

THE DOT PLOT: A GRAPHICAL DISPLAY FOR LABELED QUANTITATIVE VALUES

William G. Jacoby
Department of Political Science
Michigan State University
303 South Kedzie Hall
East Lansing, MI 48824
E-mail: jacoby@msu.edu
Web: <http://polisci.msu.edu/jacoby>

May 2006

The dot plot is an extremely useful tool for obtaining pictorial representations of quantitative information.¹ This display method is very flexible and potentially applicable to any situation where numeric values are associated with descriptive labels. For example, dot plots can be used to depict raw data, frequency counts, descriptive statistics, and parameter estimates from statistical models. A carefully constructed dot plot contains an enormous amount of information. More important, a dot plot can convey that information in a way that overcomes some of the problematic elements of its closest “competitors,” the bar chart and the pie chart. This article will introduce the dot plot, explain its major features, and provide some example applications. The discussion will then move on to explain the `dotplot` function in the `lattice` package of the R software environment and illustrate the ways that this function can be adapted to produce a wide variety of dot plots.

DEFINITION AND EXAMPLES

A dot plot is a two-dimensional graphical display of objects, showing some quantitative characteristic of those objects. One axis of the dot plot (usually the horizontal) is a scale covering the range of quantitative values to be plotted. The other axis (usually the vertical) shows descriptive labels that are associated with each of the numeric values.² The data objects usually are sorted according to the quantitative values. Plotting symbols are placed within the display area of the dot plot, locating each data object at the intersection position for its label on the vertical axis and associated numeric value on the horizontal axis. While this simple definition covers the basic features of the dot plot, it is important to emphasize that the utility and flexibility of such a display come from the details that are included in its construction. Let us consider several examples that will illustrate the dot plot’s various features and advantages.

Basic Data Display

The simplest application of the dot plot is to display the empirical distribution of values on a single variable. Of course, there are other univariate graphs that do the same thing

(e.g., the histogram, the box plot, etc.); but, the dot plot is the only kind of display that explicitly incorporates labeling information such that the reader can determine immediately which observations correspond to specific data values. Of course, the amount of data that can be contained within a single dot plot is somewhat limited. But, given sufficient clarity in the rendering device, a dot plot can provide data values for a surprisingly large number of observations.

Figure 1 shows a dot plot of policy priority scores for the American states in 1992 (Jacoby and Schneider 2001). This variable is based upon the states' proportionate spending levels across fifteen program areas. Larger values on this variable indicate that a state spent more on a set of policies that Jacoby and Schneider labeled "collective goods," including highways, parks, law enforcement, and so on. Smaller values correspond to more spending on "particularized benefits" such as welfare, health care, and employment security. This variable is measured at the interval level, so the specific numerical values are arbitrary; however, *differences* in the scores are meaningful and interpretable as proportions. For example, if state A has a score of 0.50 and state B has a score of 0.52, then B spends 2% more of its funds on collective goods policies than state A (or, alternatively, B spends 2% less on particularized benefits policies than A).

The information in Figure 1 is easy to interpret. The labels in the margin of the vertical axis are the state names. The spending priority score for each state can be determined by examining the location of the plotted point within that row: The farther to the right, the larger the proportion of the state's budget devoted to collective goods; the farther to the left, the greater the state's proportionate spending on particularized benefits.

The dot plot of a univariate dataset is effectively the same as a transposed quantile plot of the data. Therefore, the display shows the empirical cumulative distribution function for the variable. This, in turn, provides information about distributional shape. The general rule is that local density in the data is reflected by the local slope of the plotted points. Shallow local slope corresponds to relatively sparse regions within the distribution (e.g., the

tails of a unimodal distribution) while steep or nearly vertical local slope indicates a densely-populated segment within the variable's range (i.e., a mode). Thus, the single steep array of points in the central region of Figure 1 indicates that the distribution of spending priority scores is unimodal. And, while the distribution appears to be nearly symmetric (because the shallower arrays of points near the upper and lower ends are about the same size), the lower tail of the distribution is slightly more pronounced than the upper tail (the shallow string of points in the lower left is somewhat more pronounced than that in the upper-right corner of the display).

The dot plot enables visual estimates of summary statistics for the distribution. For example, the extremes are easily identified, at about 0.498 and 0.546. The median is the data value that occurs exactly halfway down (or up) the vertical axis; here, it appears to be slightly more than 0.52 (say, about 0.522). The interquartile range is the difference between the values that fall one-fourth of the way up from the bottom, and one-fourth of the way down from the top. These values are about 0.515 and 0.530, respectively, yielding an estimated IQR of about 0.015. The visual estimates (honestly, I didn't cheat by looking at the data values!) are very close to their more precisely-calculated counterparts: The five-number summary for these data is (0.497, 0.518, 0.525, 0.531, 0.547) and the IQR is 0.013.

The dot plot also facilitates the identification of particular states within the overall distribution, as well as differences between specific states. The horizontal dotted lines in the graph facilitate table look-up (i.e, the ability to associate individual data values with labels). For example, Figure 1 makes it easy to see that Massachusetts is the state that spends the highest proportion of its budget on particularized benefits while Wyoming spends the most on collective goods (i.e., these observations have the lowest and highest scores, respectively). And, South Carolina devotes about one percent more of its budget to collective goods than does Connecticut (i.e., their scale scores are about 0.52 and 0.51, respectively). In summary, the dot plot provides a great deal of information about both a variable's distribution and the specific observations within the data.

Figure 2 illustrates a variation of the basic dot plot, showing state education spending for fiscal year 2000, in dollars per capita. This display provides exactly the same kind of information that was discussed with respect to Figure 1. But, now, the varying-length horizontal lines emphasize the shape of the point array more clearly. The absence of a shallow-sloped string of points in the lower left corner indicates that there is positive asymmetry in this distribution. Furthermore, the separation between the top two points and the remaining data suggests that Vermont and Alaska might be considered mild outliers, relative to the rest of the distribution.

Note that the horizontal dotted lines in Figure 2 all emanate from the zero point on the horizontal axis. Therefore, the line lengths for different states can be compared to facilitate magnitude judgments about the data values. For example, it is easy to see that Vermont and Alaska each spend more than twice the per capita amount on education than Massachusetts and Florida. Note that such magnitude comparisons were inappropriate for the interval-level data that were displayed in Figure 1. Therefore, in order to discourage such interpretations, the horizontal lines in that earlier graph extended across the entire width of the plotting region. Thus, careful attention to the details of a dot plot can affect the kind of information that readers are able to extract readily from the display.

Dot Plots for Comparing Data Values

Dot plots are very useful for situations where the objective is to compare repeated measurements of a single variable across a set of observations. As an example, Figure 3 shows state education expenditures for 1990 and 2000. The data values are dollars per capita, adjusted for inflation, so they are directly comparable across the ten-year time span. Each state's two data values are shown as a pair of points plotted along a single horizontal line for that state. The major visual feature of the display is that the "x" falls to the right of the "o" in almost every horizontal line.³ Of course, this shows that the vast majority of states increased their education spending across the decade. The two exceptions are Alaska, which

showed a fairly sharp decrease in per capita education spending and Maine, which showed virtually identical spending levels in both years.

Note that, unlike Figure 2, the horizontal lines in Figure 3 do *not* emanate from zero, even though the data values are actually ratio-level quantities. The reason is that the display is intended to emphasize the *differences* between the respective states' expenditures across the years, and not their absolute levels of spending. Therefore, the horizontal dimension of this dot plot only runs between a pair of arbitrary values just outside the minimum and maximum data values. This serves to increase the physical distance between the two plotting symbols on each line, making it easier to judge the sizes of each state's expenditure change. In this case, most states showed an increase of about \$250-\$300 per capita. However, three states (New Hampshire, Michigan, and Vermont) stand out with much larger increases in their respective spending levels.

In Figure 3, the states are sorted by their 1990 education expenditures, so the dot plot effectively uses that year's distribution as a baseline for comparison. This display is useful if the main question is "How did the states' expenditures change after 1990?". On the other hand, one could ask the alternative question, "From where did the states come, prior to 2000?". In that case, the latter year would probably be a better benchmark for comparison. Figure 4 shows the same information as Figure 3, but with states now sorted according to their 2000 education spending values. Here, it is immediately apparent that several of the states that spent the most in 2000 also showed the largest *increases* in spending across the decade. And, while Alaska's spending actually decreased over time, it still shows the second-highest per capita spending on education. Once again, the "trick" in constructing an effective dot plot lies in adjusting the details to facilitate the kinds of judgments that the analyst wants readers to make when interpreting the graphical information in the display.

The basic dot plot data display can be adapted very easily to enable comparisons across subgroups of observations. For example, Figure 5 is a divided dot plot, showing individual state policy priority scores within separate regions. Again, a great deal of information can

be extracted from this display. The data values are sorted within the regions. And, the order of the regions within the plot is determined by their respective medians. This makes it easy to see that northeastern states tend to have the smallest scores, followed by midwestern, southern, and western states. Visual inspection indicates that the region medians fall at about 0.512, 0.523, 0.525, and 0.530, respectively. The actual, calculated, medians are 0.512, 0.525, 0.527, and 0.533 (again, I didn't cheat in my visual estimates!).

Intra-region variability can be assessed through the slope of the point array for a particular region, or through the spread of its points along the horizontal axis. Hence, in Figure 5, western states show the greatest variation in priority scores (the slope is shallowest for that region, and the points are spread out wider than any other region). In contrast, southern states seem to show the least dispersion; that region has the steepest array of points, covering the narrowest interval on the horizontal axis. Of course, this same information about regional differences could be obtained through other data analysis tools. But, it is impossible to deny the ease with which it can be extracted from the dot plot. And, the graphical display makes it easy to see potentially misleading elements of the data, such as within-region outliers (e.g., Vermont in the northeast and California in the west), that might affect the calculated values of regional summary statistics.

Statistical Summaries

Dot plots are certainly not limited to displays of raw data. They can also be used very effectively to show summaries of data values within partitions of an overall dataset either across subsets of the observations or across separate variables (or both). For example, Figure 6 uses information drawn from the 2004 National Election Study to show percentage differences in votes for Bush and Kerry across party identification categories. Since there are only seven quantitative values (and, therefore, little potential for serious difficulties with table look-up), the horizontal lines have been omitted from the interior of the display. This follows Tufte's (1983) dictum to "maximize the data-ratio" and produces a "cleaner" graphical representation of the information. At the same time, a vertical reference line has been added

at the zero point on the horizontal axis. This provides a useful basis for evaluating the quantitative information in the display, since it represents a position of partisan neutrality, or equal voting support for both candidates.

Note that the plotted points in Figure 6 are *not* sorted by their quantitative values. Instead, the ordering on the vertical axis is determined by the categories of the party identification variable which are, themselves, ordered according to the direction and intensity of individual partisan attachments. This results in a nonmonotonic array of points that is substantively interesting: It shows that nonstrong Democratic identifiers were less supportive of Kerry than independent leaners. This finding is somewhat contrary to expectations based upon theoretical considerations, but it is a pattern that has been noticed many times in the previous research literature (e.g., Petrocik 1974; Keith et al. 1991). To reiterate a point made earlier, the same information could be drawn from a crosstabulation of party identification versus 2004 voting choices. But, that would almost certainly take greater cognitive effort on the part of the observer. With the graphical display, even the relatively mild anomalous feature of the data passes the “interocular trauma test” quite easily.

When sample statistics are regarded as estimates of population parameters it is, of course, desirable to incorporate some indication of the uncertainty in those estimates due to sampling error. This is accomplished very easily in a dot plot, by adding error bars to the display. Figure 7 uses more data from the 2004 NES to show the mean importance ratings that survey respondents assigned to ten issues (scored on a one-to-five scale, with larger values indicating the issue is more important). The symmetric horizontal “wings” around each plotted point correspond to the 95% confidence interval for each mean. Again, the horizontal reference lines are omitted from the dot plot, not only because of the relatively small number of points, but also because they would interfere with visual perception of the error bars.

The plotted points in Figure 7 are, once again, sorted by their quantitative values. So, it is easy to see that government health insurance was the regarded as the most important issue (by a significant margin, since the confidence interval for that issue does not overlap

the adjacent interval for women’s roles) while aid to minorities and environmental concerns were viewed as the least important issues (again, by significant margins). While the exact mean values for the remaining seven issues vary somewhat, they all fall within a relatively narrow interval (between about 3.75 and 3.95) and the differences between them may well be due to sampling error, since their confidence intervals all overlap each other.

A dot plot is a convenient and succinct way to report estimates from a statistical model. As an example, Figure 8 shows the coefficients obtained when 2004 NES respondents’ general attitudes toward government spending and services are regressed on their ideology, party identification, and preferences toward federal spending in eleven specific policy areas. Standardized coefficients are shown, because the independent variables are measured using different units; ideology and party identification are seven-point scales (with larger values indicating liberal and Democratic identifications), while the spending items are three-point scales (with larger values indicating a preference for more spending in a policy area). Since standard errors are usually considered inappropriate for standardized coefficients, no confidence intervals are shown. However, different plotting symbols are used for variables whose nonstandardized coefficients are statistically discernible from zero, using a one-sided test at the 0.05 level. As in the two previous dot plots, horizontal lines are omitted from the plotting area and a vertical dashed reference line is inserted at the zero point.

