.54
```

```
range(obama)
```

```
## [1] 32.54 92.46
```

6.2.5.4.6 `cummin` and `cummax`, `cumsum` and `cumprod`

- Provide cumulative results

`cummin` and `cummax` provide

- the smallest and largest values so far in a vector.

Similarly, `cumsum` and `cumprod`

- provide sums and products of the values to date.

These functions make most sense

- when the input has been ordered in a useful way:

```
cummin(obama)
```

```
##  [1] 38.74 37.89 37.89 37.89 37.89 37.89 37.89 37.89 37.89 37.89 37.89 37.89
## [13] 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91
## [25] 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91 35.91
## [37] 34.35 34.35 34.35 34.35 34.35 34.35 34.35 34.35 34.22 34.22 34.22 34.22
## [49] 34.22 34.22 32.54
```

```
cumsum(obama)
```

```
##  [1]   38.74   76.63  121.54  160.40  221.34  275.00  335.59  428.05  489.96
## [10]  540.87  587.77  659.62  695.53  757.38  807.23  861.16  902.71  943.86
## [19]  983.79 1041.50 1103.42 1165.22 1222.55 1276.61 1319.61 1368.84 1415.95
## [28] 1457.55 1512.70 1566.83 1623.97 1680.88 1743.76 1793.46 1837.96 1889.34
## [37] 1923.69 1980.44 2034.91 2097.77 2142.67 2187.42 2229.21 2272.84 2307.06
## [46] 2374.52 2427.15 2484.49 2527.00 2583.22 2615.76
```

```
cumprod(obama)
```

```
##  [1] 3.8740e+01 1.4679e+03 6.5922e+04 2.5617e+06 1.5611e+08 8.3769e+09
##  [7] 5.0756e+11 4.6929e+13 2.9054e+15 1.4791e+17 6.9370e+18 4.9843e+20
## [13] 1.7899e+22 1.1070e+24 5.5185e+25 2.9761e+27 1.2366e+29 5.0885e+30
## [19] 2.0319e+32 1.1726e+34 7.2606e+35 4.4871e+37 2.5724e+39 1.3907e+41
## [25] 5.9798e+42 2.9439e+44 1.3869e+46 5.7693e+47 3.1818e+49 1.7223e+51
## [31] 9.8412e+52 5.6006e+54 3.5217e+56 1.7503e+58 7.7887e+59 4.0019e+61
## [37] 1.3746e+63 7.8011e+64 4.2492e+66 2.6711e+68 1.1993e+70 5.3669e+71
## [43] 2.2428e+73 9.7855e+74 3.3486e+76 2.2590e+78 1.1889e+80 6.8171e+81
## [49] 2.8979e+83 1.6292e+85 5.3015e+86
```

6.2.5.4.7 `quantile`

- The `quantile` function provides,
  - as you might expect,
  - quantiles (median, min, and max are special cases).

It defaults to

- the median, minimum, maximum, and lower and upper quartiles,
  - and in an impressive feat of overengineering,
  - it gives a choice of nine different calculation algorithms:

```
quantile(obama)
```

```
##     0%    25%    50%    75%   100%
## 32.540 42.755 51.380 57.335 92.460
```

```
quantile(obama, type = 5)    #to reproduce SAS results
```

```
##     0%    25%    50%    75%   100%
## 32.540 42.633 51.380 57.338 92.460
```

```
quantile(obama, c(0.9, 0.95, 0.99))
```

```
##    90%    95%    99%
## 61.920 65.170 82.155
```

6.2.5.4.8 IQR

- IQR wraps quantile
  - to give the interquartile range
  - (the 75th percentile minus the 25th percentile):

```
IQR(obama)
```

```
## [1] 14.58
```

6.2.5.4.9 fivenum

- fivenum provides a faster, greatly simplified alternative to quantile.
  - You only get one algorithm,
  - and only the default quantiles can be calculated.

It has a niche use where speed matters:

```
fivenum(obama)
```

```
## [1] 32.540 42.755 51.380 57.335 92.460
```

6.2.5.4.10 summary

- There are some shortcuts for calculating multiple statistics at once.

You've already met the summary function,

- which accepts vectors or data frames:

```
summary(obama_vs_mccain)
```

```
##         State        Region       Obama          McCain         Turnout
##  Alabama   : 1   IV     : 8   Min.   :32.5   Min.   : 6.53   Min.   :50.8
##  Alaska    : 1   I      : 6   1st Qu.:42.8   1st Qu.:40.39   1st Qu.:61.0
##  Arizona   : 1   III    : 6   Median :51.4   Median :46.80   Median :64.9
##  Arkansas  : 1   V      : 6   Mean   :51.3   Mean   :47.00   Mean   :64.1
##  California: 1   VIII   : 6   3rd Qu.:57.3   3rd Qu.:55.88   3rd Qu.:68.0
##  Colorado  : 1   VI     : 5   Max.   :92.5   Max.   :65.65   Max.   :78.0
##  (Other)   :45   (Other):14                                  NA's   :4
##   Unemployment      Income        Population          Catholic
##  Min.   :3.40   Min.   :19534   Min.   :  563626   Min.   : 6.0
##  1st Qu.:5.05   1st Qu.:23501   1st Qu.: 1702662   1st Qu.:12.0
##  Median :5.90   Median :25203   Median : 4350606   Median :21.0
##  Mean   :6.01   Mean   :26580   Mean   : 6074128   Mean   :21.7
##  3rd Qu.:7.25   3rd Qu.:28978   3rd Qu.: 6656506   3rd Qu.:29.0
##  Max.   :9.40   Max.   :40846   Max.   :37341989   Max.   :46.0
##                                                    NA's   :2
```

```
##    Protestant       Other      Non.religious     Black          Latino
##  Min.   :26.0   Min.   :0.00   Min.   : 5    Min.   : 0.4   Min.   : 1.2
##  1st Qu.:46.0   1st Qu.:2.00   1st Qu.:12    1st Qu.: 3.1   1st Qu.: 4.3
##  Median :54.0   Median :3.00   Median :15    Median : 7.4   Median : 8.2
##  Mean   :53.8   Mean   :3.29   Mean   :16    Mean   :11.1   Mean   :10.3
##  3rd Qu.:62.0   3rd Qu.:4.00   3rd Qu.:19    3rd Qu.:15.2   3rd Qu.:12.1
##  Max.   :80.0   Max.   :8.00   Max.   :34    Max.   :50.7   Max.   :46.3
##  NA's   :2      NA's   :2      NA's   :2
##   Urbanization
##  Min.   :   1.2
##  1st Qu.:  45.8
##  Median : 101.2
##  Mean   : 385.6
##  3rd Qu.: 221.4
##  Max.   :9856.5
##
```

6.2.5.4.11 `cor`, `cancor` and `cov`

- The `cor` function calculates
  - correlations between numeric vectors.

As you would expect, there was an almost perfect negative correlation

- between the fraction of people voting for Obama
- and the fraction of people voting for McCain.

(The slight imperfection is caused by voters for independent candidates.)

The `cancor` function

- (short for "canonical correlation")
- provides extra details.

And the `cov` function

- calculates covariances:

```
with(obama_vs_mccain, cor(Obama, McCain))
```

```
## [1] -0.99812
```

```
with(obama_vs_mccain, cancor(Obama, McCain))
```

```
## $cor
## [1] 0.99812
##
## $xcoef
##          [,1]
## [1,] 0.012748
##
## $ycoef
##           [,1]
## [1,] -0.012867
##
## $xcenter
## [1] 51.289
##
## $ycenter
```

```
## [1] 47
```

```
with(obama_vs_mccain, cov(Obama, McCain))
```

```
## [1] -121.7
```

The `with` function

- lets us evaluate an expression in a data environment

```
?with
```

6.2.5.5 Visualization

6.2.5.5.1 The three plotting systems in R

- Base graphics: used often, but not pretty
- Lattice graphics: an improvement, but cryptic, you can ignore
- ggplot2: grammar of graphics, great plotting system, learn this

So we will focus on base graphics and ggplot2 graphics

Over its lifetime, R has accumulated three different plotting systems.

base graphics are the oldest system,

- having been around as long as R itself.
- base graphs are easy to get started with,
  - but they require a lot of fiddling and magic incantations to polish,
  - and are very hard to extend to new graph types.

We'll ignore the lattice graphics, which is also built on the grid system.

The ggplot2 system,

- also built on top of grid,
- is the most modern of the three plotting systems.

The "gg" stands for "grammar of graphics,''

- which aims to break down graphs into component chunks.

The result is that code for a ggplot

- looks a bit like the English way of articulating what you want in the graph.

http://blog.revolutionanalytics.com/2009/09/ggplot2-and-the-grammar-of-graphics.html

https://ramnathv.github.io/pycon2014-r/visualize/ggplot2.html

See "A Layered Grammar of Graphics" by Hadley Wickham,

```
- in 3-readings in your repo.
- 1000JCGS-wickham_layered-grammar.pdf
```

The three systems are, sadly, mostly incompatible

- there are ways to combine base and grid graphics,
- but they should be considered a last resort.

The good news is that you can do almost everything you want in ggplot2,

- so learning all three systems is mostly overkill.

There are a couple of rare use cases where ggplot2 isn't appropriate:

- it does more calculation than other graphics systems,

- so for quick and dirty plots of very large datasets
  - it can be more convenient to use another system.

Also, many plotting packages are based on one of the other two systems,

- so using those packages requires a little knowledge of base or lattice.

The following examples demonstrate just base and ggplot2;

- if you are pushed for time, then just take note of the ggplot2 parts.
- This gives a taste of some of the possibilities on offer.

Fortunately, there are three excellent and easy to read books

- on graph drawing in R,
  - namely R Graphics, ggplot2, and Lattice,
- by the authors of the grid, ggplot2, and lattice systems, respectively.

So we'll exemplify base and ggplot2 graphics for visualization

6.2.5.6 Scatterplots

6.2.5.6.1 base graphics

- Perhaps the most common of all plots is the scatterplot,
  - used for exploring the relationships between two continuous variables.

The obama_vs_mccain dataset has lots of numeric variables that we can compare,

- but we'll start by asking, "Does voter income affect turnout at the polls?"

The base graphic function to draw a scatterplot is simply plot.

- The best-practice code style these days
  - is to keep all the variables you want for a plot together
  - inside a data frame (or possibly several),
  - rather than having them scattered in individual vectors.

Unfortunately, plot predates this idea,

- so we have to wrap it in a call
- to "with"" to access the columns.

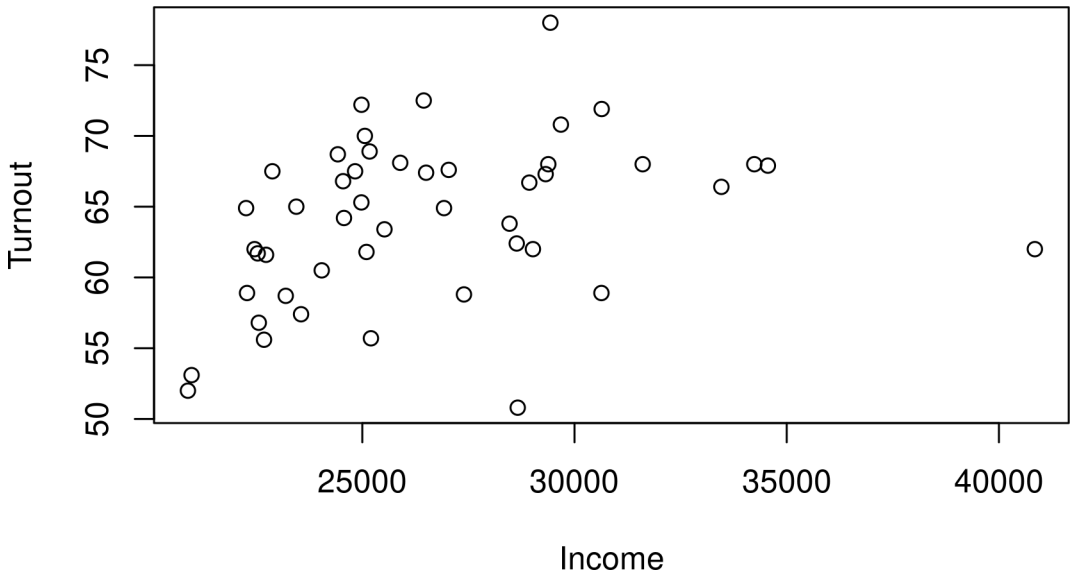
Although plot will simply ignore missing values,

- for tidiness let's remove the rows with missing Turnout values:

```
obama_vs_mccain <-
  obama_vs_mccain[!is.na(obama_vs_mccain$Turnout), ]
```

We can then create a simple scatterplot:

