Chapter 3
Visualizing Data

Visually presenting data and the results of models has become a centerpiece of modern political analysis. Many of Political Science's top journals, including the *American Journal of Political Science*, now ask for figures in lieu of tables whenever both can convey the same information. In fact, Kastellec and Leoni (2007) make the case that figures convey empirical results better than tables. Cleveland (1993) and Tufte (2001) wrote two of the leading volumes that describe the elements of good quantitative visualization, and Yau (2011) has produced a more recent take on graphing. Essentially these works serve as style manuals for graphics.[1] Beyond the suggestions these scholars offer for the sake of readers, viewing one's own data visually conveys substantial information about the data's univariate, bivariate, and multivariate features: Does a variable appear skewed? Do two variables substantively appear to correlate? What is the proper functional relationship between variables? How does a variable change over space or time? Answering these questions for oneself as an analyst and for the reader generally can raise the quality of analysis presented to the discipline.

On the edge of this graphical movement in quantitative analysis, R offers state-of-the-art data and model visualization. Many of the commercial statistical programs have tried for years to catch up to R's graphical capacities. This chapter showcases these capabilities, turning first to the `plot` function that is automatically available as part of the `base` package. Second, we discuss some of the other graphing commands offered in the `base` library. Finally, we turn to the `lattice` library, which allows the user to create Trellis Graphics—a framework for visualization

Electronic supplementary material: The online version of this chapter (doi: 10.1007/978-3-319-23446-5_3) contains supplementary material, which is available to authorized users.

[1] Other particularly key historical figures in the development of graphical measures include Halley (1686), Playfair (1786/2005), and Tukey (1977). A more comprehensive history is presented by Beniger and Robyn (1978).

J.E. Monogan III, *Political Analysis Using R*, Use R!,
DOI 10.1007/978-3-319-23446-5_3

developed by Becker, Cleveland, and others to put Cleveland's (1993) suggestions into practice. Although space does not permit it here, users are also encouraged to look up the `ggplot2` packages, which offers additional graphing options. Chang (2013), in particular, offers several examples of graphing with `ggplot2`.

In this chapter, we work with two example datasets. The first is on health lobbying in the 50 American states, with a specific focus on the proportion of firms from the health finance industry that are registered to lobby (Lowery et al. 2008). A key predictor variable is the total number of health finance firms open for business, which includes organizations that provide health plans, business services, employer health coalitions, and insurance. The dataset also includes the lobby participation rate by state, or number of lobbyists as a proportion of the number of firms, not only in health finance but for all health-related firms and in six other subareas. These are cross-sectional data from the year 1997. The complete variable list is as follows:

stno: Numeric index from 1–50 that orders the states alphabetically.
raneyfolded97: Folded Ranney Index of state two-party competition in 1997.[2]
healthagenda97: Number of bills related to health considered by the state legislature in 1997.
supplybusiness: Number of health finance establishments.
businesssupplysq: Squared number of health finance establishments.
partratebusness: Lobby participation rate for health finance—number of registrations as a percentage of the number of establishments.
predictbuspartrate: Prediction of health finance participation rate as a quadratic function of number of health finance establishments. (No control variables in the prediction.)
partratetotalhealth: Lobby participation rate for all of health care (including seven subareas).
partratedpc: Lobby participation rate for direct patient care.
partratepharmprod: Lobby participation rate for drugs and health products.
partrateprofessionals: Lobby participation rate for health professionals.
partrateadvo: Lobby participation rate for health advocacy.
partrategov: Lobby participation rate for local government.
rnmedschoolpartrate: Lobby participation rate for health education.

Second, we analyze Peake and Eshbaugh-Soha's (2008) data on the number of television news stories related to energy policy in a given month. In this data frame, the variables are:

Date: Character vector of the month and year observed.
Energy: Number of energy-related stories broadcast on nightly television news by month.
Unemploy: The unemployment rate by month.
Approval: Presidential approval by month.
oilc: Price of oil per barrel.