The dot plot in Figure 8 is particularly useful in this case, because researchers would certainly be more interested in the *relative* sizes of the regression coefficients, rather than their exact numerical values. And, from the array of points in the display, it is clear that general spending attitudes are *not* a simple additive function of policy-specific preferences. Instead, they are affected by more general political orientations (ideology and party identification) and feelings about spending that benefits particular beneficiaries (i.e., Social security, child care, schools, poor people, and welfare). General spending attitudes are, basically, unrelated to preferences about spending in foreign affairs, law enforcement, highways, and science. I

have argued elsewhere that patterns like this are very revealing about the subjective nature of public opinion toward government spending (e.g., Jacoby 2005).

ADVANTAGES OF DOT PLOTS

Dot plots are probably not very well known to most political scientists. They are covered extensively in Cleveland's work (e.g., 1993; 1994) on statistical graphics, I have used them in several of my own works (Jacoby 2006; Jacoby and Schneider 2001), and they also appear in recent articles written by a few other authors (e.g., Clinton, Jackman, Rivers 2004; Baker 2005; Jerit, Barabas, Bolsen 2006). But, it is definitely fair to say that the dot plot is a relatively uncommon graphical display in the political science research literature. This is unfortunate, because dot plots have some definite advantages over their "competitors" for displaying labeled data: pie charts and bar charts.

First, there is a simple, practical advantage: Dot plots can show a larger number of data points than either pie charts or bar charts. Pie charts are necessarily limited to a fairly small number of distinct "wedges;" otherwise, visual perception of the quantitative information becomes nearly impossible. With bar charts, the width of the bars necessarily becomes narrower as the number of distinct data values increases. In fact, with a large number of plotted values, a bar chart becomes virtually indistinguishable from a dot plot (e.g., think of the graphs that *Consumer Reports* uses to show overall ratings of models within a given category of product). As mentioned earlier, a dot plot can include a surprisingly large number of points it is really only limited by the space available in the display medium. For example, a dot plot with one hundred data values, displayed on a standard journal-size page, could be interpreted very easily.

A second, theory-based, advantage is that dot plots facilitate relatively accurate graphical perception. Visual processing of a pie chart requires the observer to make comparative judgments about the angles, arcs, and/or sizes of the various wedges within the circular diagram. In contrast, a dot plot only involves comparisons of point locations along a common

scale. Cleveland and McGill (1984; 1985; also see Cleveland 1994) show that the latter task is usually carried out much more accurately than the former.

With bar charts, a different problem emerges. A bar chart *should* be interpreted using the relative heights (or widths) of the bars along a common scale. But, Cleveland (1984) argues that bar charts actually encourage observers to make judgments based upon the relative *sizes* of the bars within the display. And, if the scale in the bar chart begins at some arbitrary value, then it is inappropriate to regard the lengths and/or areas of the bars as any sort of meaningful information about the relative magnitudes of the quantitative values being displayed in the chart. Figure 9A illustrates this problem, using a bar chart of the same mean issue importance ratings that were shown back in Figure 7. The length of the bar for “Govt. health insurance” is about twice the length of the bar for “Defense spending.” But a glance at the horizontal axis shows that the mean importance rating for the former issue is definitely not two times the size of the mean for the latter issue!

If magnitude judgments were appropriate for these data (which they are not), then one could consider dealing with the preceding problem by setting the scale origin to zero on the horizontal axis, as in Figure 9B. But, this creates still another problem. The distance from the origin to the minimum plotted value (3.40, in this case) is much larger than the range of values included in the display (3.40 to 4.10, or 0.70). This compresses the ends of the bars near the right side of the display, and hampers visual perception of the relative lengths.

A properly-constructed dot plot avoids these problems by either extending the horizontal reference lines across the entire width of the plotting region (as in Figures 1, 3, 4, and 5) or omitting them entirely (as in Figures 6, 7, and 8). In either case, the display provides no visual cues that would encourage inappropriate judgments about the relative sizes of the data values. If magnitude judgments are appropriate for the data, then the reference lines in a dot plot can be extended from the origin of the scale out to the plotted data values. Of course, this is exactly what was done in Figure 2.

A third potential advantage emerges when a dot plot is used to display the distribution of values on a single variable. As explained earlier, such a display is effectively a transposed quantile plot. It shows all of the data and, therefore, provides a particularly accurate depiction of distributional shape. Just as with a quantile plot, a dot plot avoids potential distortions that may be introduced by the binning process required to construct a more traditional histogram.

CREATING DOT PLOTS IN R

Most statistical software can be used to generate dot plots. STATA, SPSS, and SYSTAT all have routines that are either explicitly designed, or easily adapted, for this purpose. Friendly (1991) provides macros that create dot plots in SAS. But, as is often the case, R provides the most powerful facility and flexibility for creating this kind of graphical display (R Development Core Team 2006).⁴ The cost involves learning the details of the R functions that produce dot plots, along with the complexities (at least for beginners) in the ways that these functions can interact with other aspects of the R environment.

Some General Principles

Before proceeding, there are three general points that it will be useful to keep in mind: First, there are always multiple ways to achieve the same objective in R. Here, I will emphasize the `dotplot` function in the `lattice` package (Sarkar 2006). However, many of the same displays could have been produced with the `dotchart` function of the traditional R graphics system. More generally, there are always alternative strategies for preparing data and supplying parameters to R functions, apart from those I will explain below.⁵

Second, trellis graphs (i.e., the type produced by the `lattice` package) are created by issuing a *general display function* which, in turn, calls a *panel function*. The general display function sets up the exterior components of the graph (i.e., axes, scales, titles, and so on) while the panel function deals with everything within the plotting region, itself (i.e., points,

reference lines, etc.). Distinguishing between these two components is important for specifying any non- default parameter values in a graph (e.g., plotting symbols, line characteristics, and so on).

Third, the `lattice` package effectively regards a dot plot as a scatterplot between a categorical variable (or a *factor* in R nomenclature) and a quantitative variable (a *vector* in R- speak). It is very useful to think about the dot plot this way when trying to produce specific, non- standard, displays. And, it is also very helpful to keep this in mind when trying to understand why the dot plot function sometimes produces strange and unexpected results!

Creating Simple Dot Plots

In order to produce a dot plot of values stored in variable x , with labels in variable y , and both x and y contained in an R data frame called *dataset*, one could use the following general display function:

```
> dotplot(y ~ x, data = dataset, other optional arguments)
```

The only required argument in the preceding function is $y \sim x$. This is actually a simple formula in the R modeling language; it will produce a dot plot with the labels from y listed along the vertical axis, and corresponding points located at the proper horizontal location according to their respective x values.⁶ The `data` argument is optional; of course it specifies the data frame containing x and y . There are many other arguments that can be included in the function call, and quite a few of them will be presented in the discussion below.

Let us begin with a very simple and small (in terms of the dataset size) example. Table 1 shows a subset of 15 observations sampled randomly from Otis Dudley Duncan's famous dataset on the prestige of various occupations. Here, we have only the occupation names and the prestige scores assigned to each one. The following statements would load the `lattice` package, read the data into an R data frame and create a dot plot, using the default settings for the `dotplot` function:

```
> library(lattice)
> occ.prest.subset <- read.table(occprest.txt, header = T)
> dotplot(occup ~ prestige, data = occ.prest.subset)
```

The resultant dot plot is shown in Figure 10. Note that the occupation labels appear in alphabetical order along the vertical axis. This is problematic because the data values are random with respect to this arrangement. Therefore, it is difficult to interpret the information in the display. This occurs because R regards “occup” as a factor, with levels defined by its unique values. And, unless told otherwise, R places the levels of a factor into alphabetical order. That, in turn, determines the ordering on the vertical axis of the dot plot.

The problem is fixed by changing the order of the factor’s levels. Given a factor (say, *factor*) and a quantitative variable (say, *X*), the `reorder` function can be used as follows to sort the factor’s levels so that they are ordered according to the values of the variable⁷:

```
> reorder(factor, X)
```

We could apply this function to the occupational prestige data and generate a new dot plot using the following statements:

```
> occ.prest.subset$occup2 <- reorder(occ.prest.subset$occup,
+   occ.prest.subset$prestige)
> dotplot(occup2 ~ prestige, data = occ.prest.subset,
+   aspect = 1.5,
+   xlab = "Prestige score for occupation",
+   cex = 1.25,
+   col = "black",
+   lty = 2)
```

The first statement creates a new variable called *occup2*; note the use of the fully-qualified variable names. These direct R to the proper data frame (*occ.prest.subset* in this case) and also guarantee that the new variable is added to that data frame, rather than left as a separate object.⁸

In the second statement, the new variable is used as the first component of the formula in the call to `dotplot`. This statement also includes several new elements. The `aspect` argument sets the height-width ratio for the graph; while not absolutely necessary in this particular example, it does give the user more control over the final appearance of the dot plot.

The `xlab` argument sets the label for the horizontal axis. As we saw in Figure 10, the default label is the name of the variable plotted on that axis (which is often not very informative for readers). There is a corresponding `ylab` argument, but it is usually unnecessary since the data labels, themselves, are often self-explanatory. Therefore, the default for `ylab` in a `dotplot` is blank.

The next three arguments modify some of the default settings for the plotting symbols and reference lines. The parameter specified in `cex` (an acronym for "character expansion") sets the size of the plotting symbol. The value is proportional to the "normal" size, so we are telling the function to make the plotting symbols one and one-fourth times the size of the default symbol. The `col` argument sets the color of the plotting symbol (in Figure 10, the points were blue, although that may not have been obvious with monochrome reproduction). And finally, `lty` controls the style of the reference lines; the default, solid, lines are equivalent to "`lty=1`"; "`lty=2`" produces dashed lines, and so on.⁹

Figure 11 shows the dot plot created by the preceding. Notice how several potential problems in Figure 10 have been corrected in this new version: The data values are now sorted, the plotting symbols are more prominent within the display, the horizontal reference lines are a bit less obtrusive, and there is a descriptive label on the horizontal axis.

Using Panel Functions in Dot Plots

In the preceding example, the first three optional arguments (`data`, `aspect`, and `xlab`) control exterior elements of the graphical display. Therefore, they are evaluated directly by the general display function (i.e., `dotplot`), itself. The last three arguments (`cex`, `col`,

and `lty`) are fundamentally different in that they are not interpreted directly by the general display function. Instead, they are passed directly (but invisibly) to the relevant panel function (conveniently called `panel.dotplot`) which actually controls the contents of the plotting region. The following call to `dotplot` would also produce the graph shown in Figure 11. But, it differs from the previous example in that the panel function is called explicitly:

```
> dotplot(occup2 ~ prestige, data = occ.prest.subset,
+ aspect = 1.5,
+ xlab = "Prestige score for occupation",
+ panel = function (x, y) {
+   panel.dotplot(x, y,
+     cex = 1.25,
+     col = "black",
+     lty = 2})
```

The body of the panel function consists of everything that appears between the left and right curly brackets; here, it contains only a single call to `panel.dotplot`. The two arguments that must be supplied to `panel.dotplot` are `x` and `y`. They represent the variables to be plotted on the horizontal and vertical axes of the dot plot, respectively. Unless directed elsewhere, the panel function will take these from the formula specified in the general display function (i.e., the names `prestige` and `occup2` would be substituted for `x` and `y` in this case). All remaining parameters in the panel function are optional. When used, they modify the default display characteristics of the plotted points and the reference lines.

It is not really necessary to use the panel function when the defaults of the `dotplot` function (i.e., round, filled, plotting symbols and reference lines that extend across the entire range of the horizontal axis) are acceptable. But, explicit use of the panel function enables the user to make extensive modifications to the contents of the plotting region. In fact, we would usually bypass the `panel.dotplot` function entirely, since it just calls two further panel functions: `panel.xyplot` to control the plotting symbols and `panel.abline` to control

the reference lines. Thus, the following is still another alternative version of the preceding example:

```
> dotplot(occup2 ~ prestige, data = occ.prest.subset,
+ aspect = 1.5,
+ xlab = "Prestige score for occupation",
+ panel = function (x, y) {
+   panel.xyplot(x, as.numeric(y),
+     cex = 1.25,
+     col = "black",
+     lty = 2)
+   panel.abline(h = as.numeric(y),
+     col = "gray",
+     lty = 2)}))
```

Here, “as numeric(y)” appears as an argument to both functions `panel.xyplot` and `panel.abline`. This specification coerces the vertical-axis variable into a numeric vector; the result is the ordering of the factor’s levels. This enables `panel.xyplot` to treat the graph as a simple scatterplot. And, in `panel.abline`, the “h = as.numeric(y)” instruction places a horizontal line at each distinct location along the vertical axis. The remaining arguments in the panel functions should be self-explanatory.

The preceding example may seem like a lot of work to accomplish an objective that could have been produced much more easily (i.e., without explicitly calling any panel functions at all). However, the example does illustrate the point made earlier, that R regards a dot plot as a scatterplot (enhanced by reference lines) between a factor and a variable. To see why this mode of thinking is useful, let us assume that we want to produce a dot plot of the same data, but with the reference lines omitted. This can be accomplished very easily by omitting `panel.abline` from the previous panel function:

```
> dotplot(occup2 ~ prestige, data = occ.prest.subset,
+ aspect = 1.5,
+ xlab = "Prestige score for occupation",
+ panel = function (x, y) {
+   panel.xyplot(x, as.numeric(y),
+     cex = 1.25,
```

```
+ col = "black",
+ lty = 2)})
```

Figure 12 shows the result. In effect, the dot plot is now just a scatterplot of the data. So, why not use the `xyplot` function to produce it, rather than following this seemingly indirect route of using `dotplot`? The reason is that `dotplot` automatically handles the details of labeling the vertical axis; it would require a bit more work to do this in `xyplot`.