```
with(obama_vs_mccain, plot(Income, Turnout))
```



colors

- Plot has many arguments for customizing the output,
  - some of which are more intuitive than others.
  - col changes the color of the points.

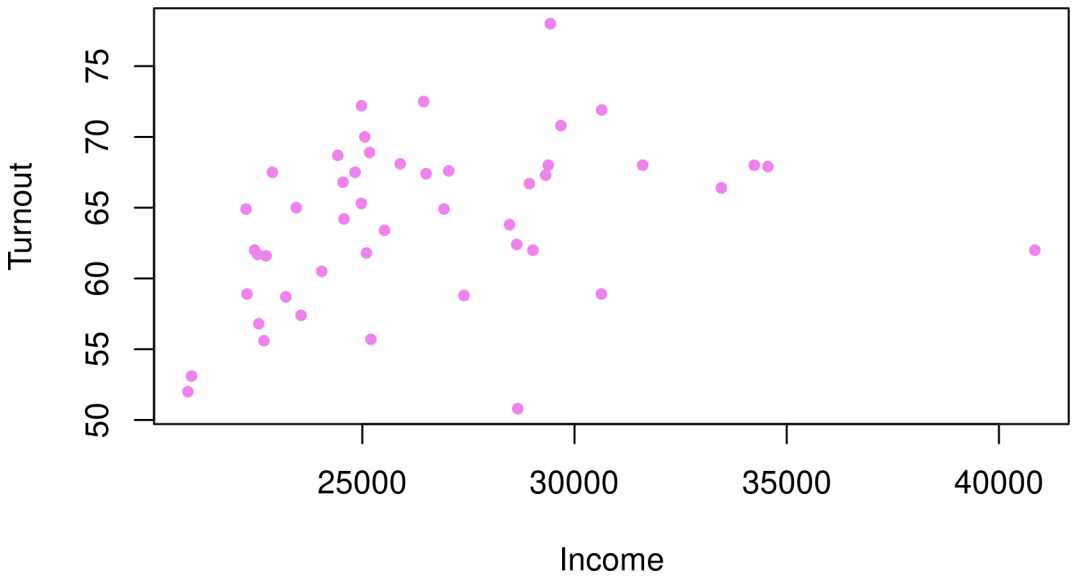
It accepts any of the named colors returned by colors,

- or an HTML-style hex value like "#123456".
- You can change the shape of the points with the pch argument
  - (short for "plot character").

This shows an updated scatterplot,

- changing the point color to violet and
- the point shape to filled-in circles:

```
with(obama_vs_mccain, plot(Income, Turnout, col = "violet", pch = 20))
```

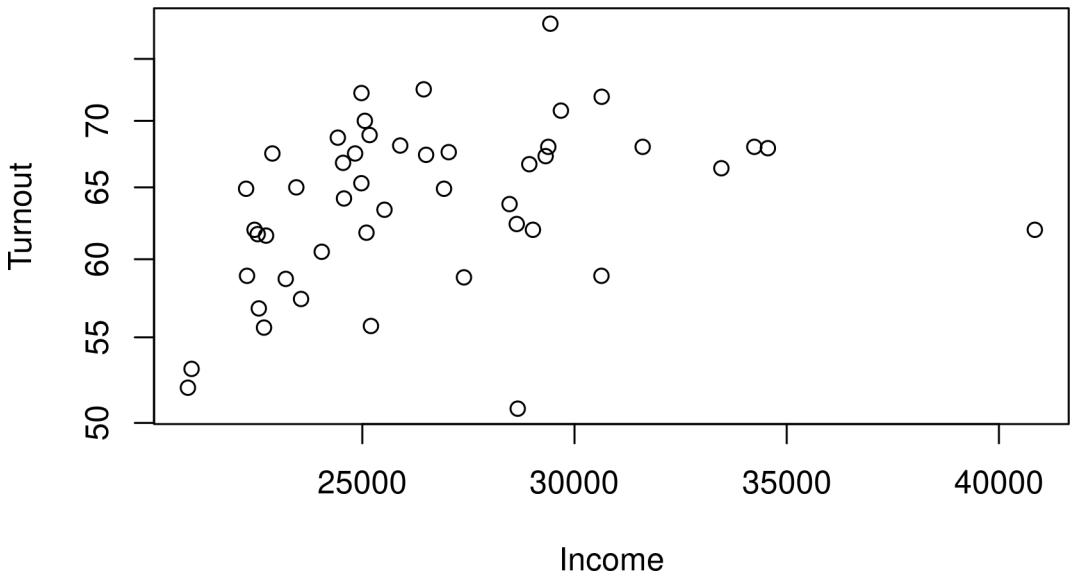


log scales

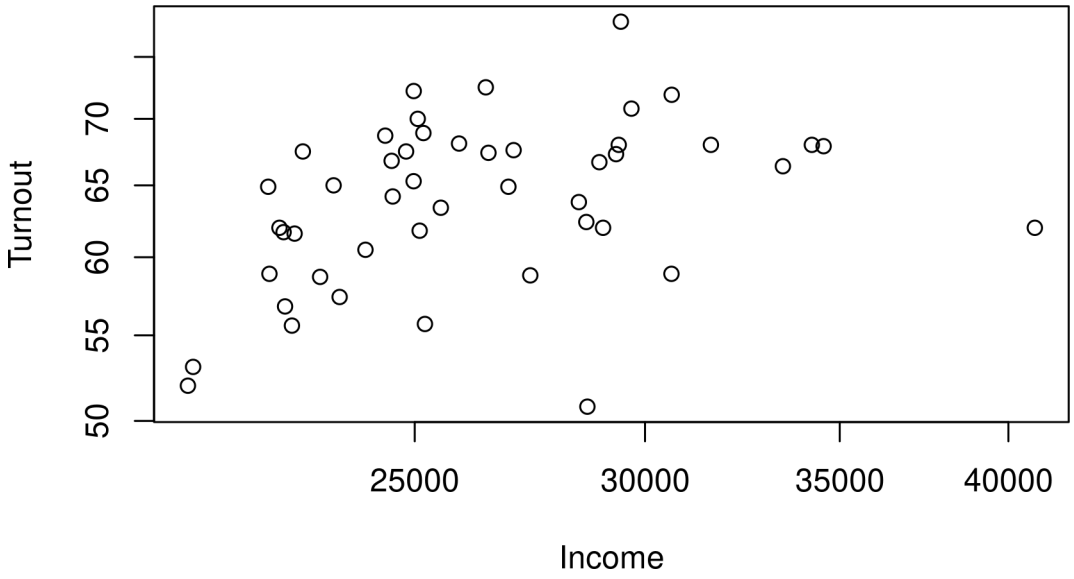
- Log scales are possible by setting the log argument.
  - log = "x" means use a logarithmic x-scale,
  - log = "y" means use a logarithmic y-scale,
  - and log = "xy" makes both scales logarithmic.

This displays some options for log-scaled axes:

```
with(obama_vs_mccain, plot(Income, Turnout, log = "y"))
```




```
with(obama_vs_mccain, plot(Income, Turnout, log = "xy"))
```



We can see that there is a definite positive correlation

- between income and turnout,
- and it's stronger on the log-log scale.

A further question is, "Does the relationship hold across all of the USA?"

- To answer this, we can split the data up
  - into the 10 Standard Federal Regions

  - given in the Region column,
  - and plot each of the subsets in a "matrix" in one figure.

The layout function is used

- to control the layout of the multiple plots in the matrix.

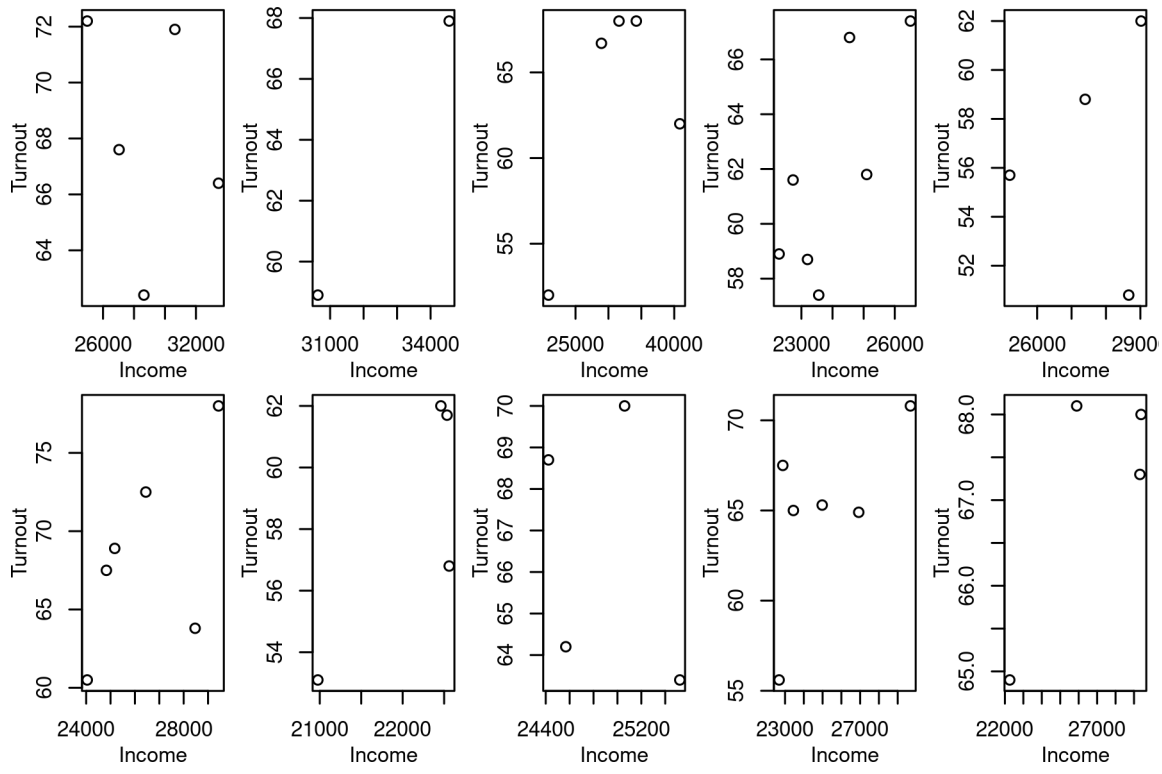
Don't feel obliged to spend a long time

- trying to figure out the meaning of the next code chunk;
- it only serves to show that drawing
  - multiple related plots together in base graphics is possible.

Sadly, the code invariably looks like it fell out of the proverbial ugly tree,

- so this technique should only be used as a last resort.
- Here's the result:

```
par(
  mar = c(3, 3, 0.5, 0.5),
  oma = rep.int(0, 4),
  mgp = c(2, 1, 0)
)
regions <- levels(obama_vs_mccain$Region)
plot_numbers <- seq_along(regions)
layout(matrix(plot_numbers, ncol = 5, byrow = TRUE))
for (region in regions) {
  regional_data <- subset(obama_vs_mccain, Region == region)
  with(regional_data,  plot(Income, Turnout))
}
```



6.2.5.6.2 ggplot2 graphics

- ggplot2 (the "2" is because it took a couple of attempts to get it right)
  - takes many of the good ideas in lattice and builds on them.

So, splitting plots up into panels is easy,

- and sequentially building plots is also possible.

Beyond that, ggplot2 has a few special tricks of its own.

Most importantly, its "grammatical" nature means that

- it consists of small building blocks,
- so it's easier to create brand new plot types,
  - if you feel so inclined.

The syntax is a very different to other plotting code,

- so mentally prepare yourself to look at something new.

Each plot is constructed with a call to the ggplot function,

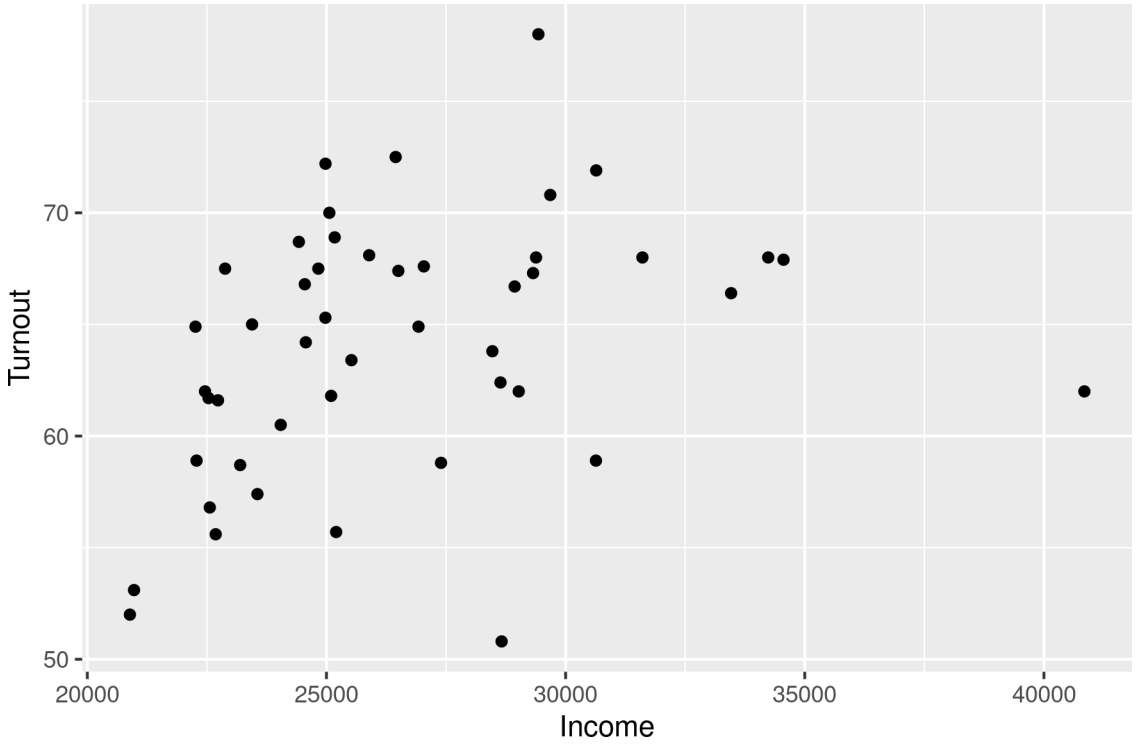
- which takes a data frame as its first argument
- and an aesthetic as its second.

In practice, that means

- passing the columns for the x and y variables
  - to the aes function.
- We then add a geom to tell the plot to display some points.

Here is the result:

```
library(ggplot2)
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point()
```



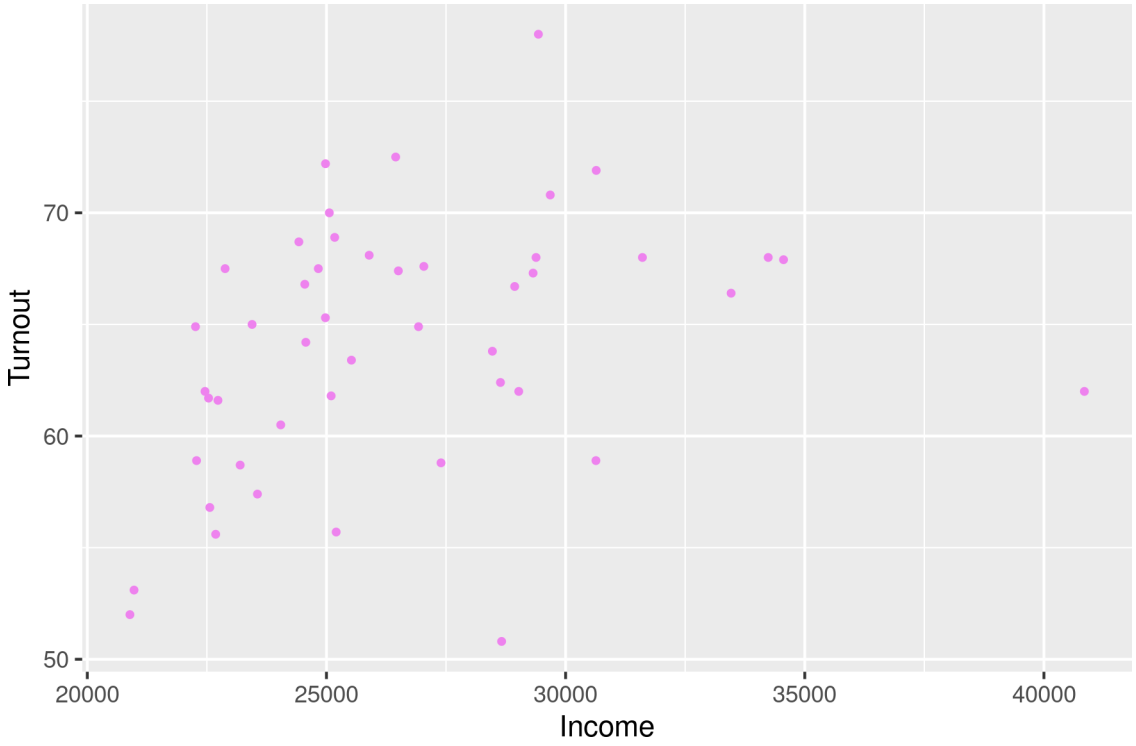
same color/shape commands as base

- ggplot2 recognizes the commands from base
  - for changing the color and shape of the points,
  - but also has its own set of more human-readable names.

In this figure,

- "shape" replaces "pch," and
- color can be specified using either "color" or "colour":

```
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point(color = "violet", shape = 20)
```



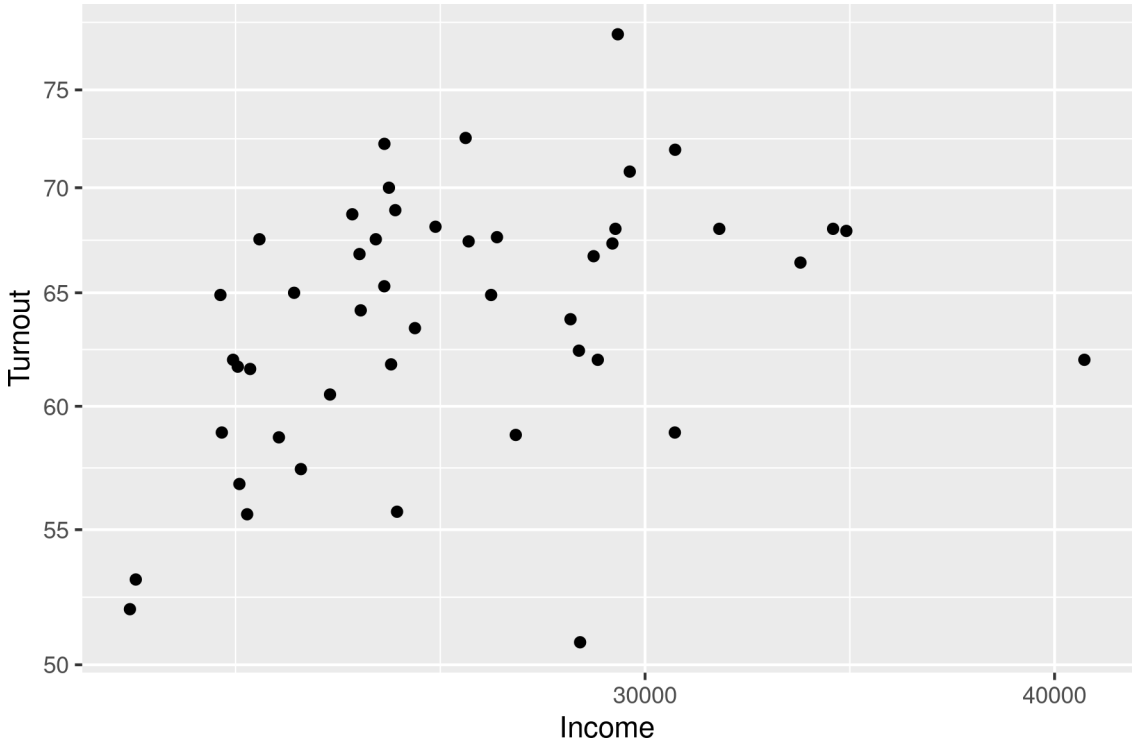
log scales

- To set a log scale, we add a scale for each axis, as seen in the next figure.

The breaks argument specifies the locations of the axis ticks.

It is optional, but used here to replicate the behavior of the base and +lattice examples:

```
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point() +
  scale_x_log10(breaks = seq(2e4, 4e4, 1e4)) +
  scale_y_log10(breaks = seq(50, 75, 5))
```



individual panels using "facets""

- To split the plot into individual panels, we add a facet.

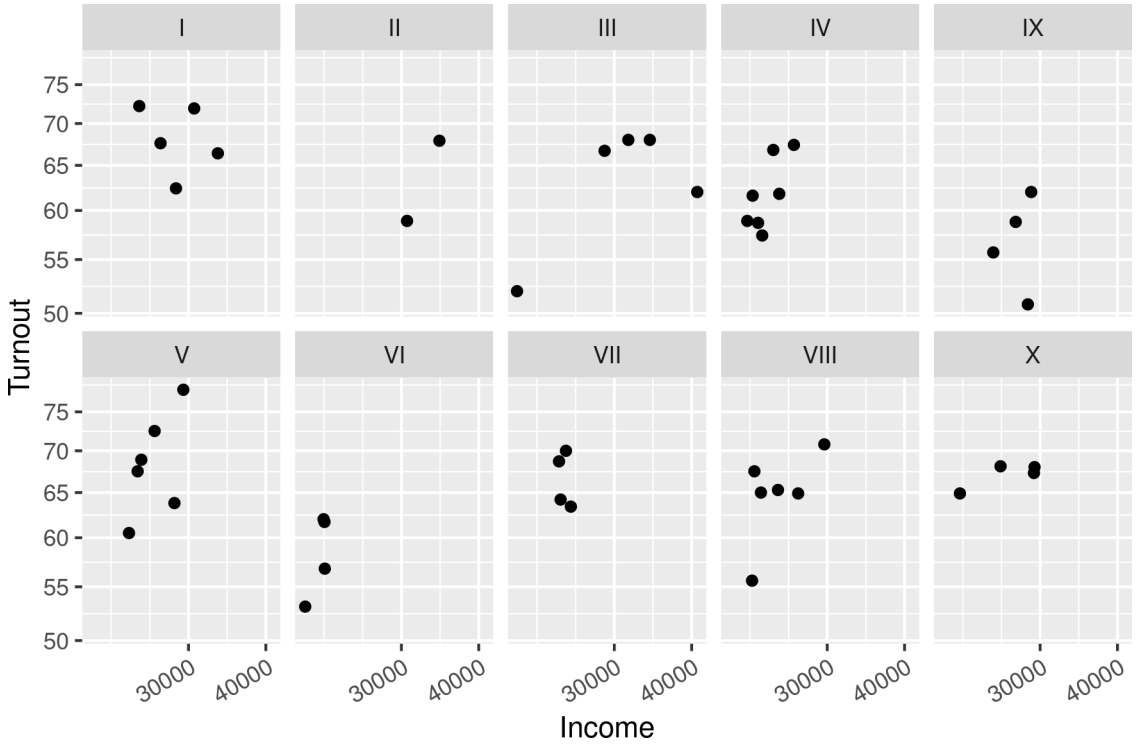
Like the lattice plots, facets take a formula argument.

The next figure demonstrates the facet_wrap function.

For easy reading,

- the x-axis ticks have been rotated by 30 degrees
- and right-justified using the theme function:

```
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point() +
  scale_x_log10(breaks = seq(2e4, 4e4, 1e4)) +
  scale_y_log10(breaks = seq(50, 75, 5)) +
  facet_wrap(~ Region, ncol = 5) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))
```



To split by multiple variables,

- we would specify a formula like ~ var1 + var2 + var3.

For the special case of splitting by exactly two variables,

- facet_grid provides an alternative that
- puts one variable in rows and one in columns.

As with lattice, ggplots can be stored in variables and added to sequentially.

The next example redraws the figure

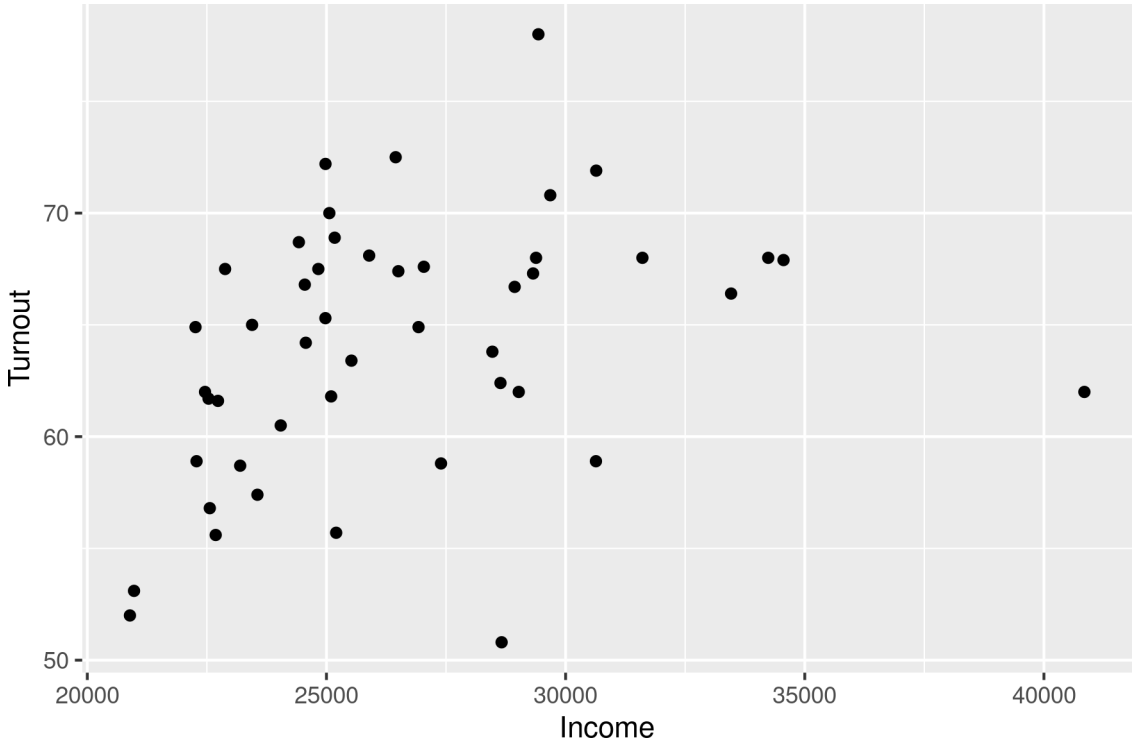
- and stores it as a variable.

As usual, wrapping the expression in parentheses

- makes it auto-print:

This figure shows the output