[2]Nebraska and North Carolina are each missing observations of the Ranney index.

freeze1: An indicator variable coded 1 during the months of August–November 1971, when wage and price freezes were imposed. Coded 0 otherwise.

freeze2: An indicator variable coded 1 during the months of June–July 1973, when price wage and price freezes were imposed. Coded 0 otherwise.

embargo: An indicator variable coded 1 during the months of October 1973–March 1974, during the Arab oil embargo. Coded 0 otherwise.

hostages: An indicator variable coded 1 during the months of November 1979–January 1981, during the Iran hostage crisis. Coded 0 otherwise.

Presidential speeches: Additional indicators are coded as 1 during the month a president delivered a major address on energy policy, and 0 otherwise. The indicators for the respective speeches are called: **rmn1173**, **rmn1173a**, **grf0175**, **grf575**, **grf575a**, **jec477**, **jec1177**, **jec479**, **grf0175s**, **jec479s**, and **jec477s**.

3.1 Univariate Graphs in the base Package

As a first look at our data, displaying a single variable graphically can convey a sense of the distribution of the data, including its mode, dispersion, skew, and kurtosis. The lattice library actually offers a few more commands for univariate visualization than base does, but we start with the major built-in univariate commands. Most graphing commands in the base package call the plot function, but hist and boxplot are noteworthy exceptions.

The hist command is useful to simply gain an idea of the relative frequency of several common values. We start by loading our data on energy policy television news coverage. Then we create a **hist**ogram of this time series of monthly story counts with the hist command. First, download Peake and Eshbaugh-Soha's data on energy policy coverage, the file named PESenergy.csv. The file is available from the Dataverse named on page vii or the chapter content link on page 33. You may need to use setwd to point R to the folder where you have saved the data. After this, run the following code:

```
pres.energy<-read.csv("PESenergy.csv")
hist(pres.energyEnergy,xlab="Television Stories",main="")
abline(h=0,col='gray60')
box()
```

The result this code produces is presented in Fig. 3.1. In this code, we begin by reading Peake and Eshbaugh-Soha's (2008) data. The data file itself is a comma-separated values file with a header row of variable names, so the defaults of read.csv suit our purposes. Once the data are loaded, we plot a histogram of our variable of interest using the hist command: pres.energy$Energy calls the variable of interest from its data frame. We use the xlab option, which allows us to define the label R prints on the horizontal axis. Since this axis shows us the values of the variable, we simply wish to see the phrase "Television Stories," describing in brief what these numbers mean. The main option defines a title printed over the top of the figure. In this case, the only way to impose a blank title is to include quotes with no content between them. A neat feature of plotting in the base package is

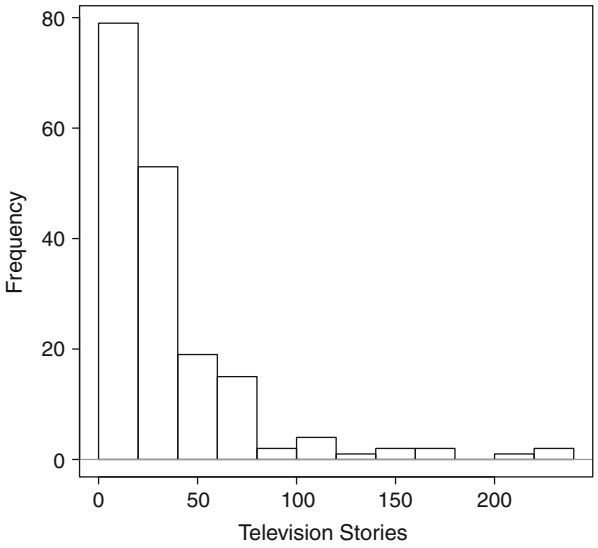

Fig. 3.1 Histogram of the monthly count of television news stories related to energy