Next, suppose that we want the reference lines in the dotplot to run only from the left side of the display to the plotted points, themselves.¹⁰ This requires adding another panel function, `panel.segments`, to the previous example:

```
> dotplot(occup2 ~ prestige, data = occ.prest.subset,
+ aspect = 1.5,
+ xlab = "Prestige score for occupation",
+ xlim = c(0, 100),
+ panel = function (x, y) {
+ panel.segments(rep(0, length(x)), as.numeric(y),
+ x, as.numeric(y),
+ col = "gray",
+ lty = 2)
+ panel.xyplot(x, y,
+ cex = 1.25,
+ col = "black",
+ lty = 2)})
```

Figure 13 shows the result of this function call. The first two arguments to `panel.segments` are vectors containing the horizontal and vertical coordinates of the starting positions for the line segments. The third and fourth arguments are vectors containing the coordinates of the terminal points for the line segments. The `rep` function creates a vector of zeroes. The size of this vector is equal to the number of values being plotted (i.e., “`length(x)`”). So, this vector contains the horizontal coordinate for the starting point of each observation’s line segment (i.e., they all begin at zero). The horizontal coordinate of the terminal point for each observation is simply the value of the variable being plotted (i.e., `x` in `panel.segments`). And, as before, the vertical coordinates for the points are supplied by “`as.numeric(y)`”.

Finally, the `xlim` argument (for “x limits”) operates in an obvious way to set the minimum and maximum values on the horizontal axis to zero and one hundred, respectively.

Creating Dot Plots for Comparisons

In the previous section, we examined several variants of a basic dot plot for displaying a set of data values. Let us next consider several dot plots for making comparisons of data subsets. Note that for these examples, I will not introduce any new datasets. Instead, I will list and explain the R functions that were used to produce some of the dot plots shown in the earlier sections of this article.

We will begin with a fairly easy example. Recall that Figure 3 showed a dot plot of state education expenditures in 1990 and 2000. That plot was generated by the following function call:

```
> dotplot(state ~ educ.1990, data = educ.spending,
+   aspect = 1.5,
+   xlim = c(600, 2600),
+   xlab = list("State education spending ($1000s per capita)",
+     cex = .75),
+   panel = function (x, y) {
+     panel.abline(h = as.numeric(y), lty = 2, col = "gray")
+     panel.xyplot(x, y, pch = 16, col = "black", cex = .75)
+     panel.xyplot(educ.spending$educ.2000,
+       y, pch = 4, col = "black", cex = .6)},
+   key = list(text = list(c("1990", "2000"), cex = .75),
+     points = list(pch = c(16, 4), col = "black", cex = .75),
+     space = "top", border = T),
+   scales = list(y = list(cex = .6)) )
```

The data for this plot are contained in a data frame called *educ.spending*. It contains three variables: *state* (a factor, with levels ordered by 1990 education expenditures), *educ.1990*, and *educ.2000*. Of course, the latter two variables contain each state’s expenditure values for the two years.¹¹ There are five features of primary interest in this call to `dotplot`. First, the label for the horizontal axis is now specified as a list. The first element in the list is the label, itself (i.e., the text that appears between the double-quotes). The second element uses

the `cex` argument (for “character expansion”) to make the font three-fourths of the default size; this is entirely optional and is done here for aesthetic purposes only.

Second, there are several functions used together within the `panel` function. The first (`panel.abline`) creates dashed gray lines at each state’s position on the vertical axis. The second function (`panel.xyplot`) plots the variables specified in the general display function. The `pch` argument sets the plotting character (to a solid circle, in this case) and `cex` makes the plotted symbols sixty percent of their default size. The third function (another `panel.xyplot`) plots points using the 2000 spending values as horizontal coordinates; once again the vertical axis coordinates are passed along from the general display function. Here, the `pch` argument specifies that the points should be plotted as x’s. Thus, the dot plot is effectively two distinct scatterplots located within a common plotting region.¹²

Note that `panel.abline` should appear before the two calls to `panel.xyplot`. The `panel` function in a trellis graph always adds elements to the display sequentially, in the order they are mentioned in the function call (this is called the “painter’s model” for constructing graphs). Generating the lines first guarantees that the data points will be drawn over them; a reverse ordering of the functions (i.e. `panel.xyplot` appearing before `panel.abline`) would cause the data points to be overwritten and partially hidden by the lines.

The third interesting feature of this function call is the specification of a separate key to explain the plotting symbols. The `key` argument provides a list to the `dotplot` function. The first two elements in this list (i.e., `text` and `points`) are, themselves, two-element vectors giving the plotting symbols and their identifying labels (they are each made three-fourths of the default size for aesthetic reasons). The third element in the list, the `space` argument, directs `dotplot` to place the key above the graphical display; Other possible specifications for this argument are `bottom`, `left`, and `right`; each of these would have the obvious effect on the placement of the key. The last element in the list, “`border = T`” is a logical condition, telling `dotplot` to print a border around the key (the default is to omit the border).

Fourth, the `scales` argument is used here to reduce the size of the labels printed along the vertical axis. This dot plot includes a fairly large number of labels ($n = 50$). If left at their default size, the state names would overlap and it would be very difficult to decode the information they provide. So, this specification is not merely for aesthetic purposes. Instead, it actually facilitates accurate perception of the information provided in the graphical display.

The fifth interesting feature in this example involves the order in which the arguments appear within the call to `dotplot`. The formula (i.e., “`state ~ educ.1990`”) must be the first argument to the function. Following that, the first few optional arguments (i.e., `data`, `aspect`, `xlim`, and `xlab`) are part of the general display function. These are followed by the panel function which, itself, is followed by two more arguments for the general display function (i.e., `key` and `scales`). Now, the exterior parts of the display must be set up before the data are plotted (e.g., the size of the plotting region cannot be determined until `dotplot` sets aside space for axes, labels, and the key). So, this particular ordering of the parameters may seem to violate the painter’s model described earlier (i.e., the panel function defining the plotting region appears before the general display function is completed). However, it works because R evaluates parameters separately in the general display function and the panel function. Specifically, the former is carried out completely before the panel function is started (analogous to the painter, who must set up a canvas and a palette of colors before starting on the content of a painting), regardless where the parameters appear in the call to `dotplot`; but, within each of the separate general display and panel functions, parameters are still evaluated sequentially.

Moving on to a more complex type of comparison plot, recall that Figure 5 showed 1992 state policy priority scores separately by region. Creating such divided plots using the `dotplot` function is a bit tricky and it involves as much work in preparing the dataset as it does in specifying the function.¹³ Let us begin by creating a slightly different-- and slightly easier-- version of the dot plot from Figure 3. Assume that the data are contained in four separate data frames called *northeast*, *midwest*, *south*, and *west*, respectively. Each data

frame has a single variable called *policy*. The two-letter USPS state name abbreviations are used as row names in each of the data frames and the observations are sorted by the values of *policy* within each data frame.

The first step is to create a single “divided” dataset in which the states from each region are grouped together and each region’s observations are separated by a “dummy” observation containing the region name and a nonsense value for *priority*. This can be accomplished as follows:

```
> label1 <- as.data.frame(-9)
> colnames(label1) <- "policy"
> rownames(label1) <- "Northeast:  "
> label2 <- as.data.frame(-9)
> colnames(label2) <- "policy"
> rownames(label2) <- "Midwest:   "
> label3 <- as.data.frame(-9)
> colnames(label3) <- "policy"
> rownames(label3) <- "South:    "
> label4 <- as.data.frame(-9)
> colnames(label4) <- "policy"
> rownames(label4) <- "West:     "
> new.data <- as.data.frame(rbind(northeast, label1,
+   midwest, label2, south, label3, west, label4))
```

Notice the strategy used in the preceding statements: For each region, a new data frame (called *label1*, *label2*, and so on) is created with a single observation and a single variable. The variable is called *policy* (just as in the data frames containing the real data), but the value is set to a nonsensical -9.¹⁴ The row name in the new data frame gives the region name, followed by a colon and three blank spaces. These extra spaces will be used to offset the region labels from the state abbreviations in the vertical axis labels of the graph. Finally, the `rbind` function (i.e., “row bind”) is used to “stack” the respective data frames so that each region label appears below the “real” data for that region; this creates a matrix which is, in turn, coerced to a data frame (using the `as.data.frame` function).

Next, we need to create an ordered factor from the row names in *new.data*. The levels of this factor will be arranged according to the order in which the observations occur in *new.data* (recall that the observations are already sorted by policy values within each region):

```
> new.data$state <- as.factor(row.names(new.data))
> new.data$sequence <- seq(1, length(new.data$policy))
> new.data$state <- reorder(new.data$state, new.data$sequence)
```

This new ordered factor is employed as the vertical axis variable in the dot plot, as follows:

```
> dotplot(state ~ policy, data = new.data,
+   aspect = 1.5,
+   xlab = "State policy priority scores, 1992",
+   xlim = c(.49, .55),
+   scales = list(y = list(cex = .6)),
+   panel = function (x, y) {
+     panel.dotplot(x[x > 0], y[x > 0],
+       pch = 16, col = "black", lty = 2)} )
```

The graph that results is shown in Figure 14. Most elements of the preceding function should be familiar to the reader. The only new technique is the use of logical conditions within the square brackets following the *x* and *y* variables specified in `panel.dotplot`. The panel function will only carry out the plotting tasks for those observations where the condition is true. Since the region labels were given values of -9 on the *policy* variable (i.e., *x* in `panel.dotplot`), the condition is false for those observations. Therefore, no horizontal lines or data points are plotted in those cases. Note that “pch=16” specifies filled circles as the plotting symbols.

The panel function only affects the interior of the plotting region. Therefore, all 52 levels of the factor, *state* (i.e., the 48 state names plus the four region names), still appear along the vertical axis (which is, of course, outside the plotting region). In this case, it is important to specify the `xlim` argument so that it just contains the legitimate values of the *policy* variable. Otherwise, the general display function for `dotplot` would regard the nonsense value, -9, as the minimum value of *policy* for purposes of constructing the horizontal axis.

The dot plot shown back in Figure 5 shows the same information as Figure 14. The only difference is somewhat “cosmetic” in that the region labels are displayed within the plotting region, rather than the left margin of the vertical axis. To reproduce this, we will need to replace the region labels with blank spaces on the vertical axis and explicitly insert the region names into the plotting region. Again, this is a bit tricky because of the way that R handles row names (every row name in a data frame must be unique, so multiple occurrences of blanks for row names will not work) and factor levels (`dotplot` graphs data values against unique levels of a factor so, once again, multiple blank spaces will not work because they would be regarded as a single level of the factor). What we will do is employ the same data frame from the previous example, but create indicator vectors to determine where data points and region names should be plotted within the plotting region. In addition, we will explicitly supply a new vector of values for the vertical axis labels and a vector containing the four region labels:

```
> has.values <- which(new.data$policy != -9)
  > no.values <- which(new.data$policy == -9)
  > state2 <- as.vector(new.data$state)
  > state2[no.values] <- " "
  > region.labels <- c("Northeast:", "Midwest:", "South:", "West:")
```

In the first two statements, the `which` function creates two vectors giving the location indices of the observations where *policy* is equal to -9 and not equal to -9, respectively. The third statement creates a new vector, *state2*, with values identical to the *state* variable in data frame *new.data*. The fourth statement uses the newly-created *no.values* vector to set four of the entries in *state2* to blank spaces. Finally, the last statement simply creates a vector of region labels. The preceding statements are followed by the following call to `dotplot`, in order to produce that graph that was shown in Figure 5:

```
> dotplot(state ~ policy, data = new.data,
+   aspect = 1.5,
+   xlim = c(.49, .55),
+   scales = list(y = list(cex = .6,
```

```

+     labels = state2)),
+   xlab = list("State policy priorities, 1992", cex = .75),
+   panel = function (x, y) {
+     panel.dotplot(x[has.values], y[has.values],
+       col = "black", lty = 2)
+     panel.text(rep(.502, 4), y[no.values],
+       labels = region.labels, cex = .6, adj = 1)} )

```

Now, the preceding `dotplot` call uses the factor *state* from the *new.data* data frame to construct the dot plot. But, the “`labels = state2`” entry in the list supplied to the `scales` argument explicitly substitutes the entries of that vector for the levels of the *state* factor in the vertical axis labels. This is how we get the blanks in the rows where the region names were found in Figure 14.

In the call to `panel.dotplot`, the index vector, `has.values`, is used as a logical condition on the `x` and `y` variables to determine where reference lines are drawn and points are plotted in the plotting region of the display. The call to `panel.text` plots the region labels. The first argument in this function is a vector of four elements, all equal to 0.502; these are the horizontal coordinates for the region labels. The vertical coordinates are the four levels of the *state* factor indexed by the *no.values* vector. At these locations, the four entries in the *region.labels* vector are plotted. The `adj` argument is a number between zero and one, inclusive, that specifies the proportion of the label that should appear to the left of the plotting location. Setting “`adj=1`” is equivalent to right-justifying the labels.