```
(gg1 <- ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point()
)
```

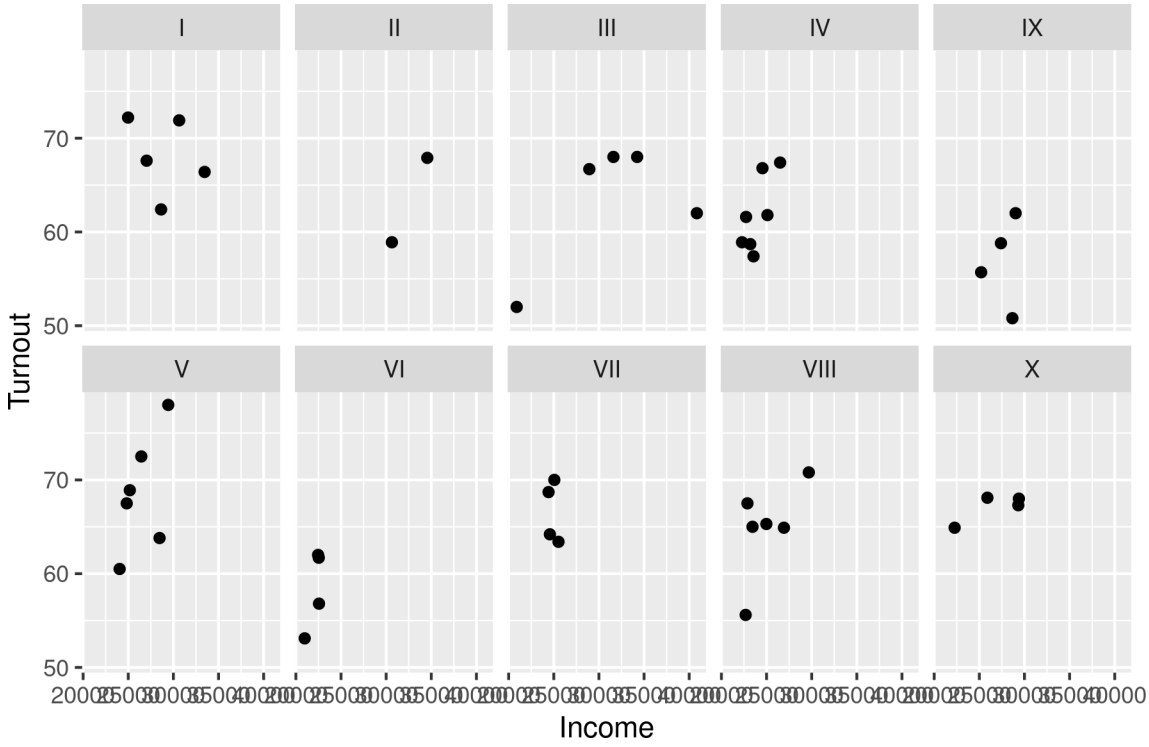


This plot is stored as a variable that is reused in the next figure.

We can then update it as follows,

- with the result shown:

```
  (gg2 <- gg1 +
    facet_wrap(~ Region, ncol = 5)
)
```



This plot reuses a ggplot2 variable from the previous figure.

6.2.5.7 Lineplots

- For exploring how a continuous variable changes over time,
  - a line plot often provides more insight than a scatterplot,
  - since it displays the connections between sequential values.

These next examples examine a year in the life

- of the crab in the crab_tag dataset,
- and how deep in the North Sea it went.

6.2.5.7.1 base graphics

- In base, line plots are created in the same way as scatterplots,
  - except that they take the argument type = "l".

To avoid any dimensional confusion

- we plot the depth as a negative number
- rather than using the absolute values given in the dataset.

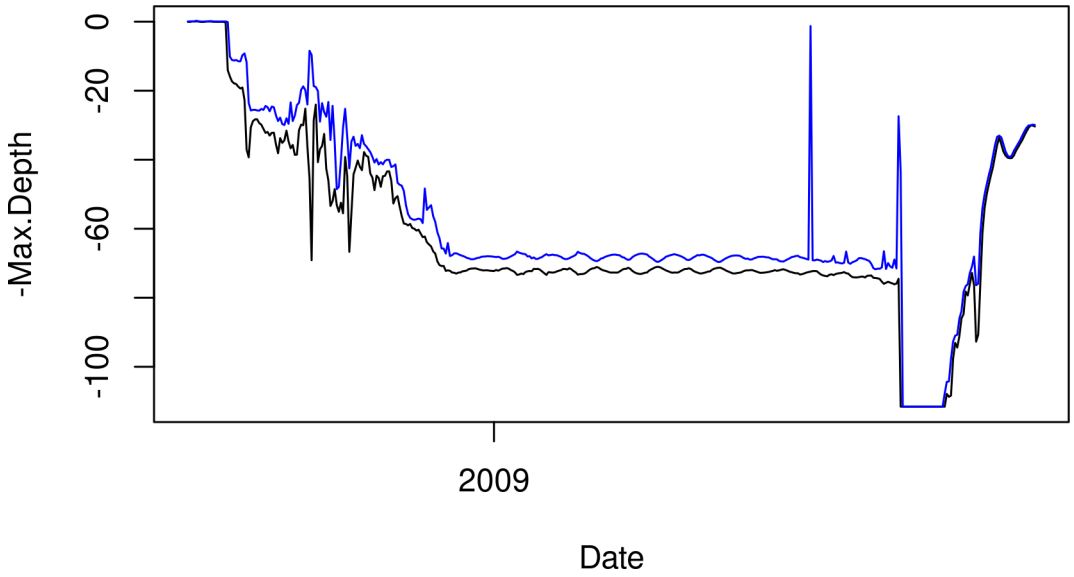
Ranges in the plot

- default to the ranges of the data
  - plus a little bit more;
  - see the x-axis section of the ?par help page for the exact details.

To get a better sense of perspective,

- we'll manually set the y-axis limit
  - to run from the deepest point that the crab went in the sea up to sea level,
  - by passing a ylim argument. The next figure displays the resulting line plot:

```
with(
  crab_tag$daylog,
  plot(Date, -Max.Depth, type = "l", ylim = c(-max(Max.Depth), 0))
)
with(
  crab_tag$daylog,
  lines(Date, -Min.Depth, col = "blue")
)
```



At the moment, this only shows half the story.

- The Max.Depth argument is the deepest point in the sea
  - that the crab reached on a given day.

We also need to add a line for the Min.Depth

- to see the shallowest point on each day.

Additional lines can be drawn on an existing plot using the lines function.

The equivalent for scatterplots is points.

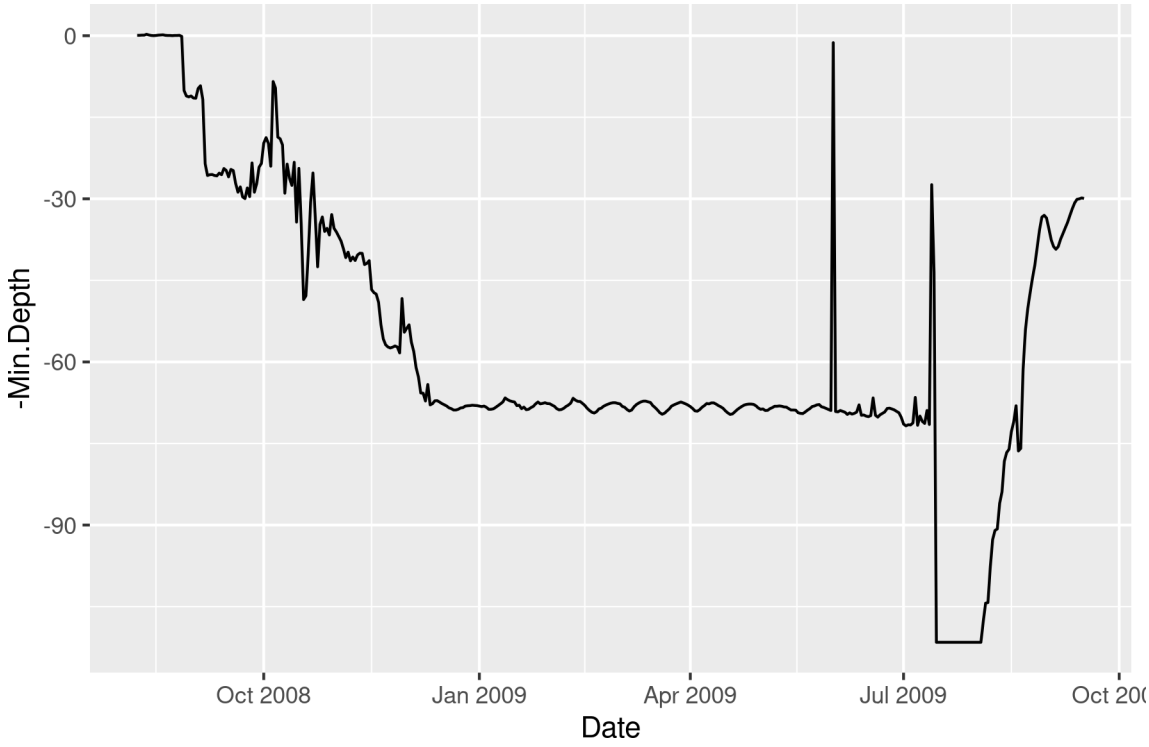
Lets see the figure with the additional line:

6.2.5.7.2 ggplot2 graphics

- In ggplot2, swapping a scatterplot for a line plot
  - is as simple as swapping geom_plot for geom_line

(This shows the result):

```
ggplot(crab_tag$daylog, aes(Date, -Min.Depth)) +
  geom_line()
```



There's a little complication with drawing multiple lines, however.

When you specify aesthetics in the call to ggplot,

- you specify them for every geom.

That is, they are "global" aesthetics for the plot.

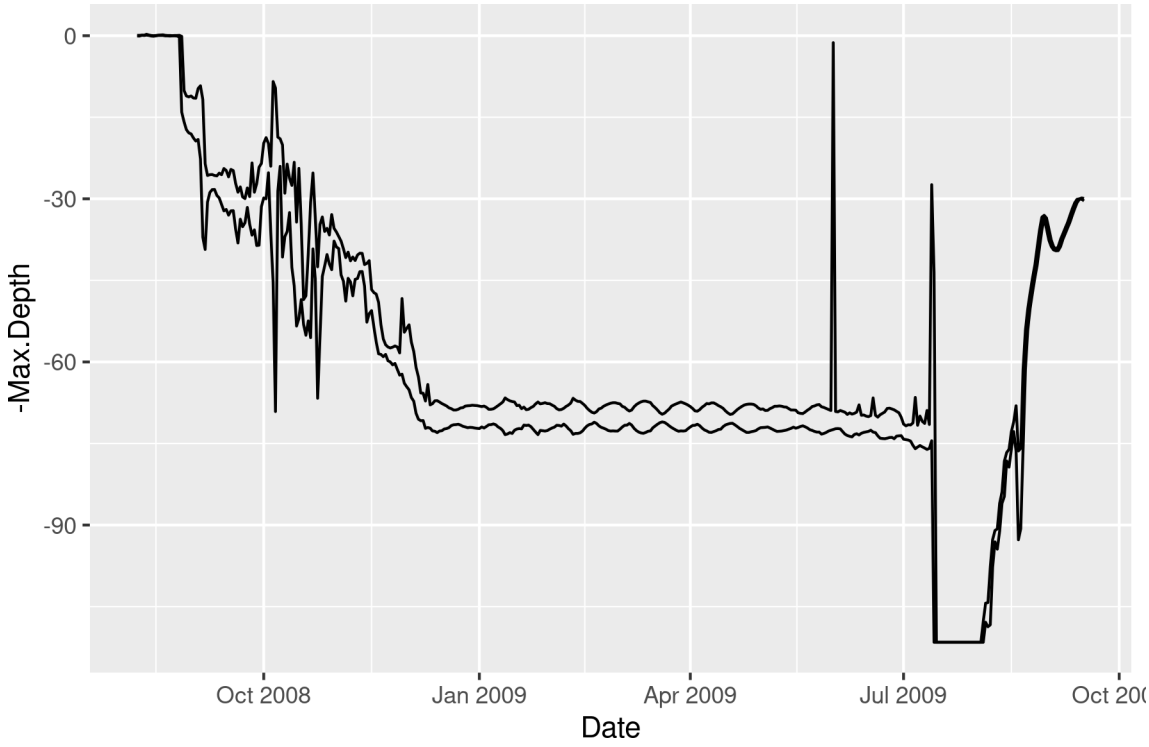
In this case, we want to specify

- the maximum depth in one line
- and the minimum depth in another.

One solution to this is to

- specify the y-aesthetic inside each call to geom_line:

```
ggplot(crab_tag$daylog, aes(Date)) +
  geom_line(aes(y = -Max.Depth)) +
  geom_line(aes(y = -Min.Depth))
```



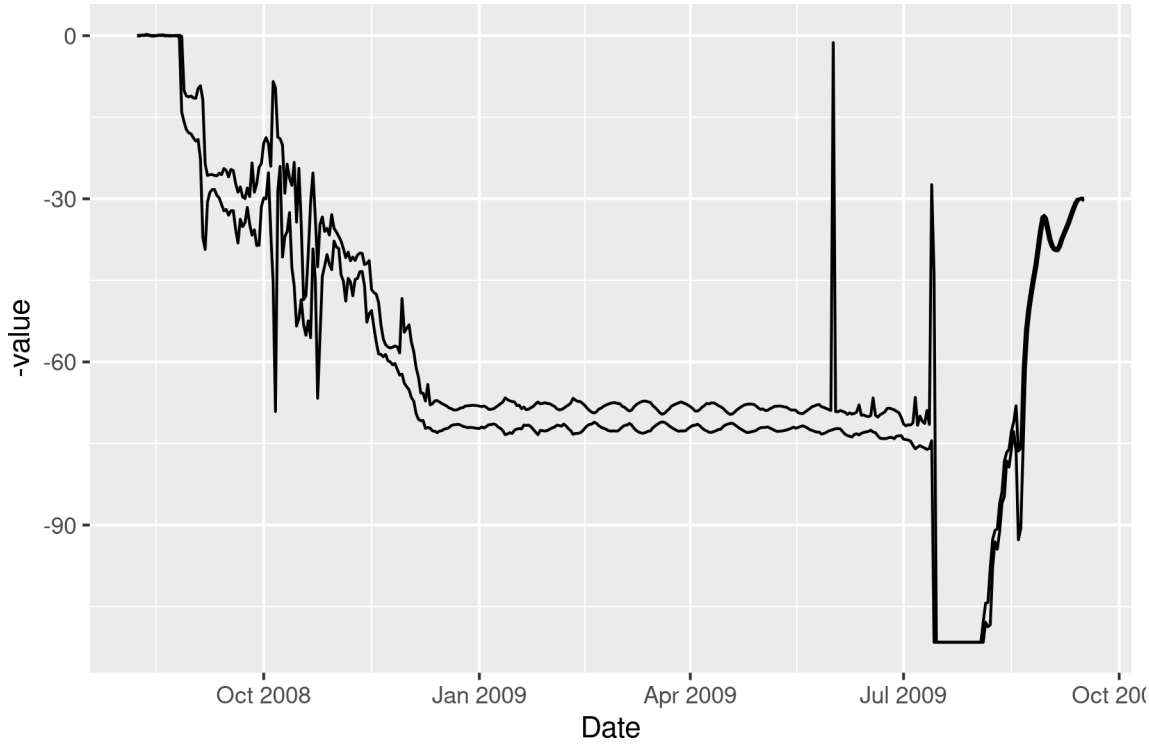
This is a bit clunky, though,

- as we have to call geom_line twice,
- and actually it isn't a very idiomatic solution.

The "proper" ggplot2 way of doing things,

- shown in in the next figure,
- is to melt the data to long form
  - and then group the lines:

```
library(reshape2)
crab_long <- melt(
  crab_tag$daylog,
  id.vars      = "Date",
  measure.vars = c("Min.Depth", "Max.Depth")
)
ggplot(crab_long, aes(Date, -value, group = variable)) +
  geom_line()
```



In this case, where there are only two lines,

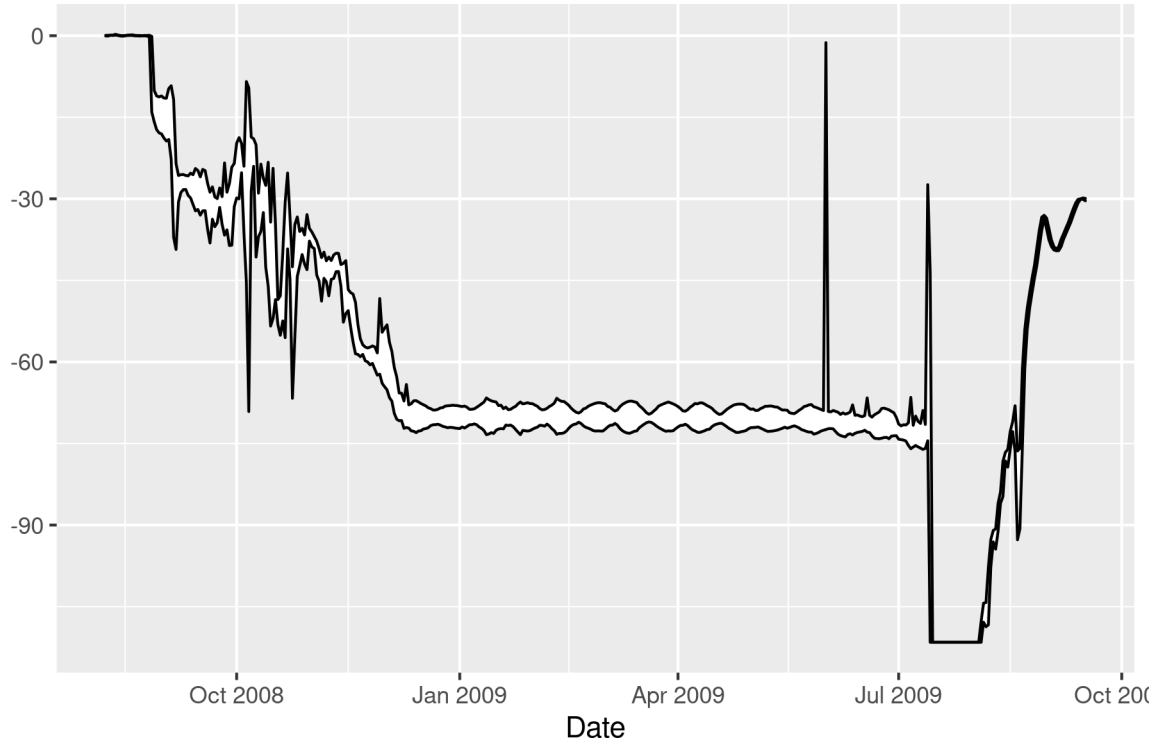
- there is an even better solution
  - that doesn't require any data manipulation.
- geom_ribbon plots two lines, and the contents in between.

For prettiness,

- we pass the color and fill argument to the geom,
- specifying the color of the lines and the bit in between.

This shows the result:

```
ggplot(crab_tag$daylog, aes(Date, ymin = -Min.Depth, ymax = -Max.Depth)) +
  geom_ribbon(color = "black", fill = "white")
```



Whichever system you use to draw the plot,

- the behavior of the crab is clear.

In September it lives in shallow waters for the mating season,

- then it spends a few months migrating into deeper territory.
- Through winter, spring, and summer
  - it happily sits on the North Sea seabed
  - (except for an odd, brief trip to the surface at the start of June
    - dodgy data, or a narrow escape from a fishing boat?),
  - then it apparently falls off a cliff in mid-July,
  - before making its way back to shallow climes
  - for another round of rumpypumpy, at which point it is caught.

6.2.5.8 Histograms

- If you want to explore the distribution of a continuous variable,
  - histograms are the obvious choice.

Dataviz purists often note that kernel density plots

- generally give a "better" representation of the underlying distribution. -The downside is that every time you show them to a non-statistician,
  - you have to spend 15 minutes explaining what a kernel density plot is.

6.2.5.8.1 base graphics

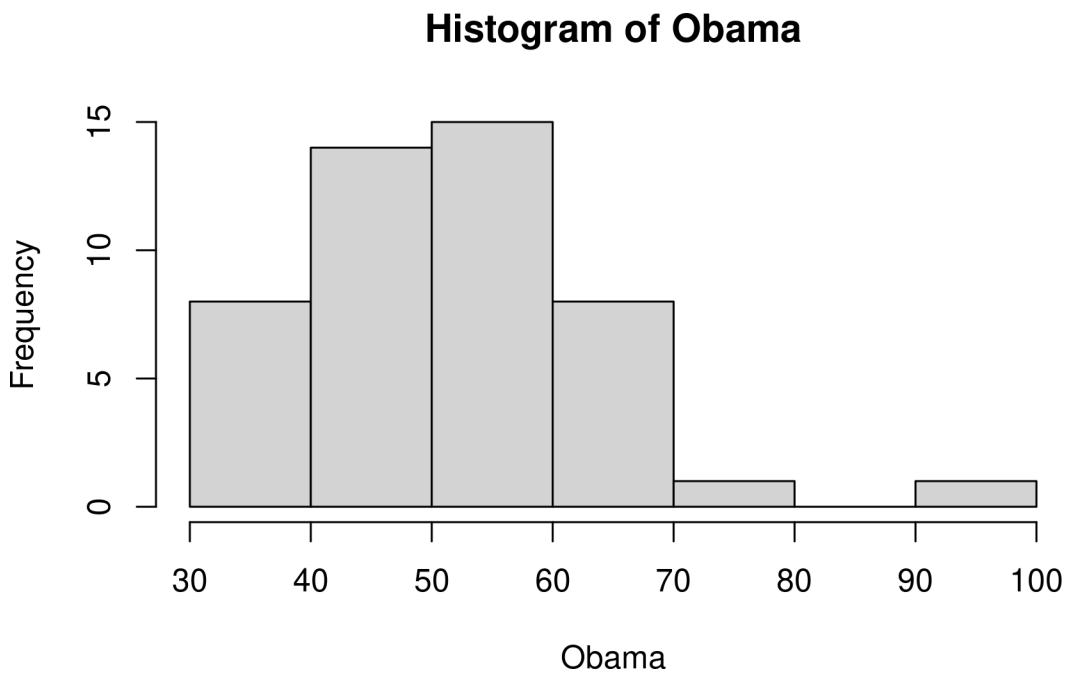
- For the next examples we'll return to the obama_vs_mccain dataset,
  - this time looking at the distribution of the percentage of votes for Obama.

In base, the `hist` function draws a histogram.

Like `plot`, it doesn't have a data argument,

- so we have to wrap it inside a call to with:

```
with(obama_vs_mccain, hist(Obama))
```



The number of breaks is calculated by default by Sturges's algorithm.

It is good practice to experiment with the width of bins

- in order to get a more complete understanding of the distribution.

This can be done in a variety of ways:

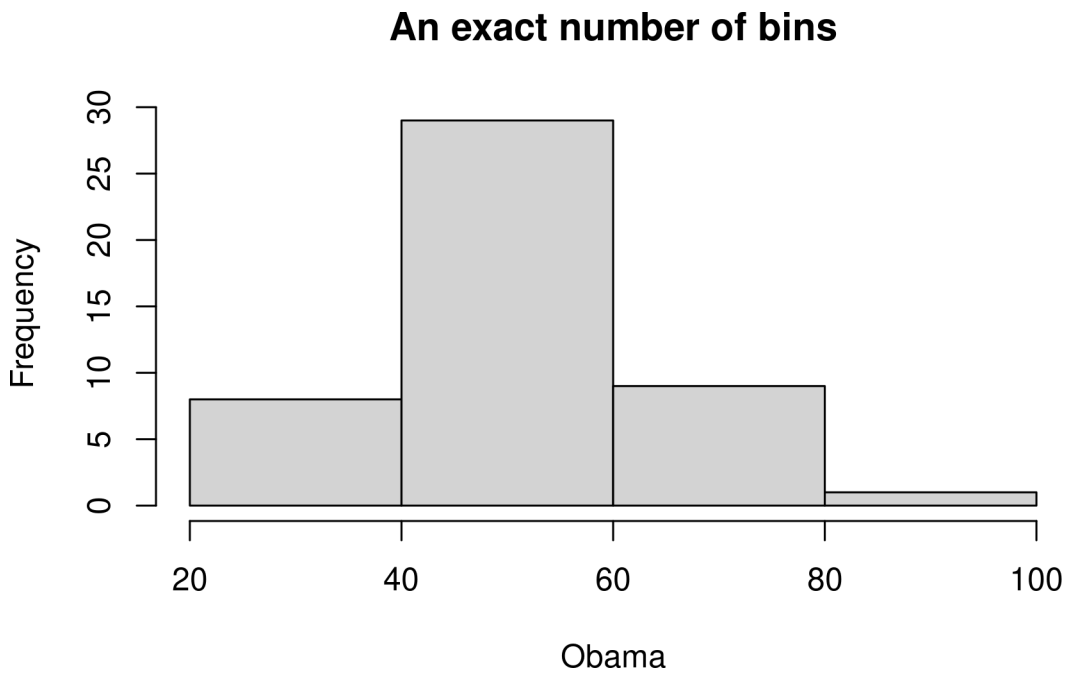
- you can pass hist a single number to specify the number of bins,
- or a vector of bin edges,
- or the name of a different algorithm for calculating the number of bins
  - ("scott" and "fd" are currently supported on top of the default of "sturges"),
- or a function that calculates one of the first two options.

It's really flexible. In the following examples,

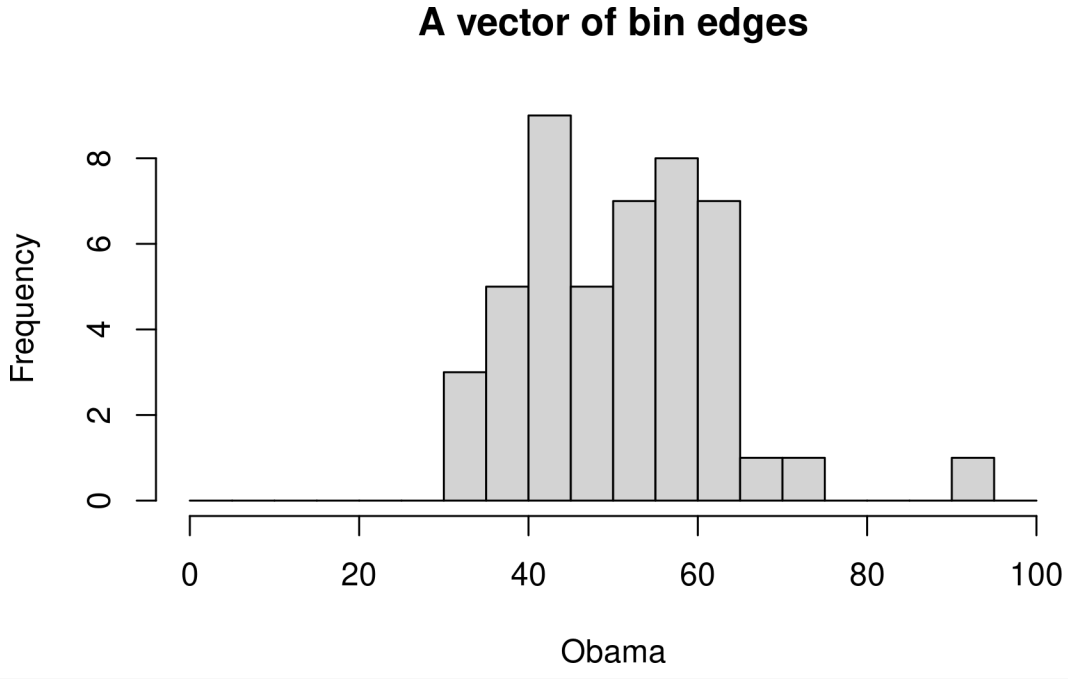
- the results of which are shown in in the next 5 figures
- the main argument creates a main title above the plot.

It works for the plot function too:

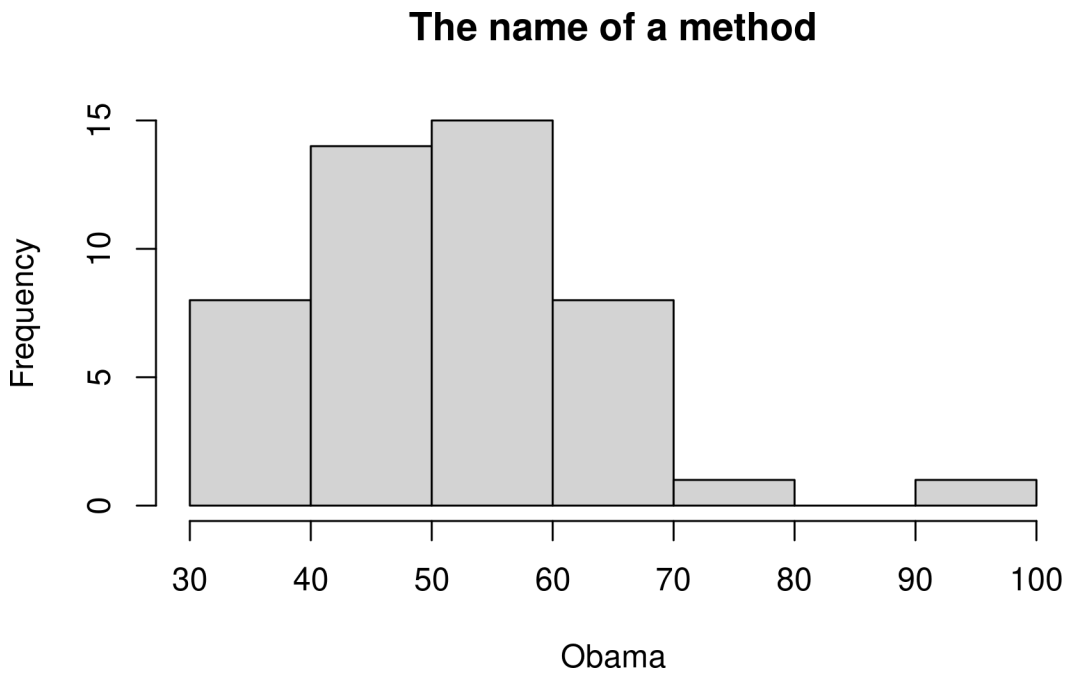
```
  with(obama_vs_mccain,
    hist(Obama, 4, main = "An exact number of bins")
)
```



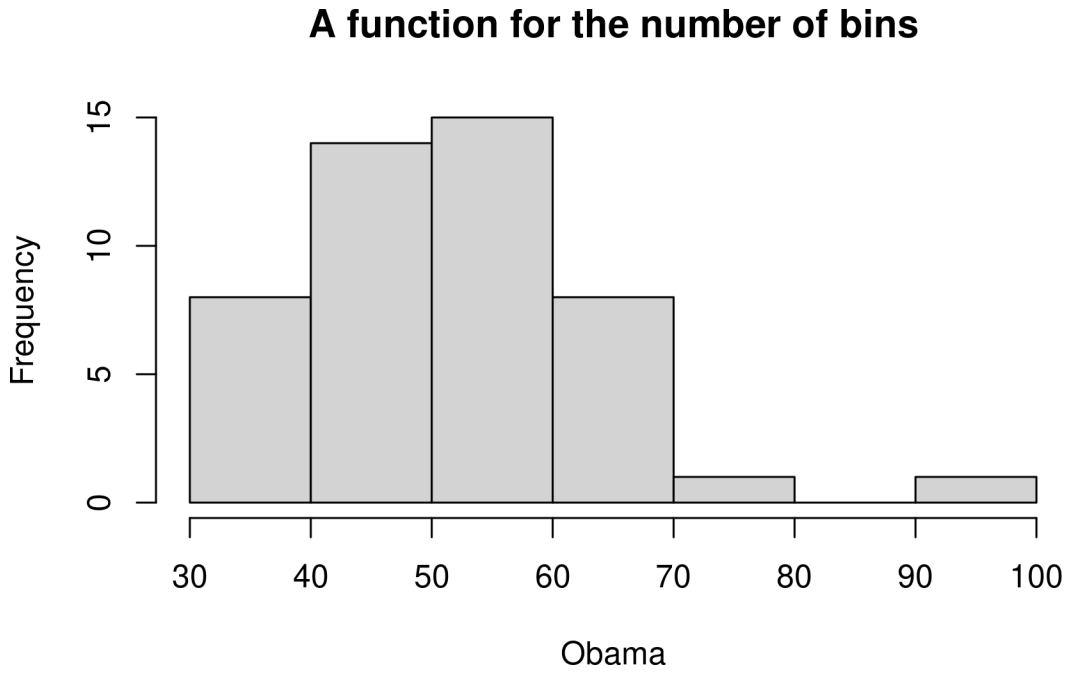

```
with(obama_vs_mccain,
  hist(Obama, seq.int(0, 100, 5), main = "A vector of bin edges")
)
```



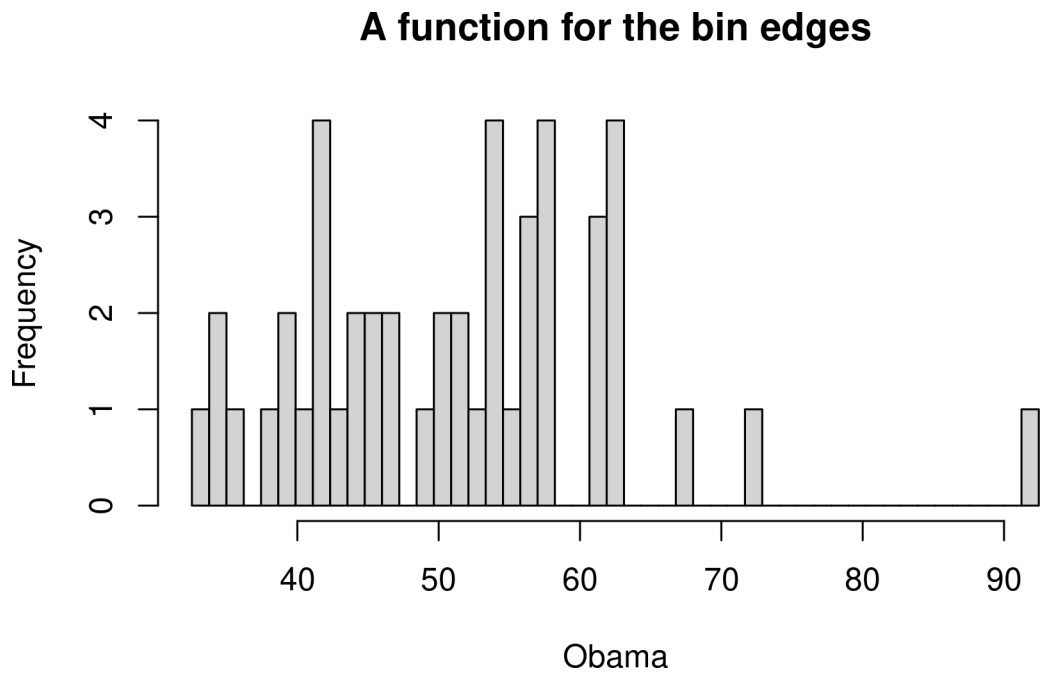

```
with(obama_vs_mccain,
  hist(Obama, "FD", main = "The name of a method")
)
```




```
with(obama_vs_mccain,
  hist(Obama, nclass.scott, main = "A function for the number of bins")
)
```




```
binner <- function(x)
{
  seq(min(x, na.rm = TRUE), max(x, na.rm = TRUE), length.out = 50)
}
with(obama_vs_mccain,
  hist(Obama, binner, main = "A function for the bin edges")
)
```



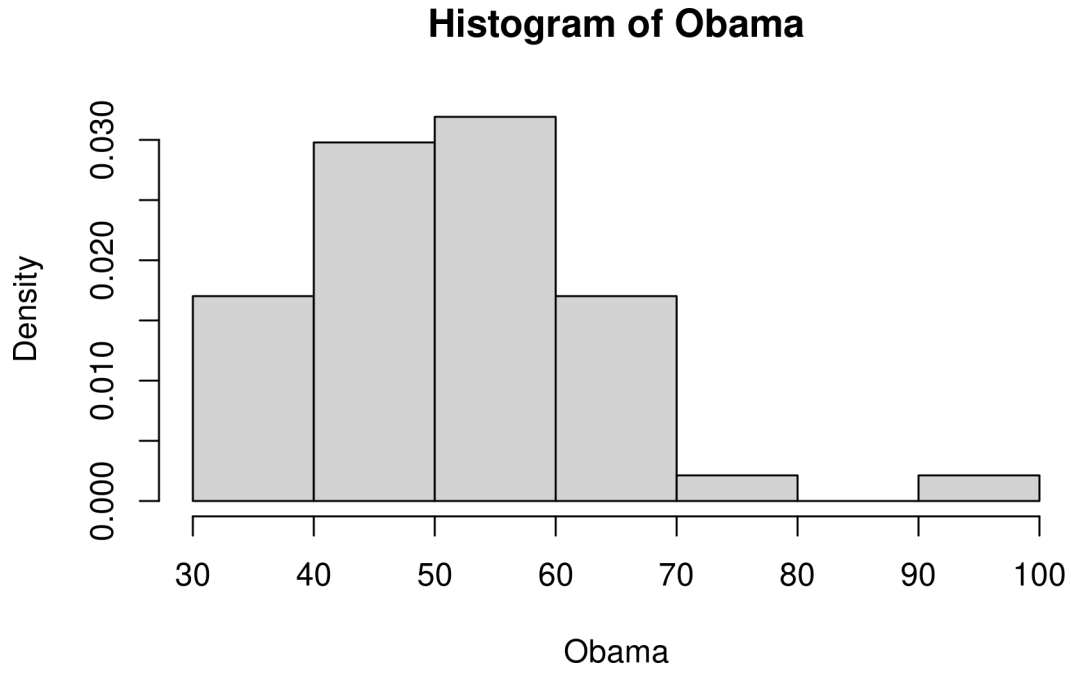
The freq argument controls

- whether or not the histogram shows counts or probability densities in each bin.

It defaults to TRUE if and only if the bins are equally spaced.

here's the output:

```
with(obama_vs_mccain, hist(Obama, freq = FALSE))
```



6.2.5.8.2 ggplot2 graphics

- ggplot2 histograms are created by adding a histogram geom.

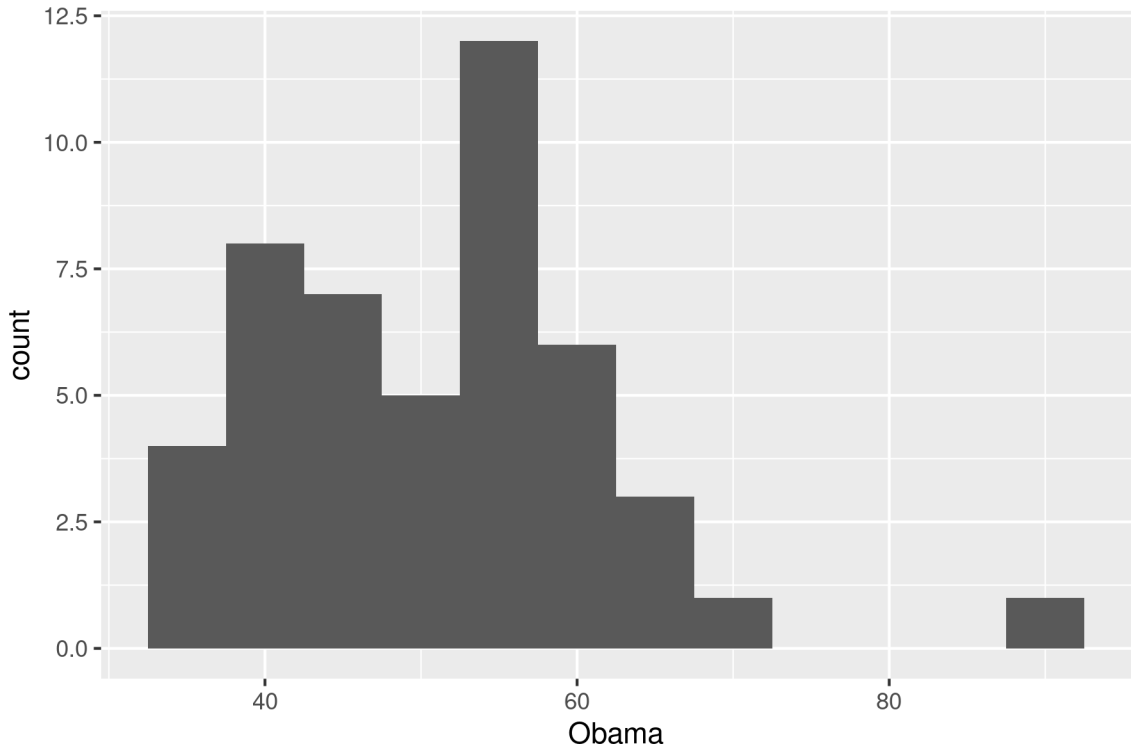
  - Bin specification is simple:
  - just pass a numeric bin width to geom_histogram.

The rationale is to force you to manually experiment

- with different numbers of bins,
- rather than settling for the default.

Here is the usage:

```
ggplot(obama_vs_mccain, aes(Obama)) +
  geom_histogram(binwidth = 5)
```

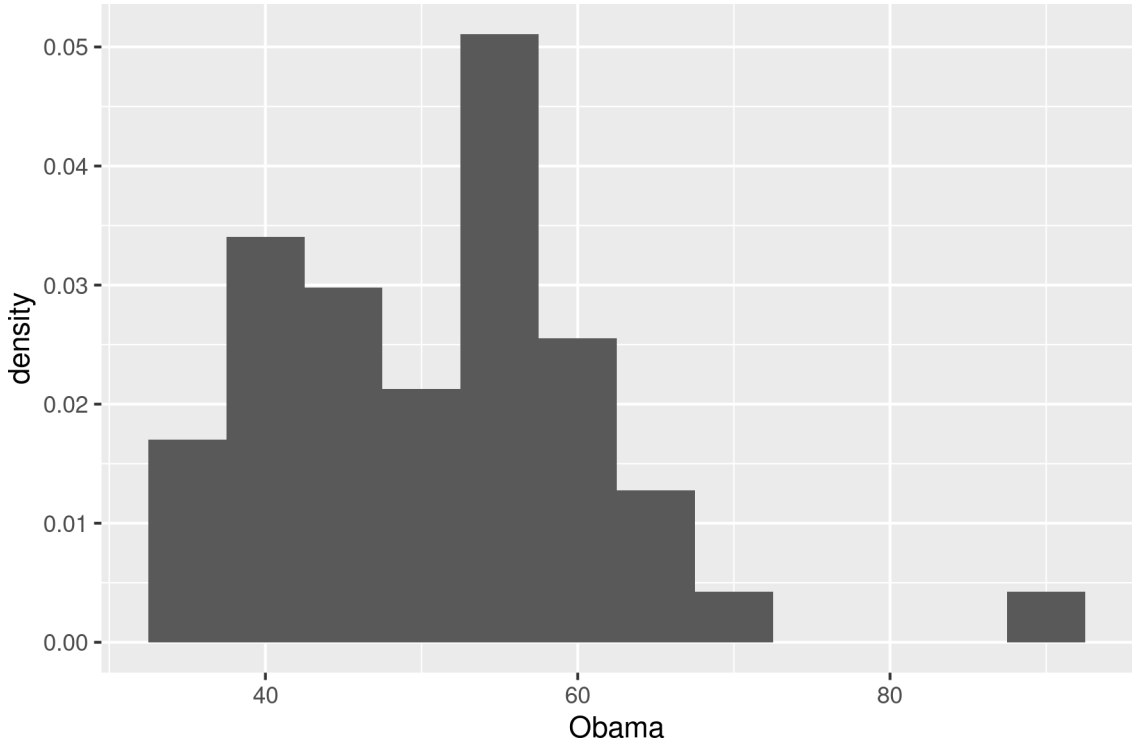


You can choose between counts and densities

- by passing the special names ..count.. or ..density.. to the y-aesthetic.

This figure demonstrates the use of ..density..:

```
ggplot(obama_vs_mccain, aes(Obama, ..density..)) +
  geom_histogram(binwidth = 5)
```



6.2.5.9 Boxplots

- If you want to explore the distribution of lots of related variables,
  - you could draw lots of histograms.

For example, if you wanted to see the distribution of Obama votes by US region,

- you could use latticing/faceting to draw 10 histograms.

This is just about feasible, but it doesn't scale much further.

If you need a hundred histograms,

- the space requirements can easily overwhelm the largest monitor.

Box plots (sometimes called box and whisker plots)

- are a more space-efficient alternative
- that make it easy to compare many distributions at once.

You don't get as much detail as with a histogram or kernel density plot,

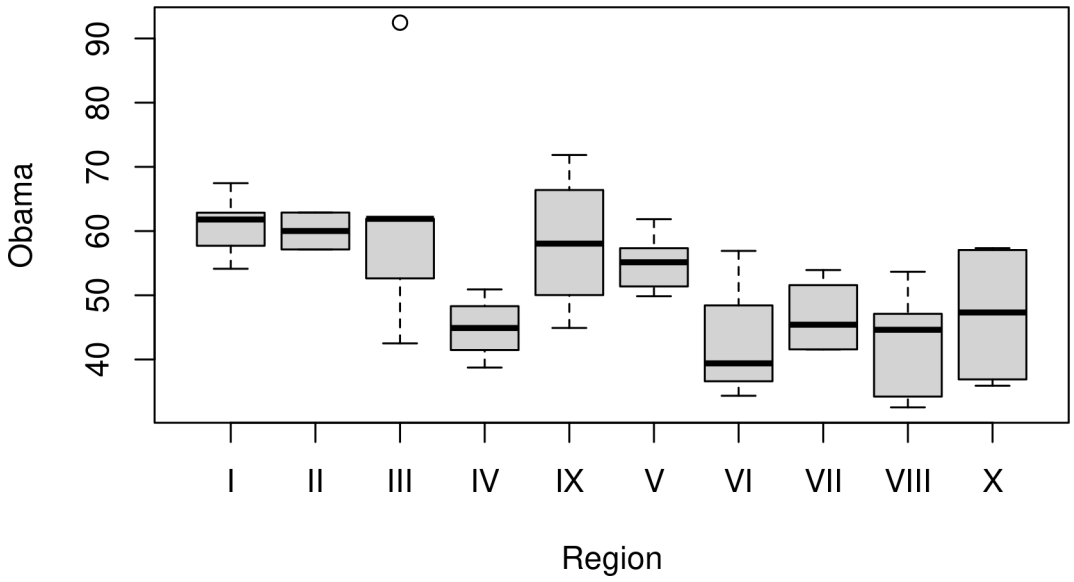
- but simple higher-or-lower and narrower-or-wider comparisons can easily be made.

6.2.5.9.1 base graphics

- The base function for drawing box plots is called boxplot;
  - it is heavily inspired by lattice,
  - insofar as it uses a formula interface and has a data argument.

This shows the usage:

```
boxplot(Obama ~ Region, data = obama_vs_mccain)
```



This type of plot is often clearer

- if we reorder the box plots from smallest to largest, in some sense.

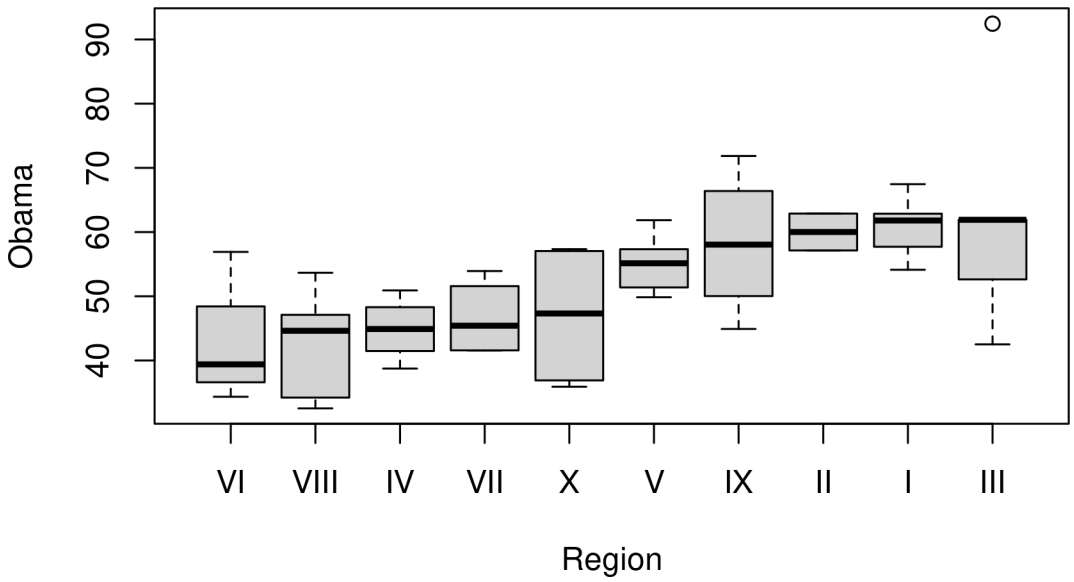
The reorder function changes the order of a factor's levels,

- based upon some numeric score.

In this figure we score the Region levels

- by the median Obama value for each region:

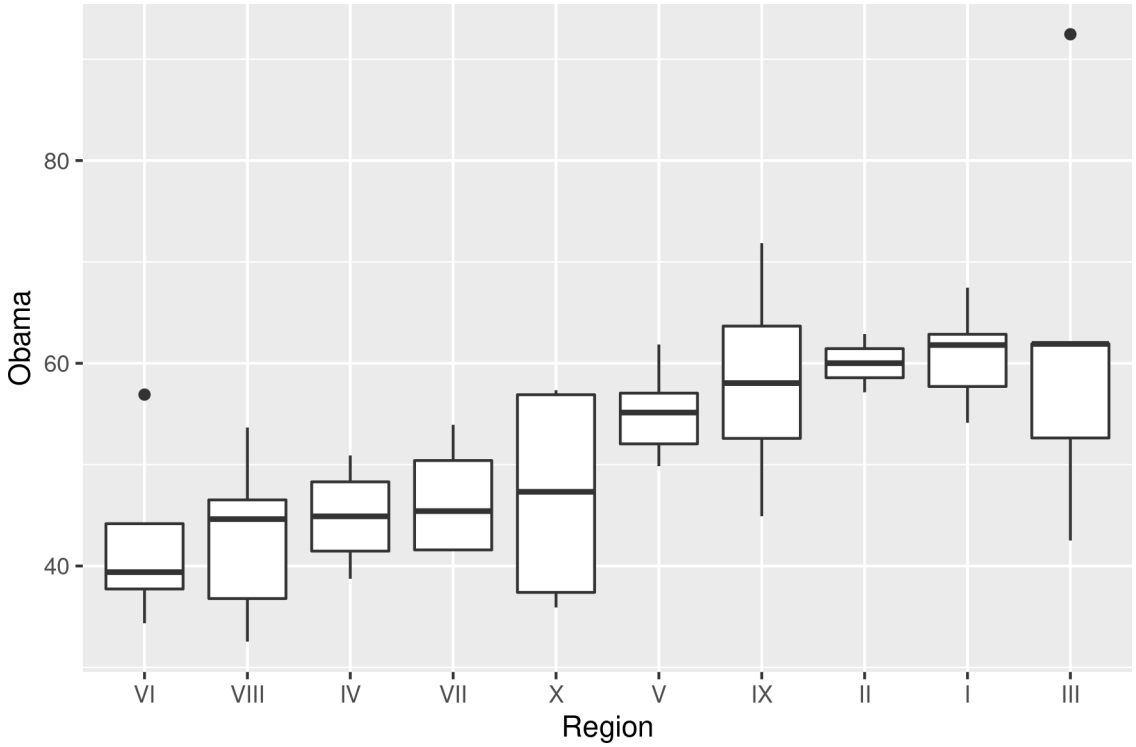
```
ovm <- within(
  obama_vs_mccain,
  Region <- reorder(Region, Obama, median)
)
boxplot(Obama ~ Region, data = ovm)
```



6.2.5.9.2 ggplot2 graphics

- The ggplot2 equivalent box plot, shown next,
  - just requires that we add a geom_boxplot:

```
ggplot(ovm, aes(Region, Obama)) +
  geom_boxplot()
```



6.2.5.10 Barcharts

- Bar charts (a.k.a. bar plots) are the natural way of
  - displaying numeric variables
  - split by a categorical variable.

In the next examples, we look at

- the distribution of religious identification across the US states.

Data for Alaska and Hawaii are not included in the dataset,

- so we can remove those records:

```
  ovm <- ovm[!(ovm$State %in% c("Alaska", "Hawaii")), ]
# remove Alaska and Hawaii from the dataframe
```

6.2.5.10.1 base graphics

- In base, bar charts are created with the barplot function.
  - As with the plot function,
    - there is no argument to specify a data frame,
  - so we need to wrap it in a call to with.

The first argument to barplot contains the lengths of the bars.

If that is a named vector

- (which it won't be if you are doing things properly
  - and accessing data from inside a data frame),
- then those names are used for the bar labels.

Otherwise, as we do here,

- you need to pass an argument called names.arg to specify the labels.

By default the bars are vertical,

- but in order to make the state names readable we want horizontal bars,
- which can be generated with horiz = TRUE.

To display the state names in full,

- we also need to do some fiddling with the plot parameters,
  - via the par function.
- For historical reasons, most of the parameter names are abbreviations
  - rather than human-readable values, so the code can look quite terse.

It's a good idea to read the ?par help page before you modify a base plot.