that a few commands can add additional information to a plot that has already been drawn. The `abline` command is a flexible and useful tool. (The name **a**-**b** **line** refers to the linear formula $y = a + bx$. Hence, this command can draw lines with a slope and intercept, or it can draw a horizontal or vertical line.) In this case, `abline` adds a horizontal line along the 0 point on the vertical axis, hence `h=0`. This is added to clarify where the base of the bars in the figure is. Finally, the `box()` command encloses the whole figure in a box, often useful in printed articles for clarifying where graphing space ends and other white space begins. As the histogram shows, there is a strong concentration of observations at and just above 0, and a clear positive skew to the distribution. (In fact, these data are reanalyzed in Fogarty and Monogan (2014) precisely to address some of these data features and discuss useful means of analyzing time-dependent media counts.)

Another univariate graph is a box-and-whisker plot. R allows us to obtain this solely for the single variable, or for a subset of the variable based on some other available measure. First drawing this for a single variable:

```
boxplot(pres.energy$Energy,ylab="Television Stories")
```

The result of this is presented in panel (a) of Fig. 3.2. In this case, the values of the monthly counts are on the vertical axis; hence, we use the `ylab` option to label the vertical axis (or **y**-axis **lab**el) appropriately. In the figure, the bottom of the box represents the first quartile value (25th percentile), the large solid line inside the box represents the median value (second quartile, 50th percentile), and the top of the box represents the third quartile value (75th percentile). The whiskers, by default, extend to the lowest and highest values of the variable that are no more than 1.5 times the interquartile range (or difference between third and first quartiles) away

from the box. The purpose of the whiskers is to convey the range over which the bulk of the data fall. Data falling outside of this range are portrayed as dots at their respective values. This boxplot fits our conclusion from the histogram: small values including 0 are common, and the data have a positive skew.

Box-and-whisker plots also can serve to offer a sense of the conditional distribution of a variable. For our time series of energy policy coverage, the first major event we observe is Nixon's November 1973 speech on the subject. Hence, we might create a simple indicator where the first 58 months of the series (through October 1973) are coded 0, and the remaining 122 months of the series (November 1973 onward) are coded 1. Once we do this, the `boxplot` command allows us to condition on a variable:

```
pres.energy$post.nixon<-c(rep(0,58),rep(1,122))
boxplot(pres.energy$Energy~pres.energy$post.nixon,
     axes=F,ylab="Television Stories")
axis(1,at=c(1,2),labels=c("Before Nov. 1973",
     "After Nov. 1973"))
axis(2)
box()
```

This output is presented in panel (b) of Fig. 3.2. The first line of code defines our pre v. post November 1973 variable. Notice here that we again define a vector with `c`. Within `c`, we use the `rep` command (for **rep**eat). So `rep(0,58)` produces 58 zeroes, and `rep(1,122)` produces 122 ones. The second line draws our boxplots, but we add two important caveats relative to our last call to `boxplot`: First, we list `pres.energy$Energy~pres.energy$post.nixon` as our data argument. The argument before the tilde (~) is the variable for which we want the distribution, and the argument afterward is the conditioning variable. Second, we add the `axes=F` command. (We also could write `axes=FALSE`, but R accepts `F` as an abbreviation.) This gives us more control over how the horizontal and

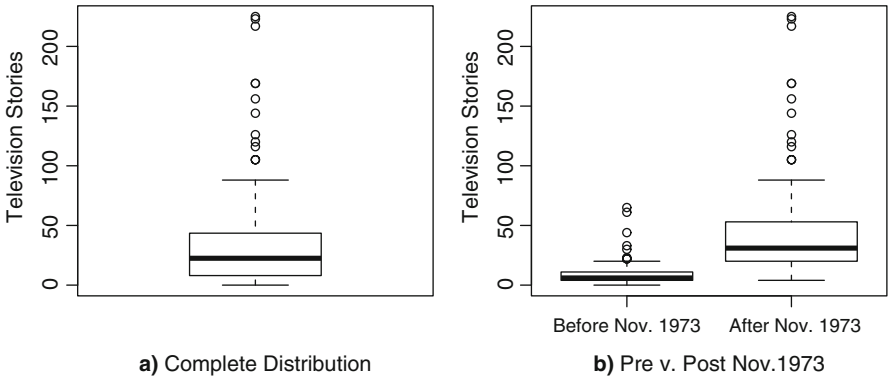

Fig. 3.2 Box-and-whisker plots of the distribution of the monthly count of television new stories related to energy. Panel (**a**) shows the complete distribution, and panel (**b**) shows the distributions for the subsets before and after November 1973