Dot Plots and Statistical Summaries

As shown earlier, dot plots can be very useful for presenting the results of statistical analyses. For example, Figure 6 showed the percentage difference in Bush and Kerry votes across the party identification categories. Assume that these data are contained in a data frame called *party.vote*. This data frame is taken from a crosstabulation.¹⁵ It consists of seven rows (corresponding to the seven categories of the party identification index, given in order from “strong Democrat” to “strong Republican”) and three columns (containing the

partisan labels, the Bush vote, and the Kerry vote, respectively). The columns of the data frame are named *partyid*, *bush*, and *kerry*, respectively. The following R statements would reproduce Figure 6:

```
> party.vote$partyid <- reorder(party.vote$partyid, seq(1:7))
> dotplot(partyid ~ (bush - kerry), data = party.vote,
+   aspect = 1.5,
+   xlab = list("Percent voting for Bush minus percent voting for Kerry",
+   cex = .75),
+   panel = function (x, y) {
+   panel.xyplot(x, y, pch = 16, cex = 1.25, col = "black")
+   panel.abline(v = 0, lty = 2, col = "gray")} )
```

The first statement is used to fix the order of the party identification categories, and prevent them from being listed in alphabetical order in the dot plot. Note that the call to `dotplot` specifies that the *difference* between two variables is to be plotted. This eliminates the need to create a new variable for this purpose. In the panel function, we use `panel.xyplot` rather than `panel.dotplot` in order to eliminate the horizontal reference lines. The `pch` and `cex` arguments specify solid circles for the plotting symbols, one and one-fourth times the default size. And, the argument “`v = 0`” in `panel.abline` places a vertical, dashed, gray line at the zero point on the horizontal axis.

R makes it very easy to pass information from a statistical analysis over to a graphing function, with a minimum of manual copying or cutting-and-pasting. For example, Figure 7 showed a dot plot of ten sample means with error bars representing confidence intervals. Assume that the data used to calculate these means are contained in a data frame, *import.2004*, with 1212 rows (i.e., the sample size for the 2004 NES) and ten columns (one for each of the importance ratings). The graphing task is complicated a bit by the presence of missing values (coded as NA’s) within the data. Descriptive labels for the ten variables in *import.2004* are contained in the separate vector, *var.labels*. The means and confidence intervals for the dot plot are created with the following statements:

```

> sample.means <- mean(import.2004, na.rm = T)
> std.devs <- sd(import.2004, na.rm = T)
> where.nonmissing <- !(apply(import.2004, c(1, 2), is.na))
> sample.ns <- apply(where.nonmissing, 2, sum)
> std.errs <- std.devs / (sample.ns ^ .5)
> lower <- sample.means + (std.errs * qt(.025, (sample.ns - 1)))
> upper <- sample.means + (std.errs * qt(.975, (sample.ns - 1)))
> new.data <- data.frame(var.labels, sample.means, lower, upper)
> new.data$var.labels <- reorder(new.data$var.labels,
+   new.data$sample.means)

```

The first two statements use R's statistical functions to calculate the column means and standard deviations from data frame *import.2004*; in each case, the result is a ten-element vector. The next two statements determine the number of nonmissing observations within each column. The logical matrix, *where.nonmissing*, is the same size as *import.2004*. It has value TRUE in the cells that correspond to nonmissing data in *import.2004*, and FALSE in the cells that correspond to missing data. The ten-element *sample.ns* vector is then created by summing within columns of *where.nonmissing* (note that TRUE evaluates to one and FALSE to zero in the `sum` function). The *std.errs* vector contains the standard errors of the sample means, created by dividing the elements of the *std.devs* vector by the square roots of the elements in the *sample.ns* vector. The lower and upper bounds of the confidence intervals for the respective means are obtained by adding the product of the standard errors and the appropriate *t* values (obtained using the `qt` function) to the means. Next, the `data.frame` function concatenates the vectors of variable labels, sample means, and the limits of the confidence intervals as columns of a new data frame, called *new.data*. Finally, the levels of the factor *var.labels* are ordered according to the sample means, using the `reorder` function. The display in Figure 7 would be reproduced by the following call to `dotplot`:

```

> dotplot(var.label ~ sample.means, data = new.data,
+   aspect = 1.5,
+   xlim = c(3.3, 4.3),
+   xlab = "Mean importance rating",
+   panel = function (x, y) {
+     panel.xyplot(x, y, pch = 16, col = "black")

```

```
+   panel.segments(new.data$lower, as.numeric(y),
+   new.data$upper, as.numeric(y), lty = 1, col = "black")} )
```

Once again, we use `panel.xyplot` rather than `panel.dotplot` in order to eliminate the horizontal reference lines. The `panel.segments` function draws the error bars using the variables *lower* and *upper* as the horizontal coordinates for the ends of the line segments. Note that the fully-qualified variable names must be specified, since the general display function does not pass the name of the data frame from the `data` argument to the panel function.

It is also very straightforward to create a dot plot from the parameter estimates in a statistical model, such as a regression equation. Recall that Figure 8 showed standardized OLS coefficients, with different plotting symbols used to show which independent variables have significant and nonsignificant effects. Assume that this regression model is stored in an R object called “`model1`”.¹⁶ Now, the component parts of this (or any) model, and the parts of the summary of this model (i.e., the output obtained by calling the function `summary(model1)`) are both lists. Each element in an R list has a name, so we can access any part of the model or its summary by simply using that part’s name. For example, “`summary(model1)$coefficients`” would refer to the element of the *summary(model1)* list named *coefficients*.

The *coefficients* list element is a matrix in which the rows correspond to the regressors in the equation (beginning with the intercept, and following in the order that the variables were listed in the R formula that created the model). The matrix has four columns: The first contains the regression coefficients, the second contains the standard errors, the third holds the *t* statistics for the null hypothesis that each parameter is zero, and the fourth column contains the observed probability values for two-sided tests of that null hypothesis on each coefficient. The variable names are used as the row names for this matrix (“(Intercept)” is the name for the first row). Since these names are sometimes a bit cryptic, we will again

assume that descriptive variable labels are included in a factor called *var.labels*. We can create the data frame that will be used to reproduce Figure 8 with the following statements:

```
> coeff.info <-  
+   summary(model1)$coefficients[2:model1$rank, c(1,4)]  
> coeff.data <- data.frame(var.labels, coeff.info)  
> colnames(coeff.data) <- c("var.label", "coeff", "prob")  
> coeff.data$var.label <- reorder(coeff.data$var.label,  
+   coeff.data$coeff)
```

The first statement creates a matrix of the coefficients and the observed probability values by extracting rows and columns from the *coefficients* entry in the *summary(model1)* list.¹⁷ The first row of the *coefficients* matrix is not used because it contains the intercept (which is zero, by definition, with standardized coefficients). The “*model1\$rank*” specification extracts the *rank* element from the *model1* list.¹⁸ This is the rank of the crossproducts matrix for the independent variables; in the absence of perfect collinearity, this is equal to the number of regressors. The vector “*c(1, 4)*” extracts the first and fourth columns from the “coefficients” matrix. The second statement creates a new data frame by concatenating the vector of variable labels and the newly-created matrix of coefficients and probabilities. The third statement sets the column names in this data frame (the names picked up from the model summary are longer and a bit clumsy to use), and the fourth statement reorders the levels of the factor containing the variable labels so that they correspond to the sizes of the coefficients. Figure 8 then could be reproduced by the following call to *dotplot*:

```
> dotplot(var.label ~ coeff, data = coeff.data,  
+   aspect = 1.5,  
+   xlab = list("Standardized regression coefficient",  
+   cex = .75),  
+   scales = list(cex = .75),  
+   panel = function(x, y) {  
+     panel.xyplot(x[coeff.data$prob < .05],  
+       y[coeff.data$prob < .05],  
+       cex = 1.25, pch = 16, col = "black")  
+     panel.xyplot(x[coeff.data$prob >= .05],  
+       y[coeff.data$prob >= .05],
```

```

+     cex = 1.25, pch = 1, col = "black")
+   panel.abline(v = 0, lty = 2, col = "gray") },
+   key = list(text = list(c("      Significant, 0.05 level:",
+ "Not significant, 0.05 level:"), cex = .75),
+   points = list(pch = c(16, 1), col = "black", cex = .75),
+   space = "top", border = T) )

```

This function combines a number of the techniques introduced earlier; while they are perhaps used a bit differently, there is really nothing new here. First, we specify the formula, the dataset, the aspect ratio, the horizontal axis label, and the reduced size for the axis tick labels. The `panel` function contains two different calls to `panel.xyplot` (again, we use these to eliminate the horizontal reference lines that `panel.dotplot` would create). In each one, logical conditions are used to specify different plotting symbols for significant (i.e., “`coeff.data$prob < .05`”) and nonsignificant (i.e., “`coeff.data$prob >= .05`”) coefficients. The `panel.abline` function creates the vertical reference line and, finally, a key for the plotting symbols is placed at the top of the display.

Further Resources

Hopefully, the examples presented above will help readers use the R statistical computing environment in order to generate not only basic dot plots, but also more complex versions of this graphical display. I have tried to supplement the listings of R statements with more general insights regarding the nature and operation of trellis graphics functions. I hope that this information will enable researchers to create dot plots that are appropriate for their own particular needs and data analysis contexts. At the same time, many of the programming “tricks” discussed above should also prove useful for modifying the default parameters in the R functions that create other kinds of graphical displays (e.g., the `xyplot` function to create scatterplots in the `lattice` package).

As with any set of specific examples, those provided in this article only scratch the surface of a potentially vast subject. For this reason, further documentation about the graphical tools available in R is generally very desirable. The most convenient source of information

is the R online help system. However, many users find the help files for `lattice` functions to be a bit terse. As a more user-friendly alternative, the *S-Plus Trellis Graphics User's Manual* is an excellent guide to the entire trellis system and its general usage. Another extremely helpful source of information is “A Tour of Trellis Graphics,” by Richard A. Becker, William S. Cleveland, Ming-Jen Shyu, and Stephen P. Kaluzny. This paper expands upon the basic information provided in the *User's Manual* and provides detailed examples illustrating how to create and modify trellis graphs. While these documents were written for the commercially-available S-Plus system, virtually all of their content applies directly to the `lattice` package in R, as well. Both are available on the Trellis Display web site:

<http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/>

I have also included copies of these two documents on my own web site. Still another source of information is the book, *R Graphics*, by Paul Murrell. This is a general reference work which provides comprehensive and readily-accessible treatment of both the traditional and the grid (which contains the `lattice` package) graphics systems in R. Finally, John Fox's book, *An R and S-Plus Companion to Applied Regression*, provides an enormous amount of information and advice about working with the statistical and graphics functions in R. I rely on all of these sources in my own work, and I recommend them very highly.

CONCLUSIONS

In conclusion, dot plots are excellent graphical displays for labeled quantitative data values. They contain a great deal of information, are easy to interpret, and overcome a number of the problems associated with other kinds of displays. Dot plots are also extremely flexible; they can be modified in various ways to handle many different data analysis situations. They are useful both for analytic purposes (to paraphrase Tukey, they show the researcher features that he/she never expected to see) and for presentational displays (i.e., they guide the observer see the researcher's conceptions of the most important features in the data). While a number of software packages contain routines for producing dot plots, the R statistical

computing environment is unparalleled in its ability to modify and adapt this graphical display to a wide variety of research contexts. For all of these reasons, dot plots (along with the requisite programming knowledge to create them) constitute a very useful addition to the methodologist's "toolbox."

NOTES

1. Dot plots are sometimes called “index plots” (e.g., Fox 2002).
2. Of course, this structure could be reversed, by placing the labels on the horizontal axis and the quantitative values on the vertical axis. Fox (2002) presents several examples showing this kind of dot plot (pages 30, 196, 198). Nevertheless, categorical labels are usually easier to read when they are placed along the vertical axis. And, it is usually possible to include a larger number of labels than would be the case on the horizontal axis.
3. Open, rather than solid, circles are used as the plotting symbol for the 1990 expenditures because the 1990 and 2000 values for Maine are almost identical. Therefore, they overlap in the dot plot and a solid plotting symbol would hide the “x”.
4. The examples in this article were produced using R 2.3.0. With earlier versions of R, some modifications would be necessary in order to make the examples work properly.
5. All of the datasets used in the examples are available on my website, along with detailed R scripts to reproduce all of the figures from this article. The web site also includes alternative versions of many graphs and a number of additional dot plot examples. The URL is <http://polisci.msu.edu/jacoby/>.
6. The contents of the formula could be reversed (i.e., with “x” placed before the tilde and “y” afterward) to produce a dot plot with labels on the horizontal axis and variable values on the vertical axis. However, several other modifications to the `dotplot` call usually would be necessary in order to produce legible labels. An example of such a “horizontal” dot plot is given on my website.
7. Strictly speaking, the `reorder` function sorts the factor’s levels by the values of a function applied to X , within unique levels of *factor*. The default function in `reorder` is the sample mean.
8. We could use the `attach` function to add *occ.prest.subset* to R’s search path. This might save a bit of typing. However, we would still have to add any new variables or modified versions of existing variables to the data frame. So, fully-qualified names would still need to be used in such cases.
9. Figuring out the various arguments, defaults, and optional parameters for R graphics can be an extremely frustrating experience. But, there are a few ways to get some help with this process. The `show.settings()` function displays the plotting characteristics (e.g., line styles, plotting symbols, colors, etc.) that are currently in effect for trellis displays. Issuing the statement “`trellis.par.get()`” will return a comprehensive list of default values for the parameters used to create trellis displays. The function `trellis.par.set` can be used

to modify specific parameters. Fox (2002, pages 239-242) shows how to create some very handy graphs for keeping track of plotting symbols and line types.