```
?par
```

The `las` parameter

- (short for "label axis style")
- controls whether labels are
  - horizontal, vertical, parallel, or perpendicular to the axes.

Plots are usually more readable if you set las = 1, for horizontal.

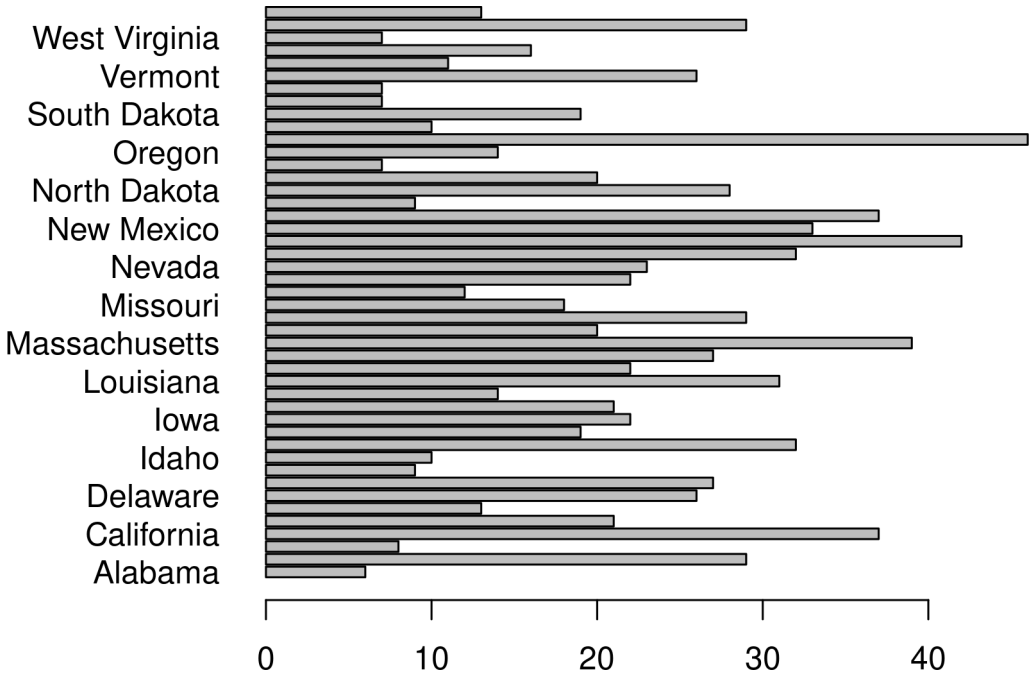
The `mar` parameter

- is a numeric vector of length 4,
- giving the width of the plot margins
  - at the bottom/left/top/right of the plot.

We want a really wide left hand side to fit the state names.

This figure shows the output of the following code:

```
par(las = 1, mar = c(3, 9, 1, 1))
with(ovm, barplot(Catholic, names.arg = State, horiz = TRUE))
```



Simple bar charts like this are fine,

- but more interesting are bar charts of several variables at once.

We can visualize the split of religions by state

- by plotting the Catholic, Protestant, Non.religious, and Other columns.

For plotting multiple variables,

- we must place them into a matrix,
- one in each row (rbind is useful for this).

The column names of this matrix are used for the names of the bars;

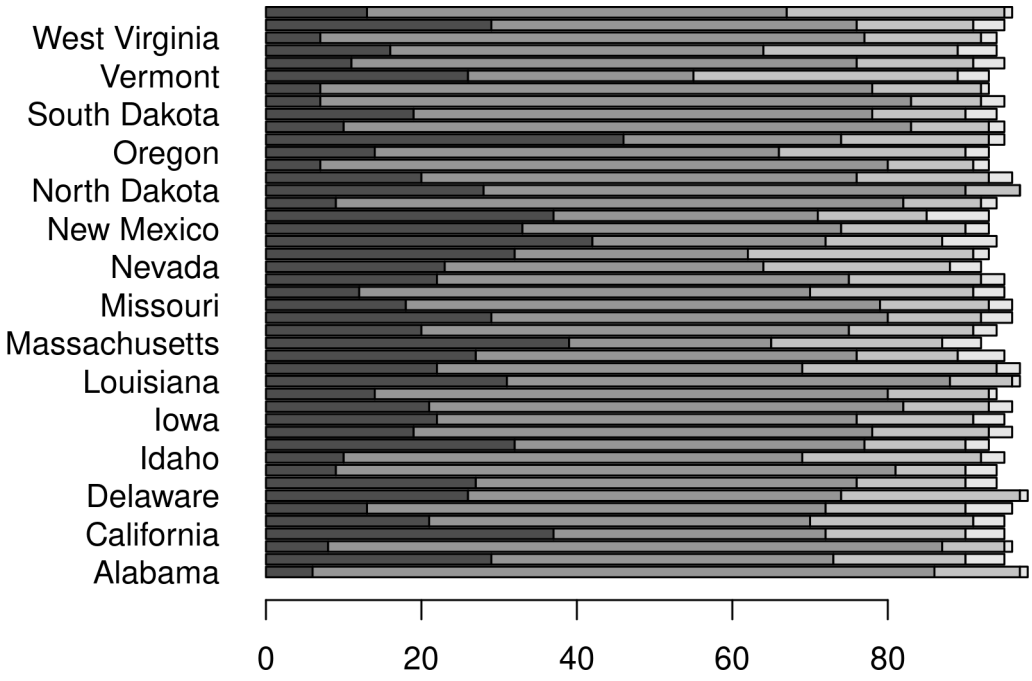
- if there are no column names
- we must pass a names.arg like we did in the last example.

By default, the bars for each variable are drawn next to each other,

- but since we are examining the split between the variables,
- a stacked bar chart is more appropriate.

Passing beside = FALSE achieves this, as illustrated in the next figure:

```
religions <- with(ovm, rbind(Catholic, Protestant, Non.religious, Other))
colnames(religions) <- ovm$State
par(las = 1, mar = c(3, 9, 1, 1))
barplot(religions, horiz = TRUE, beside = FALSE)
```



6.2.5.10.2 ggplot2 graphics

- ggplot2 requires a tiny bit of work be done to the data to replicate this plot.

We need the data in long form,

- so we must first melt the columns that we need:

```
religions_long <- melt(
  ovm,
  id.vars = "State",
  measure.vars = c("Catholic", "Protestant", "Non.religious", "Other")
)
```

Like base, gplot2 defaults to vertical bars;

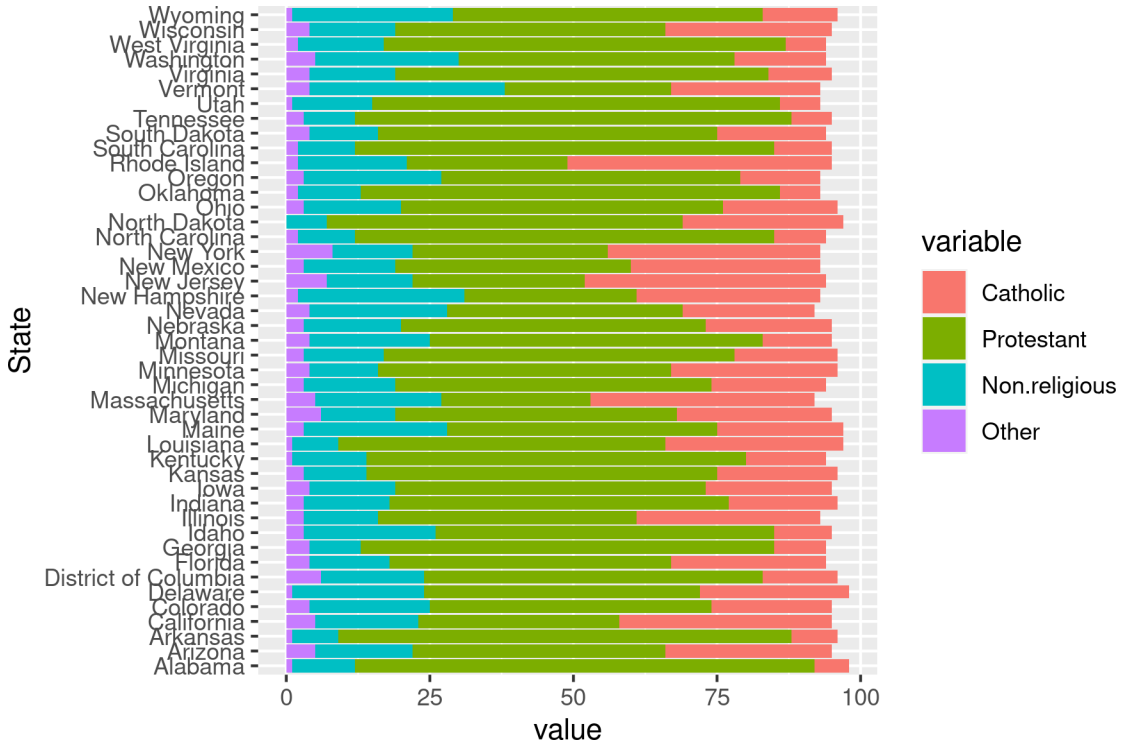
- adding coord_flip swaps this.

Finally, since we already have the lengths of each bar in the dataset

- (without further calculation)
- we must pass stat = "identity" to the geom.

Bars are stacked by default, as shown in this figure:

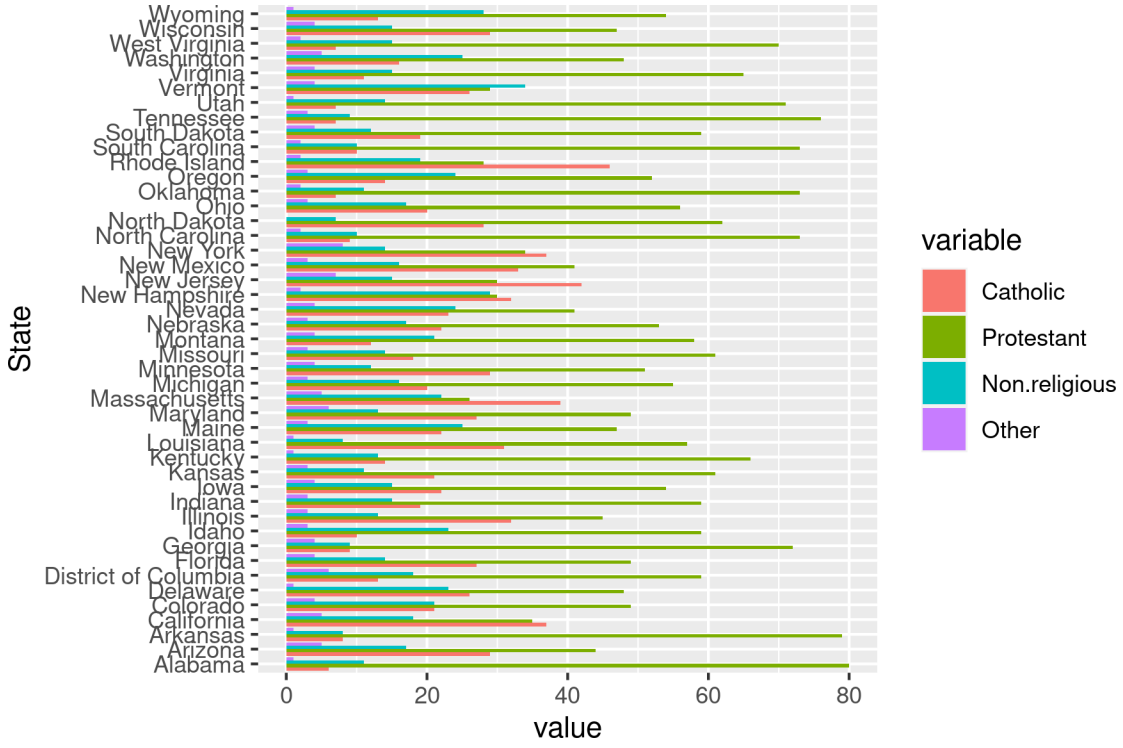
```
ggplot(religions_long, aes(State, value, fill = variable)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



To avoid the bars being stacked,

- we would have to pass the argument position = "dodge" to geom_bar.

```
ggplot(religions_long, aes(State, value, fill = variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip()
```



6.2.5.11 Other graphics systems

- There are many packages that contain plotting capabilities
  - based on one or more of the three systems.

For example, the vcd package has lots of plots

- for visualizing categorical data,
- such as mosaic plots and association plots.

plotrix has loads of extra plot types,

- and there are specialist plots scattered in many other packages.

latticeExtra and GGally extend the lattice and ggplot2 packages,

- and grid provides access to the underlying framework that supports both these systems.

6.2.5.12 interactive visualization You may have noticed that all the plots covered so far are static.

There have in fact been a number of attempts

- to provide dynamic and interactive plots.9

There is no perfect solution yet,

- but there are many interesting and worthy packages that attempt this.

gridSVG lets you write grid-based plots (lattice or ggplot2) to SVG files.

These can be made interactive,

- but it requires some knowledge of JavaScript.
  - playwith allows pointing and clicking to interact with base or lattice plots.
  - iplots provides a whole extra system of plots with even more interactivity.
- It isn't easily extensible, but the common plots types are there,
  - and you can explore the data very quickly via mouse interaction.

googleVis provides an R wrapper around Google Chart Tools,

- creating plots that can be displayed in a browser.

rggobi provides an interface to

- GGobi (for visualizing high-dimensional data), and

rgl provides an interface to OpenGL

- for interactive 3D plots.

The animation package

- lets you make animated GIFs or SWF animations.

6.2.5.13 Interfacing to d3 javascript graphics

- The r2d3 package provides an interface to javascript d3 visualizations
  - d3 Data Driven Documents
  - A gallery of d3 graphics
  - And on Observable

6.2.5.14 Summary

- There are loads of summary statistics that can be calculated.
- R has three plotting systems: base, lattice, and ggplot2.
- All the common plot types are supported in every system.
- There is some support in R for dynamic and interactive plotting.

6.2.5.15 Cites examples from

- Learning R, by Richard Cotton, 2013, O'Reilly Media, Inc.