vertical axes are presented. In the subsequent command, we add axis 1 (the bottom horizontal axis), adding text labels at the tick marks of 1 and 2 to describe the values of the conditioning variable. Afterward, we add axis 2 (the left vertical axis), and a box around the whole figure. Panel (b) of Fig. 3.2 shows that the distribution before and after this date is fundamentally different. Much smaller values persist before Nixon's speech, while there is a larger mean and a greater spread in values afterward. Of course, this is only a first look and the effect of Nixon's speech is confounded with a variety of factors—such as the price of oil, presidential approval, and the unemployment rate—that contribute to this difference.

3.1.1 Bar Graphs

Bar graphs can be useful whenever we wish to illustrate the value some statistic takes for a variety of groups as well as for visualizing the relative proportions of nominal or ordinally measured data. For an example of barplots, we turn now to the other example data set from this chapter, on health lobbying in the 50 American states. Lowery et al. offer a bar graph of the means across all states of the lobbying participation rate—or number of lobbyists as a percentage of number of firms—for all health lobbyists and for seven subgroups of health lobbyists (2008, Fig. 3). We can recreate that figure in R by taking the means of these eight variables and then applying the `barplot` function to the set of means. First we must load the data. To do this, download Lowery et al.'s data on lobbying, the file named `constructionData.dta`. The file is available from the Dataverse named on page vii or the chapter content link on page 33. Again, you may need to use `setwd` to point R to the folder where you have saved the data. Since these data are in Stata format, we must use the `foreign` library and then the `read.dta` command:

```
library(foreign)
health.fin<-read.dta("constructionData.dta")
```

To create the actual figure itself, we can create a subset of our data that only includes the eight predictors of interest and then use the `apply` function to obtain the mean of each variable.

```
part.rates<-subset(health.fin,select=c(
     partratetotalhealth,partratedpc,
     partratepharmprod,partrateprofessionals,partrateadvo,
     partratebusness,partrategov,rnmedschoolpartrate))
lobby.means<-apply(part.rates,2,mean)
names(lobby.means)<-c("Total Health Care",
     "Direct Patient Care","Drugs/Health Products",
     "Health Professionals","Health Advocacy","
     Health Finance","Local Government","Health Education")
```

In this case, `part.rates` is our subsetted data frame that only includes the eight lobby participation rates of interest. On the last line, the `apply` command allows us to take a matrix or data frame (`part.rates`) and apply a function of interest (`mean`) to either the rows or the columns of the data frame. We want the mean of

each variable, and the columns of our data set represent the variables. The 2 that is the second component of this command therefore tells apply that we want to apply mean to the *columns* of our data. (By contrast, an argument of 1 would apply to the *rows*. Row-based computations would be handy if we needed to compute some new quantity for each of the 50 states.) If we simply type lobby.means into the R console now, it will print the eight means of interest for us. To set up our figure in advance, we can attach an English-language name to each quantity that will be reported in our figure's margin. We do this with the names command, and then assign a vector with a name for each quantity.