10. As explained earlier this line style is only appropriate for ratio-level data, where magnitude comparisons are appropriate. But, the occupational prestige scores are actually ratio-level values, even if they are not usually interpreted as such: In fact, the scores are the percentages of survey respondents that rated each occupation as “good” or “excellent” on a five-point scale. So, it should be reasonable to extend the lines from the zero point to the data values in this example.
11. Figure 5 could be reproduced by reordering the levels of *state* according to the values of *educ.2000*, and then issuing exactly the same call to `dotplot`.
12. Although I prefer to use the panel functions as shown here, an alternative approach would be to use the `groups` argument in the `dotplot` call. Doing so would produce identical results. One possible advantage of this approach is that the `simpleKey` function can be used as a slightly easier way to create a key for the graph. See my web site for an example, Becker et al. (1996) for more details on the `groups` argument, and the `lattice` help files for `simpleKey` for more information on that function.
13. Divided dot plots may comprise one situation where it is easier to use the `dotchart` function from the traditional R graphics system in order to obtain the desired results. I provide an example on my website and the help file for `dotchart` gives more details.
14. Of course, there is nothing special about -9. Any other value could have been used, as long as it would not be mistaken for a legitimate data value. We also could have assigned the R missing value, NA. But comparisons involving missing values are a bit tricky in R, so I find it easier to avoid any complications by simply specifying a numeric, but nonsensical, value in situations that involve logical comparisons.
15. The contents of a crosstabulation can be used directly as input to the `dotplot` function. My website provides an example, and also shows how Figure 6 could be constructed using this approach. More details are also provided in the R help file for `dotplot.table`.
16. In this case, the standardized coefficients were obtained by standardizing the data (using R’s `scale` function) before estimating the model.
17. Of course, it is not really appropriate to use inferential procedures with standardized coefficients. But, the probability values are identical to those that are obtained from the nonstandardized coefficients, so no harm is done.
18. The same *rank* object is also an element in the `summary(model1)` list. While that list could have been used here, the reference to `model1` reduces the line length a bit.

REFERENCES

- Baker, Andy. (2005) "Who Wants to Globalize? Consumer Tastes and Labor Markets in a Theory of Trade Policy Beliefs." *American Journal of Political Science* 49: 924-938.
- Becker, Richard A.; William S. Cleveland; David A. James. (1996) S-Plus Trellis Graphics User's Manual (Trellis Versions 2.0 & 2.1). Seattle, WA: MathSoft, Inc.
- Becker, Richard A.; William S. Cleveland; Ming-Jen Shyu; Stephen P. Kaluzny. (1996) "A Tour of Trellis Graphics." Unpublished manuscript.
- Cleveland, William S. (1984) "Graphical Methods for Data Presentation: Full Scale Breaks, Dot Charts, and Multibased Logging." *The American Statistician* 38: 270-280.
- Cleveland, William S. (1993) *Visualizing Data*. Summit, NJ: Hobart Press.
- Cleveland, William S. (1994) *The Elements of Graphing Data (Revised Edition)*. Summit, NJ: Hobart Press.
- Cleveland, William S. and Robert McGill. (1984) "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods." *Journal of the American Statistical Association* 79: 531-553.
- Cleveland, William S. and Robert McGill. (1985) "Graphical Perception and Graphical Methods for Analyzing and Presenting Scientific Data." *Science* 229: 828-833.
- Clinton, Joshua; Simon Jackman; Douglas Rivers. (2004) "The Statistical Analysis of Roll Call Data." *American Political Science Review* 98: 355-370.
- Duncan, Otis Dudley. (1961) "A Socioeconomic Index for All Occupations." In Reiss, Jr., A. J. (Editor) *Occupation and Social Status*. New York: Free Press.
- Fox, John. (2002) *An R and S-Plus Companion to Applied Regression*. Thousand Oaks, CA: Sage.
- Friendly, Michael. (1991) *SAS System for Statistical Graphics*. Cary, NC: SAS Institute.
- Jacoby, William G. (2005) "Is It Really Ambivalence? Public Opinion Toward Government Spending." In Stephen C. Craig and Michael D. Martinez (Editors) *Ambivalence and the Structure of Political Opinion*. New York: Palgrave Macmillan.
- Jacoby, William G. (2006) "Value Choices in American Public Opinion." *American Journal of Political Science* 50: 706-723 (Forthcoming).
- Jacoby, William G. and Sandra K. Schneider. (2001) "Variability in State Policy Priorities: An Empirical Analysis." *Journal of Politics* 63: 544-568.
- Jerit, Jennifer; Jason Barabas; Toby Bolsen. (2006) "Citizens, Knowledge, and the Information Environment." *American Journal of Political Science* 50: 266-282.

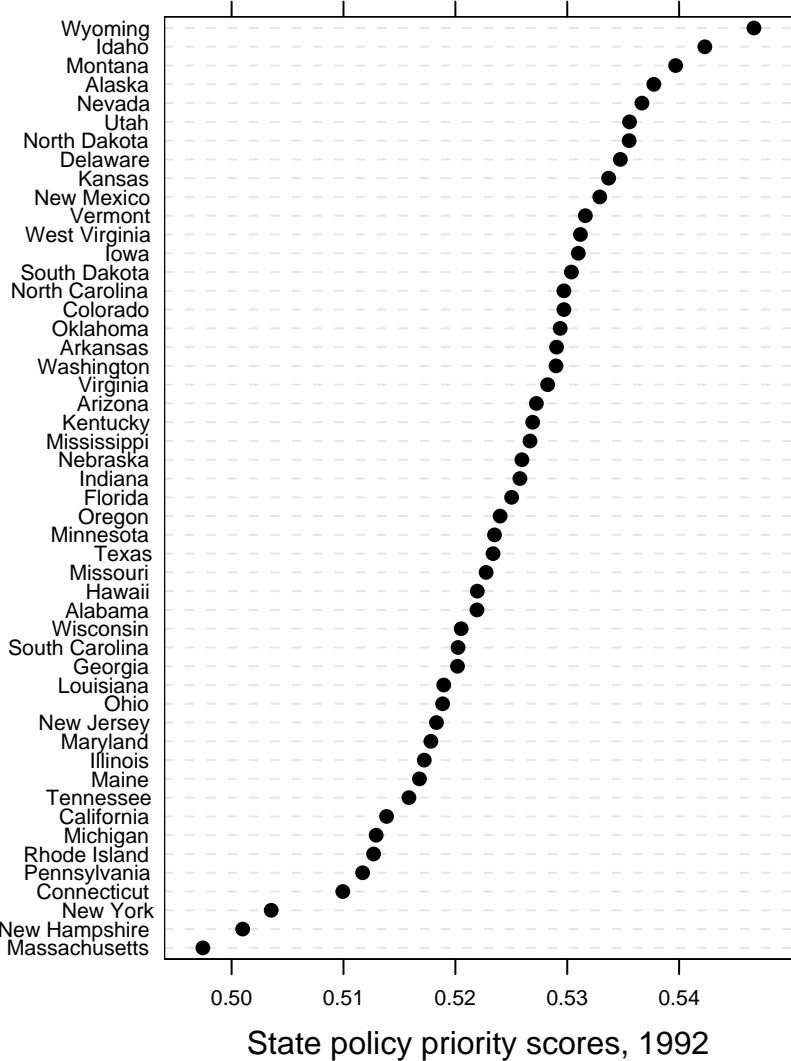
- Keith, Bruce E.; David B. Magleby; Candice J. Nelson; Elizabeth Orr; Mark C. Westlye; Raymond E. Wolfinger. (1992) *The Myth of the Independent Voter*. Berkeley, CA: University of California Press.
- Murrell, Paul. (2006) *R Graphics*. Boca Rotan, FL: Chapman & Hall/CRC.
- Petrocik, John R. (1974) "An Analysis of Intransitivities in the Index of Party Identification." *Political Methodology* 1: 31-47.
- R Development Core Team (2006). "R: A Language and Environment for Statistical Computing." Vienna, Austria: R Foundation for Statistical Computing.
- Sarkar, Deepayan. (2006). "lattice: Lattice Graphics." R Package, Version 0.13-8.
- Tufte, Edward R. (1983) *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

Table 1: Prestige Scores for 15 Occupations.

Occupation	Prestige Score
Accountant	82
Author	76
Professor	93
Civil Engineer	88
Physician	97
RR Conductor	38
Store Manager	45
Mail Carrier	34
Carpenter	33
Machinist	57
Gas Station Attendant	10
Taxi Driver	10
Barber	20
Cook	16
Janitor	8

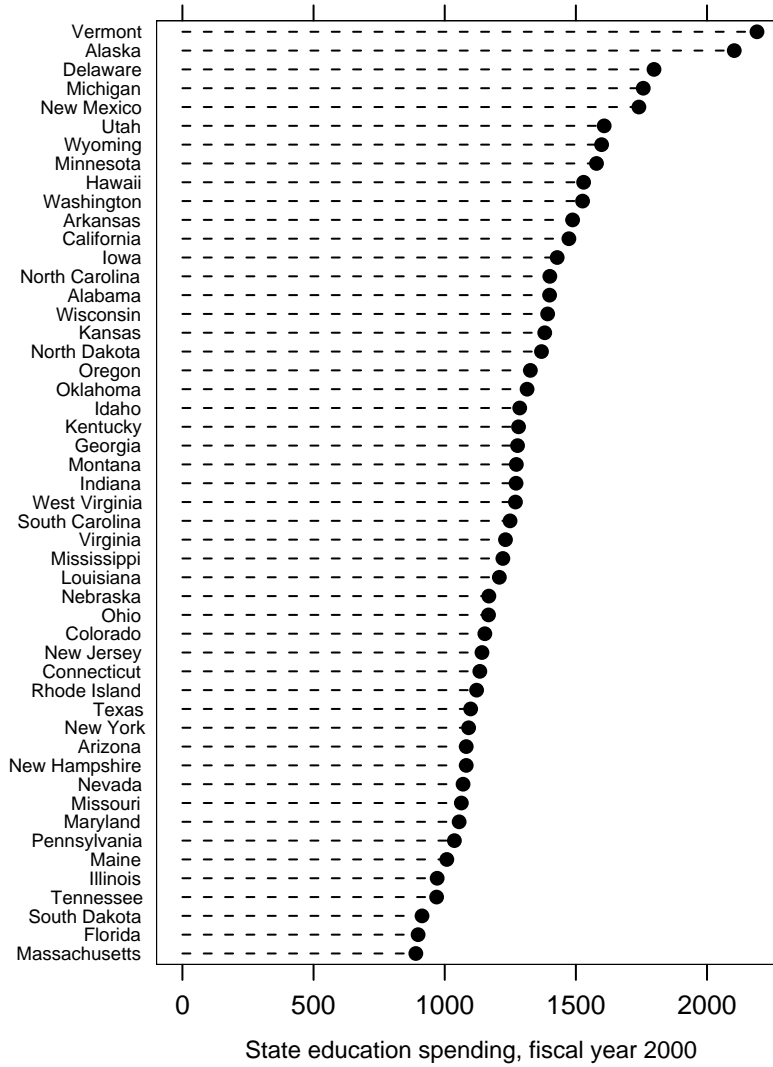
Data Source: Duncan (1961).

Figure 1: Dot Plot of 1992 State Policy Priority Scores.



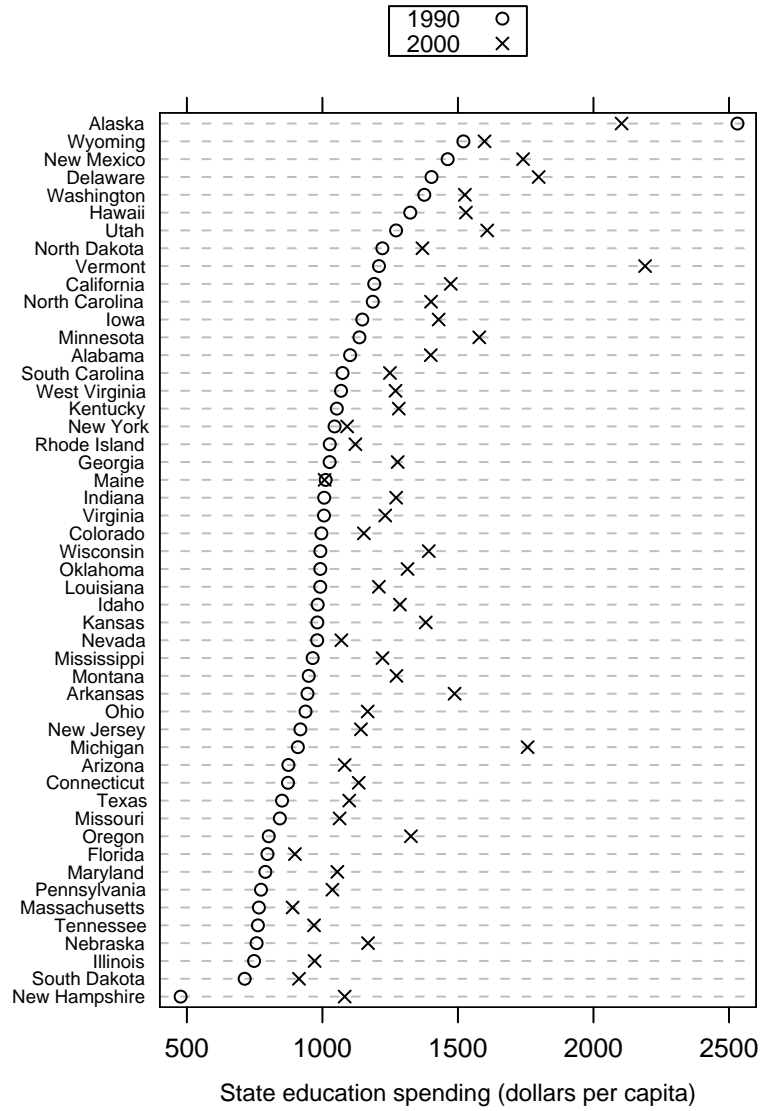
Data Source: Jacoby and Schneider (2001).