To actually draw our bar graph, we use the following code:

```
par(mar=c(5.1, 10 ,4.1 ,2.1))
barplot(lobby.means,xlab="Percent Lobby Registration",
     xlim=c(0,26),horiz=T,cex.names=.8,las=1)
text(x=lobby.means,y=c(.75,1.75,3,4.25,5.5,6.75,8,9),
     labels=paste(round(lobby.means,2)),pos=4)
box()
```

The results are plotted in Fig. 3.3. The first line calls the par command, which allows the user to change a wide array of defaults in the graphing space. In our

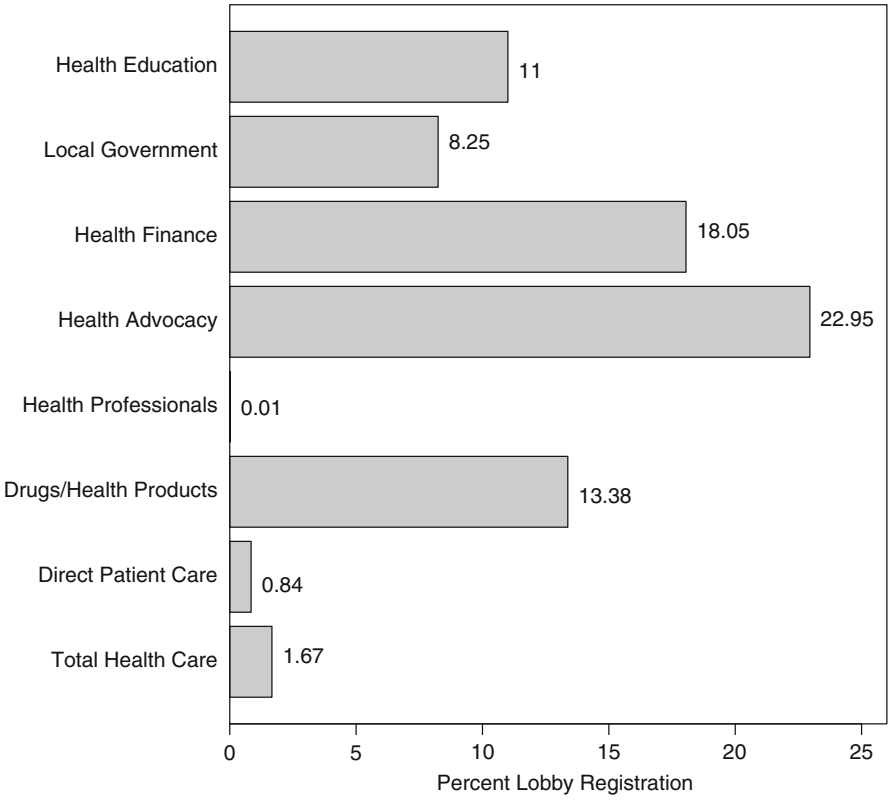

Fig. 3.3 Bar graph of the mean lobbying participation rate in health care and in seven sub-guilds across the 50 U.S. states, 1997

case, we need a bigger left margin, so we used the `mar` option to change this, setting the second value to the relatively large value of 10. (In general, the margins are listed as bottom, left, top, then right.) Anything adjusted with `par` is reset to the defaults after the plotting window (or device, if writing directly to a file) is closed. Next, we actually use the `barplot` command. The main argument is `lobby.means`, which is the vector of variable means. The default for `barplot` is to draw a graph with vertical lines. In this case, though, we set the option `horiz=T` to get horizontal bars. We also use the options `cex.names` (**c**haracter **ex**pansion for axis **names**) and `las=1` (**l**abel **a**xis **s**tyle) to shrink our bar labels to 80 % of their default size and force them to print horizontally, respectively.[3] The `xlab` command allows us to describe the variable for which we are showing the means, and the `xlim` (**x**-axis **lim**its) command allows us to set the space of our horizontal axis. Finally, we use the `text` command to print the mean for each lobby registration rate at the end of the bar. The `text` command is useful any time we wish to add text to a graph, be these numeric values or text labels. This command takes `x` coordinates for its position along the horizontal axis, `y` coordinates for its position along the vertical axis, and `labels` values for the text to print at each spot. The `pos=4` option specifies to print the text to the right of the given point (alternatively 1, 2, and 3 would specify below, left, and above, respectively), so that our text does not overlap with the bar.