Figure 2: Dot Plot of State Education Spending, 2000 (in dollars per capita).



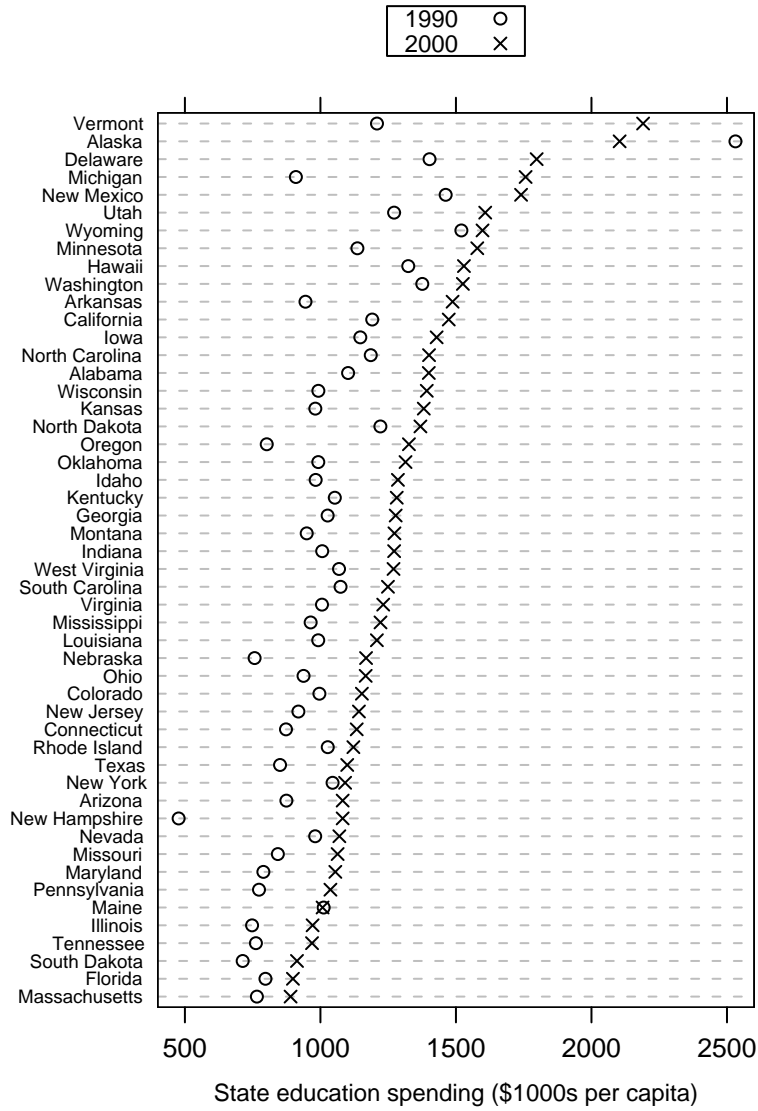
Data Source: *State Government Finances.*

Figure 3: Dot Plot of State Education Spending, 1990 and 2000 (in inflation-adjusted dollars per capita).



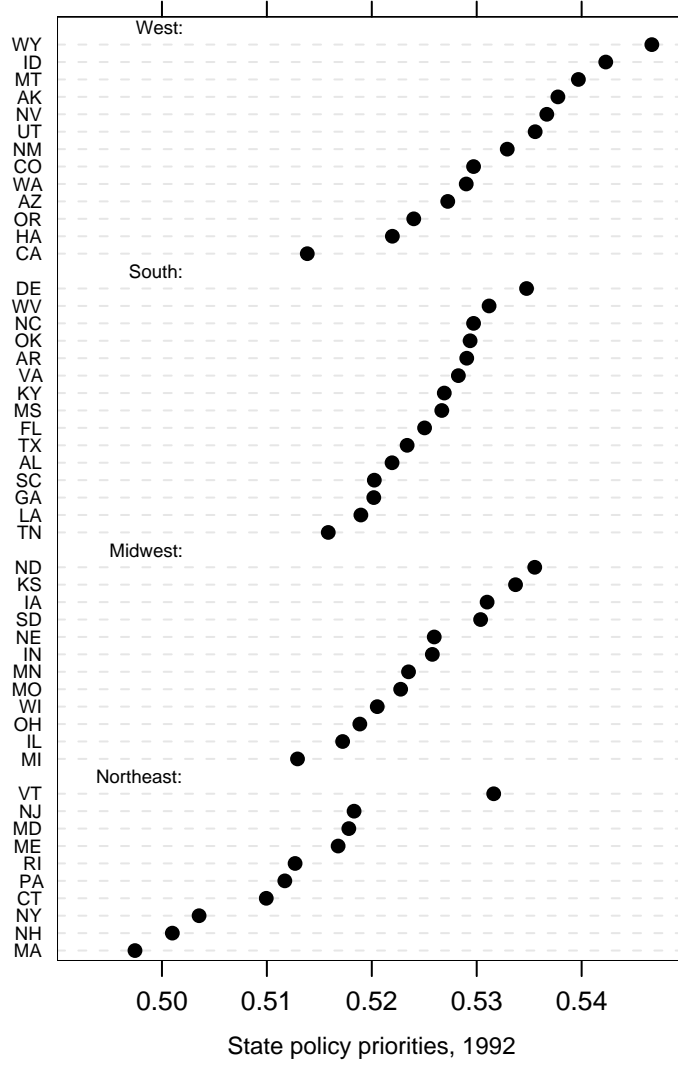
Data Source: *State Government Finances.*

Figure 4: Dot Plot of State Education Spending, 1990 and 2000 (in inflation-adjusted dollars per capita), Sorted by 2000 Expenditures.



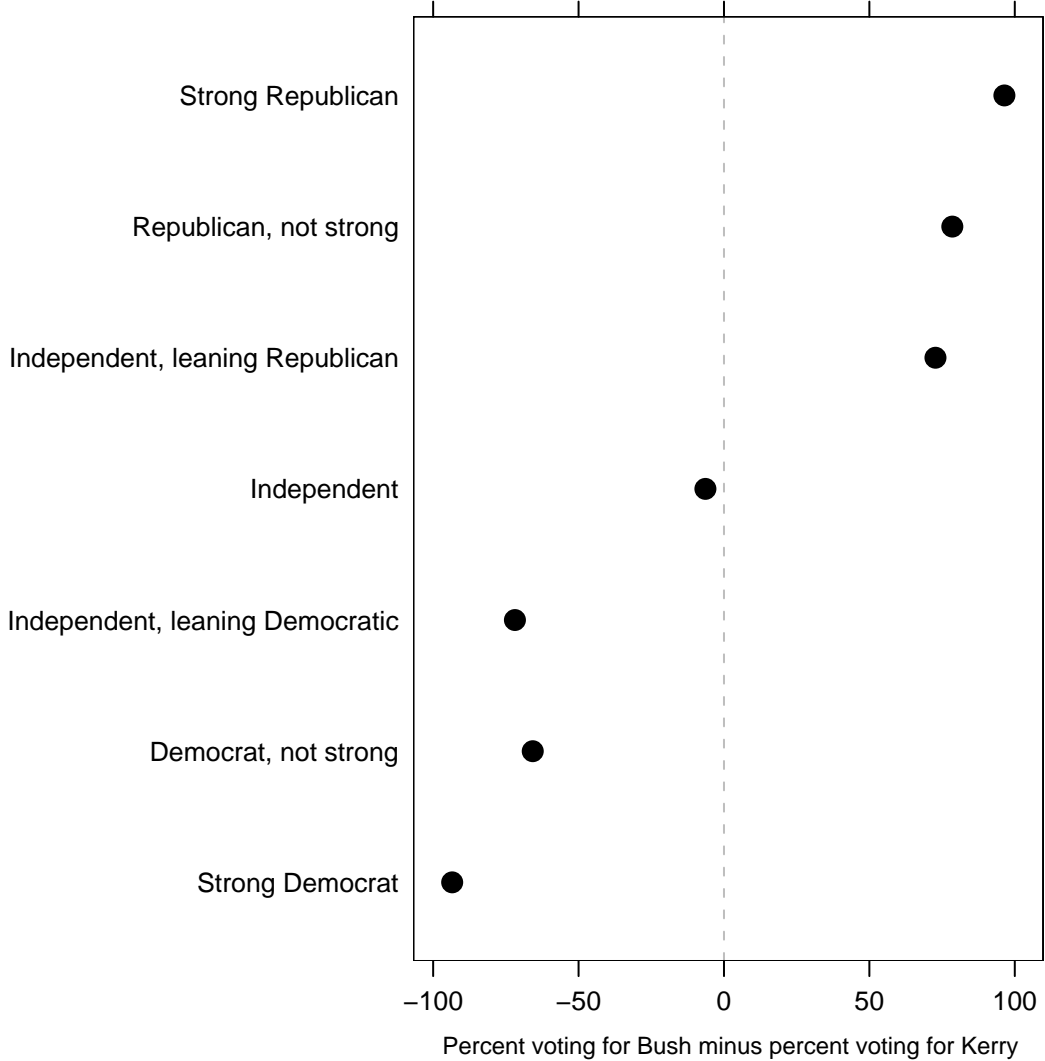
Data Source: *State Government Finances.*

Figure 5: Dot Plot of 1992 State Policy Priority Scores, by Region.



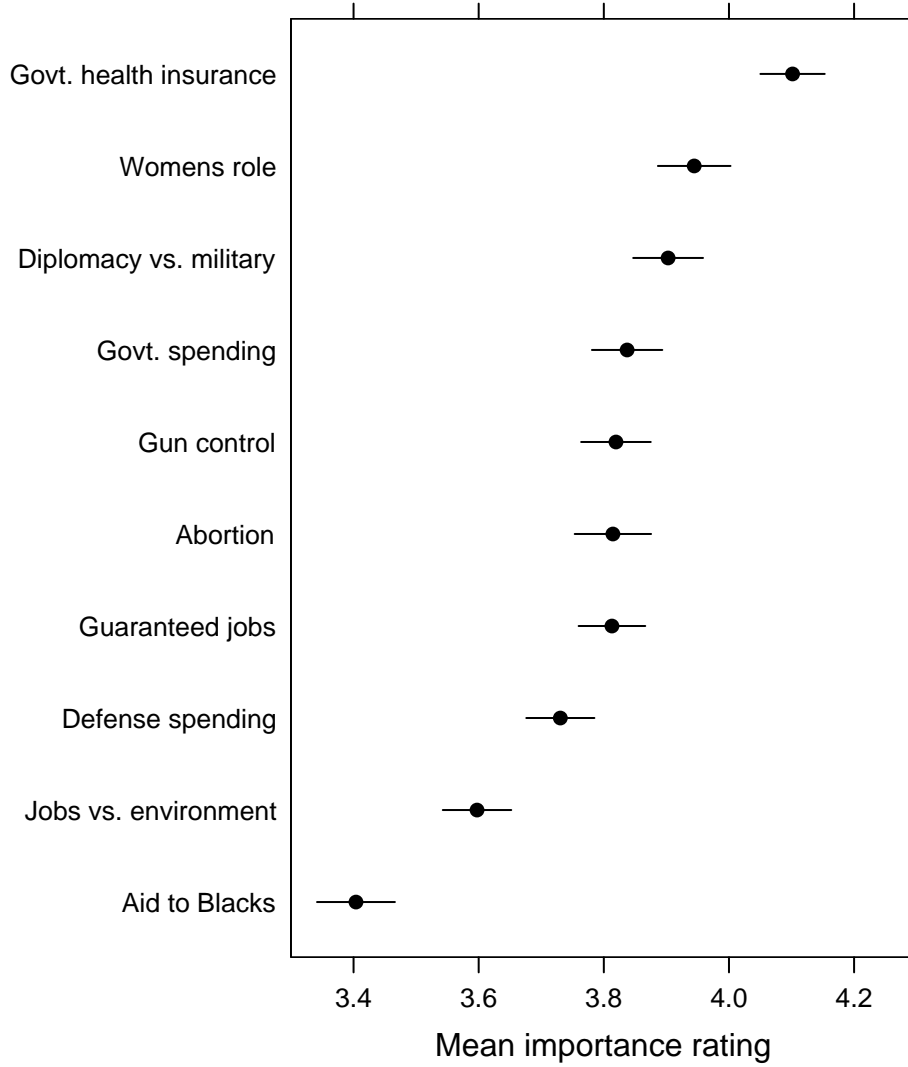
Data Source: Jacoby and Schneider (2001).

Figure 6: Dot Plot of Net 2004 Presidential Vote, by Party Identification



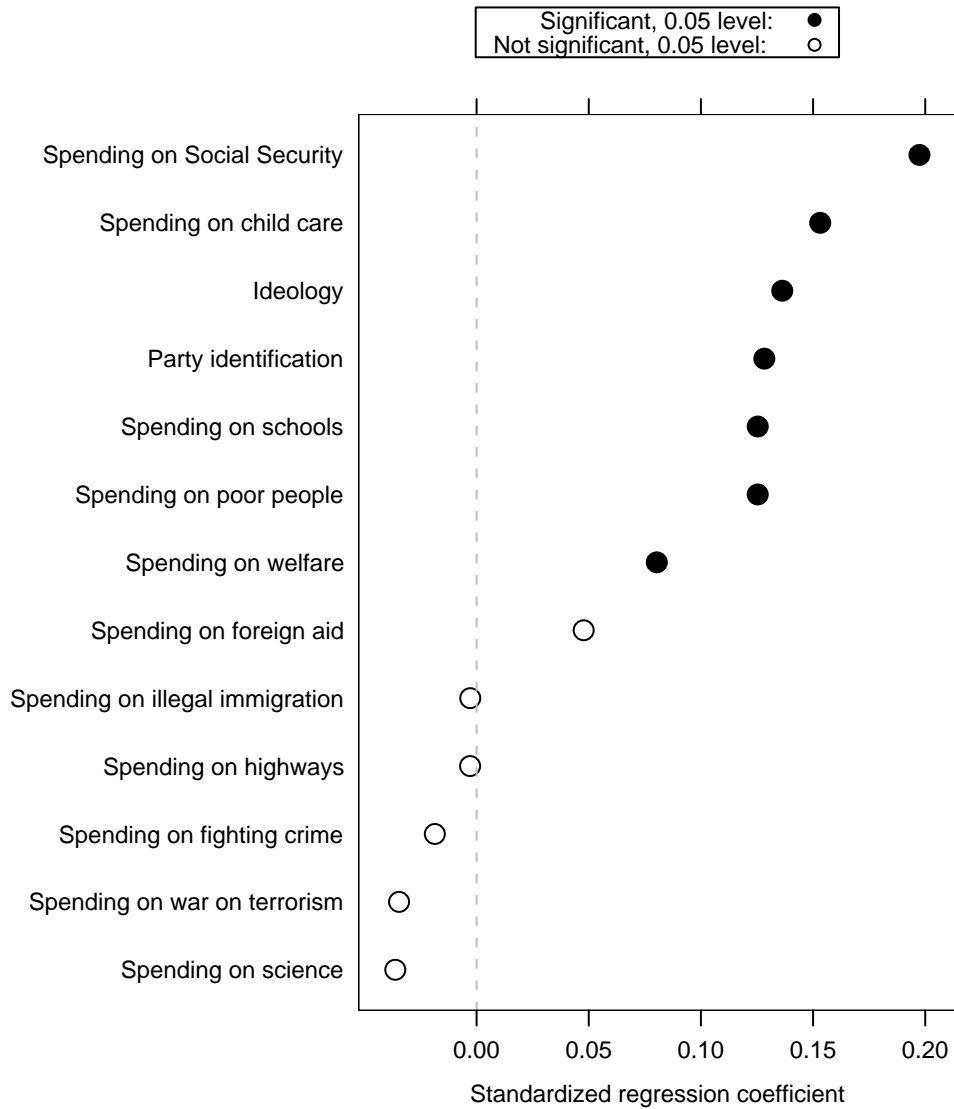
Data Source: 2004 CPS National Election Study.

Figure 7: Dot Plot of Mean Issue Importance Ratings, 2004 CPS National Election Study.



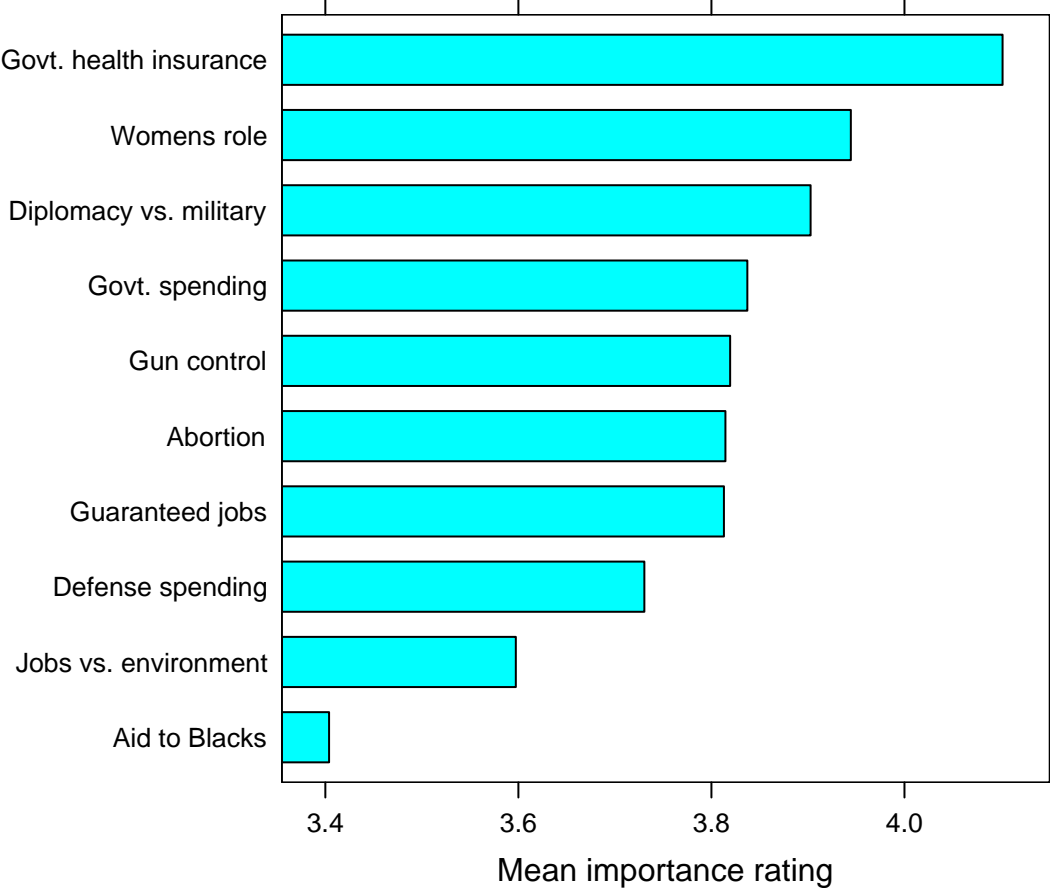
Note: Error Bars Represent 95% Confidence Intervals Around the Respective Mean Values.

Figure 8: Dot Plot of Standardized Regression Coefficients from Model Predicting Attitudes Toward Government Spending



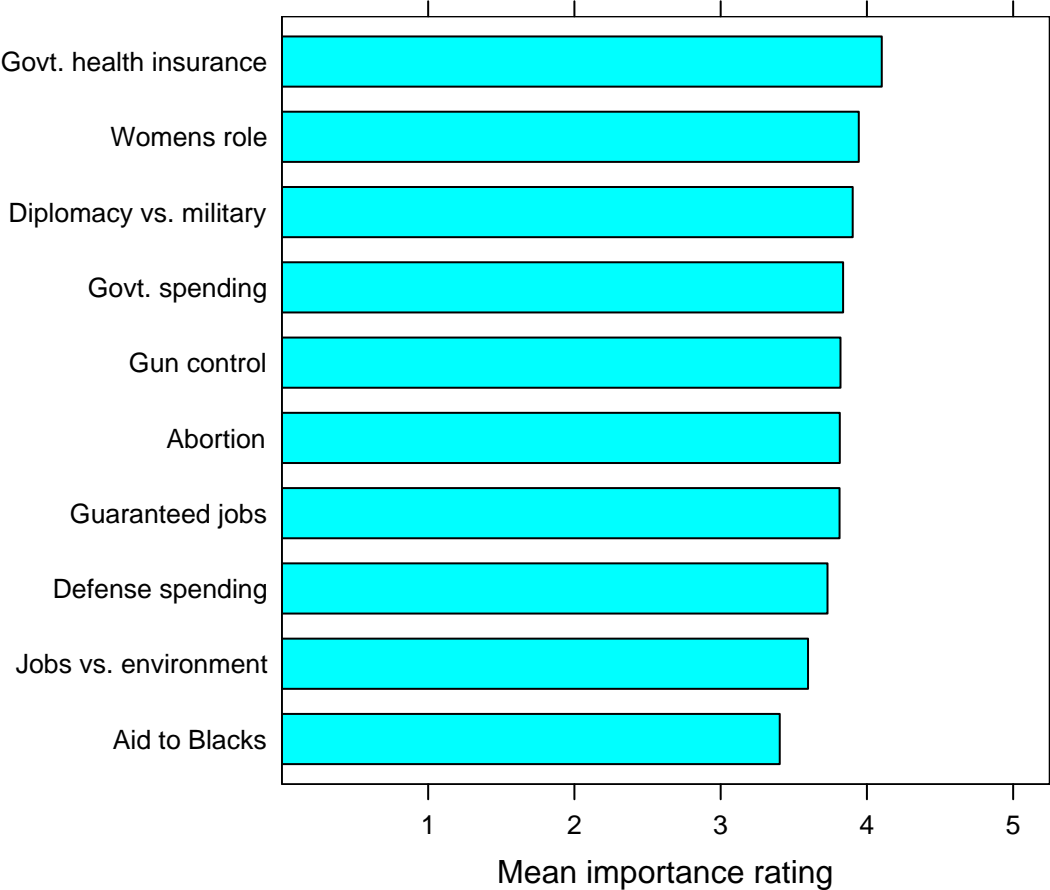
Note: Data source is the 2004 CPS National Election Study. Solid black plotting symbols indicate coefficients that are significantly different from zero in a two-sided test at the 0.05 level. Open-circle plotting symbols indicate coefficients that are not significantly different from zero.

Figure 9A: Bar Chart of Mean Issue Importance Ratings, 2004 CPS National Election Study.



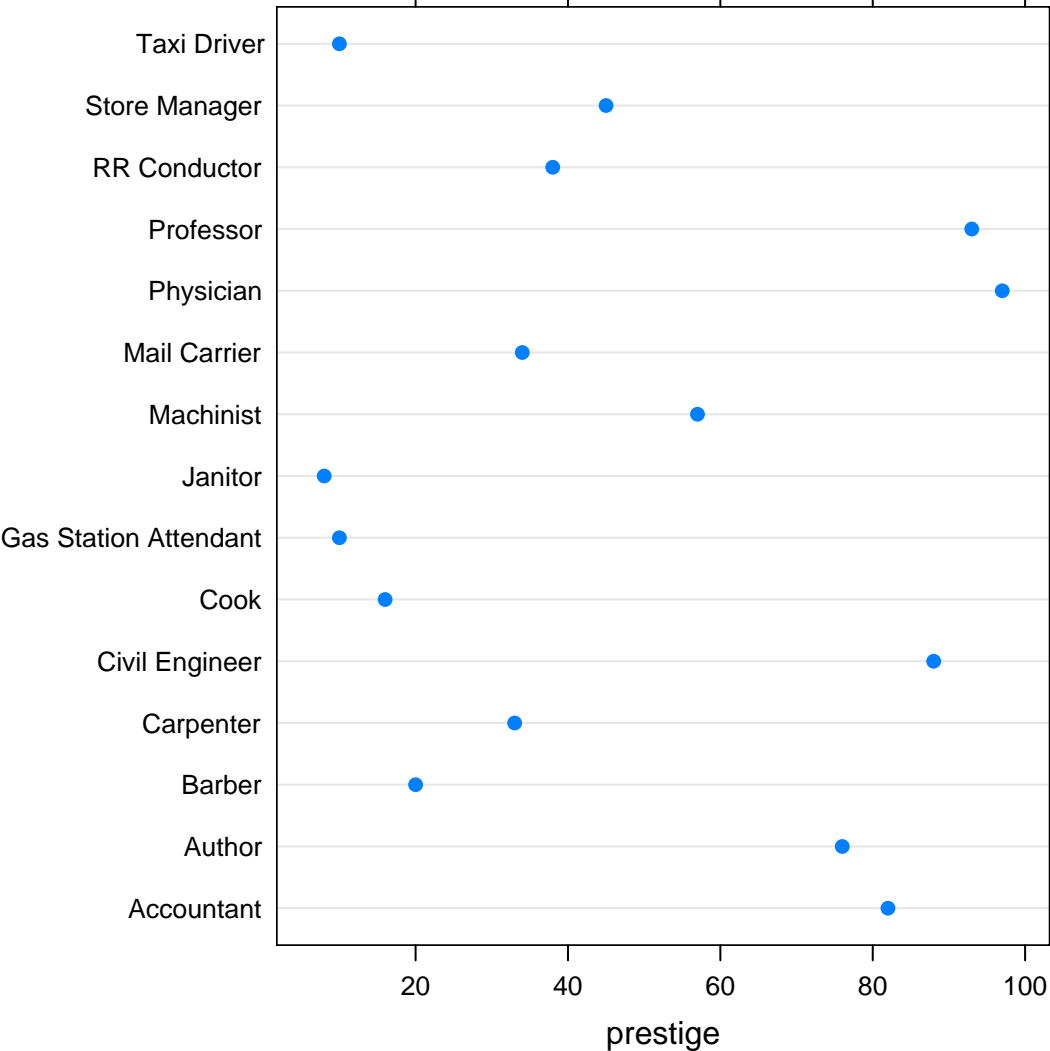
Note: Horizontal axis limits are determined by the empirical range of the data values.