3.2 The `plot` Function

We turn now to `plot`, the workhorse graphical function in the `base` package. The `plot` command lends itself naturally to bivariate plots. To see the total sum of arguments that one can call using `plot`, type `args(plot.default)`, which returns the following:

```
function (x, y=NULL, type="p", xlim=NULL, ylim=NULL,
    log="", main=NULL, sub=NULL, xlab=NULL, ylab=NULL,
    ann=par("ann"), axes=TRUE, frame.plot=axes,
    panel.first=NULL, panel.last=NULL, asp=NA, ...)
```

Obviously there is a lot going on underneath the generic `plot` function. For the purpose of getting started with figure creation in R we want to ask what is essential. The answer is straightforward: one variable `x` must be specified. Everything else has either a default value or is not essential. To start experimenting with `plot`, we continue to use the 1997 state health lobbying data loaded in Sect. 3.1.1.

With `plot`, we can plot the variables separately with the command `plot(varname)`, though this is definitively less informative than the kinds of

[3] The default `las` value is 0, which prints labels parallel to the axis. 1, our choice here, prints them horizontally. 2 prints perpendicular to the axis, and 3 prints them vertically.

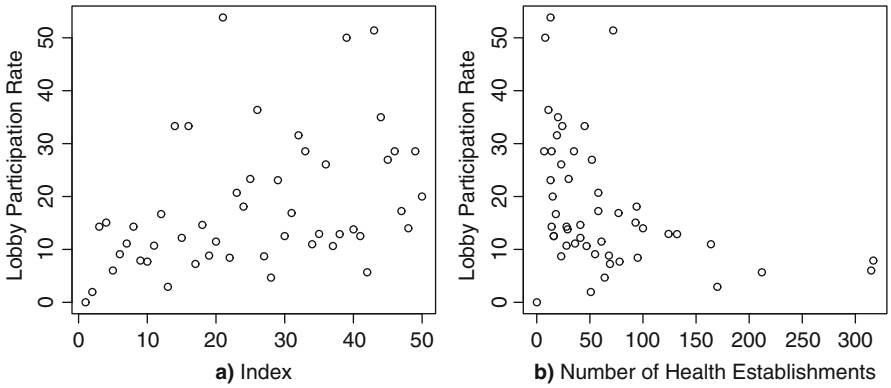

Fig. 3.4 Lobby participation rate of the health finance industry alone and against the number of health finance business establishments. (**a**) Index. (**b**) Number of Health Establishments

graphs just presented in Sect. 3.1. That said, if we simply wanted to see all of the observed values of the lobby participation rate by state of health finance firms (`partratebusness`), we simply type:

```
plot(health.fin$partratebusness,
     ylab="Lobby Participation Rate")
```

Figure 3.4a is returned in the R graphics interface. Note that this figure plots the lobby participation rate against the row number in the data frame: With cross-sectional data this index is essentially meaningless. By contrast, if we were studying time series data, and the data were sorted on time, then we could observe how the series evolves over time. Note that we use the `ylab` option because otherwise the default will label our vertical axis with the tacky-looking `health.fin$partratebusness`. (Try it, and ask yourself what a journal editor would think of how the output looks.)

Of course, we are more often interested in bivariate relationships. We can explore these easily by incorporating a variable `x` on the horizontal axis (usually an *independent* variable) and a variable `y` on the vertical axis (usually a *dependent* variable) in the call to plot:

```
plot(y=health.fin$partratebusness,x=health.fin$supplybusiness,
     ylab="Lobby Participation Rate",
     xlab="Number of Health Establishments")
```

This produces Fig. 3.4b, where our horizontal axis is defined by the number of health finance firms in a state, and the vertical axis is defined by the lobby participation rate of these firms in the respective state. This graph shows what appears to be a decrease in the participation rate as the number of firms rises, perhaps in a curvilinear relationship.