Figure 9B: Bar Chart of Mean Issue Importance Ratings, 2004 CPS National Election Study (Alternate Version).



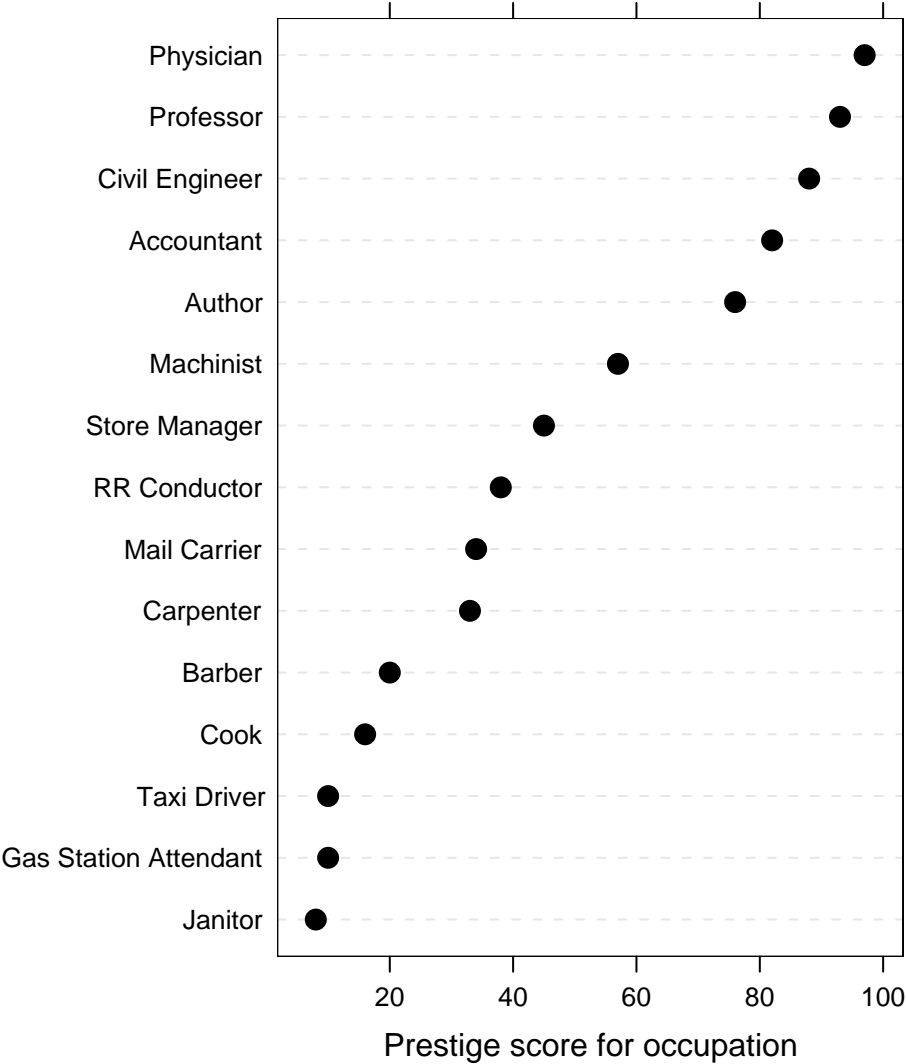
Note: Horizontal axis limits are set explicitly to zero and five.

Figure 10: Dot Plot of Prestige Scores for 15 Occupations.



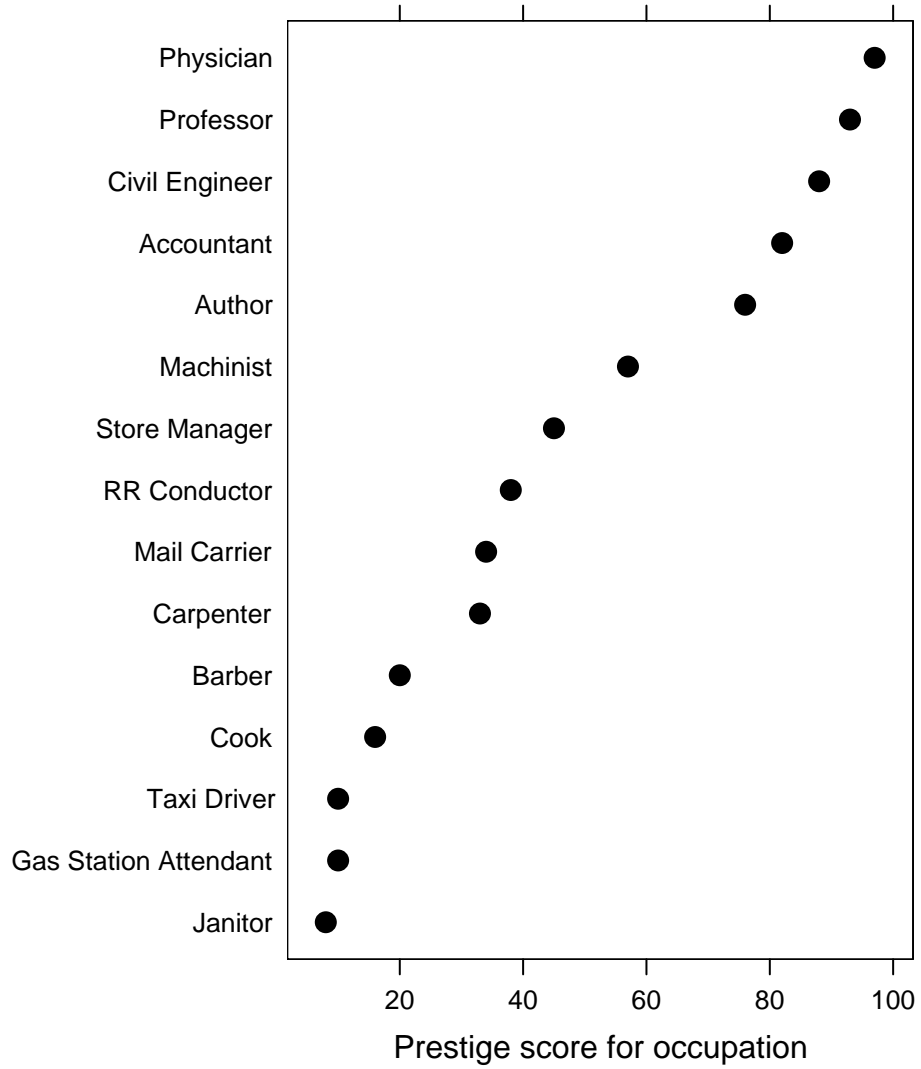
Data Source: Duncan (1961).

Figure 11: Dot Plot of Prestige Scores for 15 Occupations, with Points Sorted by Data Values.



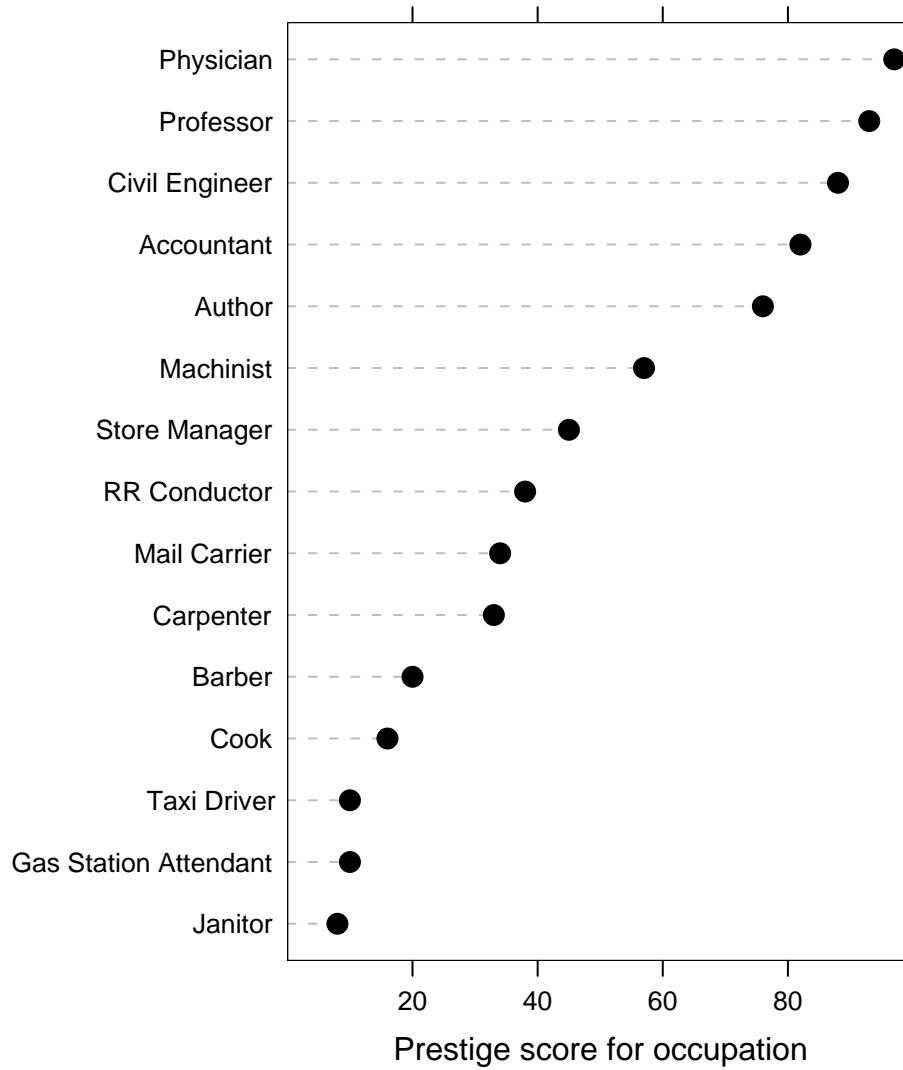
Data Source: Duncan (1961).

Figure 12: Dot Plot of Prestige Scores for 15 Occupations, with Reference Lines Omitted.



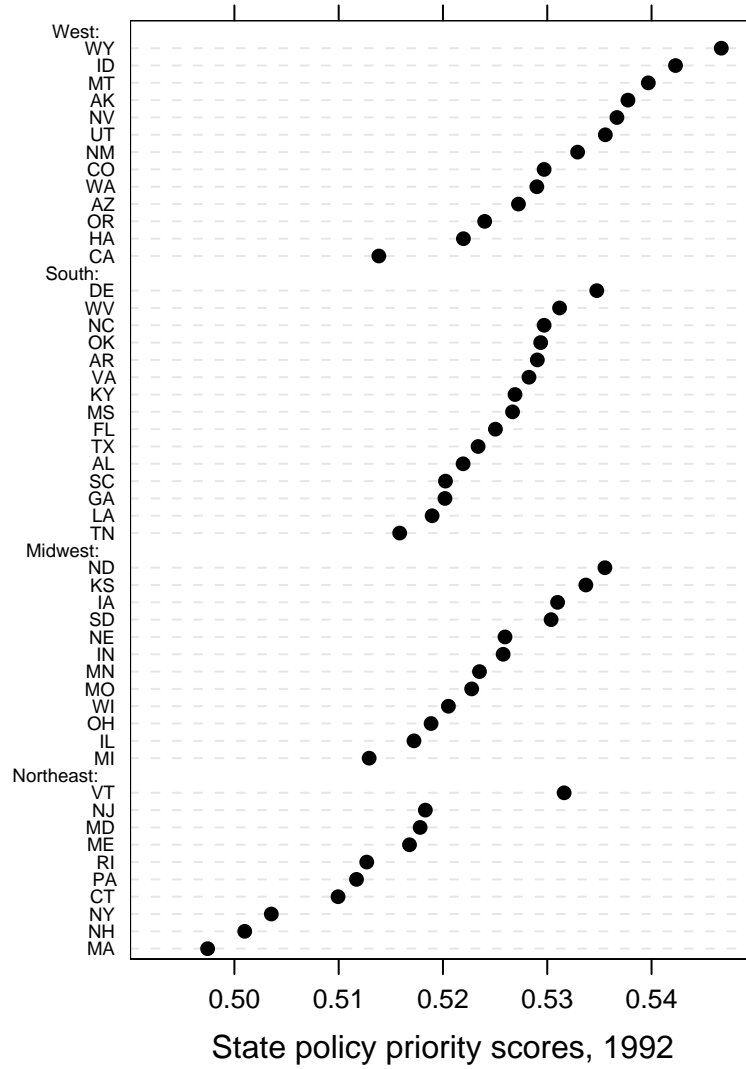
Data Source: Duncan (1961).

Figure 13: Dot Plot of Prestige Scores for 15 Occupations, with Reference Lines Proportional to Data Values.



Data Source: Duncan (1961).

Figure 14: Dot Plot of 1992 State Policy Priority Scores, by Region (Alternate Version).



Data Source: Jacoby and Schneider (2001).