One useful tool is to plot the functional form of a bivariate model onto the scatterplot of the two variables. In the case of Fig. 3.4b, we may want to compare

how a linear function versus a quadratic, or squared, function of the number of firms fits the outcome of lobby participation rate. To do this, we can fit two linear regression models, one that includes a linear function of number of firms, and the other that includes a quadratic function. Additional details on regression models are discussed later on in Chap. 6. Our two models in this case are:

```
finance.linear<-lm(partratebusness~supplybusiness,
     data=health.fin)
summary(finance.linear)
finance.quadratic<-lm(partratebusness~supplybusiness+
     I(supplybusiness^2),data=health.fin)
summary(finance.quadratic)
```

The `lm` (**l**inear **m**odel) command fits our models, and the `summary` command summarizes our results. Again, details of `lm` will be discussed in Chap. 6. With the model that is a linear function of number of firms, we can simply feed the name of our fitted model (`finance.linear`) into the command `abline` in order to add our fitted regression line to the plot:

```
plot(y=health.fin$partratebusness,x=health.fin$supplybusiness,
     ylab="Lobby Participation Rate",
     xlab="Number of Health Establishments")
abline(finance.linear)
```

As mentioned before, the `abline` command is particularly flexible. A user can specify `a` as the intercept of a line and `b` as the slope. A user can specify `h` as the vertical-axis value where a horizontal line is drawn, or `v` as the horizontal-axis value where a vertical line is drawn. Or, in this case, a regression model with one predictor can be inserted to draw the best-fitting regression line. The results are presented in Fig. 3.5a.

Alternatively, we could redraw this plot with the quadratic relationship sketched on it. Unfortunately, despite `abline`'s flexibility, it cannot draw a quadratic

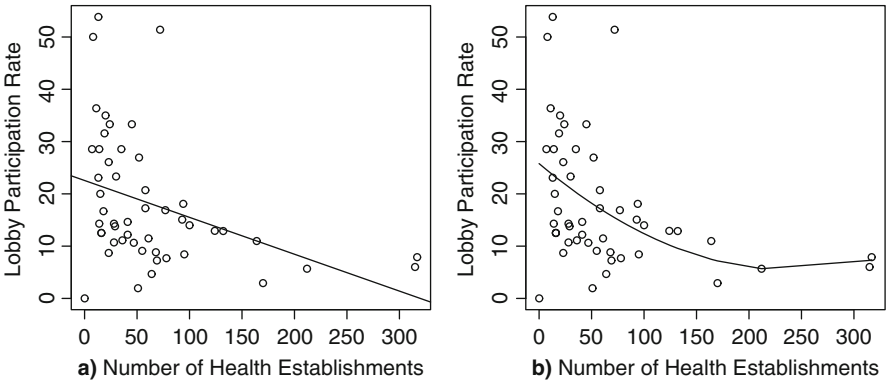

Fig. 3.5 Lobby participation rate of the health finance industry against the number of health establishments, linear and quadratic models. (**a**) Linear function. (**b**) Quadratic function

relationship by default. The easiest way to plot a complex functional form is to save the predicted values from the model, reorder the data based on the predictor of interest, and then use the lines function to add a connected line of all of the predictions. Be sure the data are properly ordered on the predictor, otherwise the line will appear as a jumbled mess. The code in this case is:

```
plot(y=health.fin$partratebusness,x=health.fin$supplybusiness,
     ylab="Lobby Participation Rate",
     xlab="Number of Health Establishments")
finance.quadratic<-lm(partratebusness~supplybusiness+
     I(supplybusiness^2), data=health.fin)
health.fin$quad.fit<-finance.quadratic$fitted.values
health.fin<-health.fin[order(health.fin$supplybusiness),]
lines(y=health.fin$quad.fit,x=health.fin$supplybusiness)
```

This outcome is presented in Fig. 3.5b. While we will not get into lm's details yet, notice that I(supplybusiness^2) is used as a predictor. I means "as **is**", so it allows us to compute a mathematical formula on the fly. After redrawing our original scatterplot, we estimate our quadratic model and save the fitted values to our data frame as the variable quad.fit. On the fourth line, we reorder our data frame health.fin according to the values of our input variable supplybusiness. This is done by using the order command, which lists vector indices in order of increasing value. Finally, the lines command takes our predicted values as the vertical coordinates (y) and our values of the number of firms as the horizontal coordinates (x). This adds the line to the plot showing our quadratic functional form.

3.2.1 *Line Graphs with* `plot`

So far, our analyses have relied on the plot default of drawing a scatterplot. In time series analysis, though, a line plot over time is often useful for observing the properties of the series and how it changes over time. (Further information on this is available in Chap. 9.) Returning to the data on television news coverage of energy policy first raised in Sect. 3.1, let us visualize the outcome of energy policy coverage and an input of oil price.

Starting with number of energy stories by month, we create this plot as follows:

```
plot(x=pres.energy$Energy,type="l",axes=F,
     xlab="Month", ylab="Television Stories on Energy")
axis(1,at=c(1,37,73,109,145),labels=c("Jan. 1969",
     "Jan. 1972","Jan. 1975","Jan. 1978","Jan. 1981"),
     cex.axis=.7)
axis(2)
abline(h=0,col="gray60")
box()
```

This produces Fig. 3.6a. In this case, our data are already sorted by month, so if we only specify x with no y, R will show all of the values in correct temporal

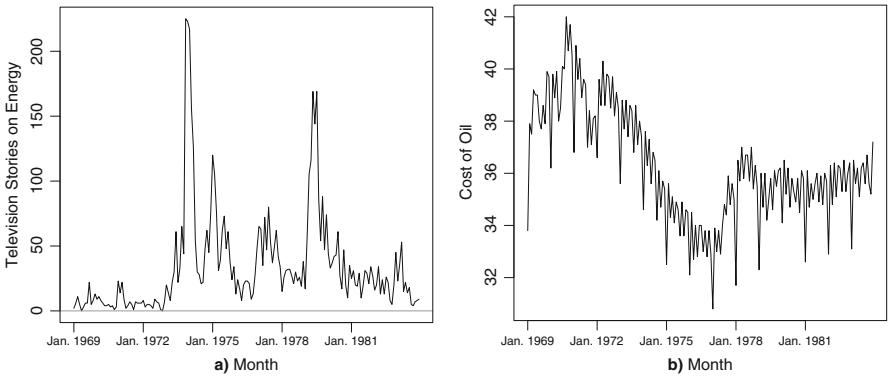

Fig. 3.6 Number of television news stories on energy policy and the price of oil per barrel, respectively, by month. (**a**) News coverage. (**b**) Oil price

order.[4] To designate that we want a line plot instead of a scatterplot of points, we insert the letter `l` in the `type="l"` option. In this case, we have turned off the axes because the default tick marks for month are not particularly meaningful. Instead, we use the `axis` command to insert a label for the first month of the year every 3 years, offering a better sense of real time. Notice that in our first call to `axis`, we use the `cex.axis` option to shrink our labels to 70 % size. This allows all five labels to fit in the graph. (By trial and error, you will see that R drops axis labels that will not fit rather than overprint text.) Finally, we use `abline` to show the zero point on the vertical axis, since this is a meaningful number that reflects the complete absence of energy policy coverage in television news. As our earlier figures demonstrated, we see much more variability and a higher mean after the first 4 years. The figure of the price of oil per barrel can be created in a similar way:

```
plot(x=pres.energy$oilc,type="l",axes=F,xlab="Month",
     ylab="Cost of Oil")
axis(1,at=c(1,37,73,109,145),labels=c("Jan. 1969",
     "Jan. 1972","Jan. 1975","Jan. 1978","Jan. 1981"),
     cex.axis=.7)
axis(2)
box()
```

Again, the data are sorted, so only one variable is necessary. Figure 3.6b presents this graph.

[4] Alternatively, though, if a user had some time index in the data frame, a similar plot could be produced by typing something to the effect of: `pres.energy$Time<-1:180; plot(y=pres.energy$Energy,x=pres.energy$Time,type="l")`.

3.2.2 *Figure Construction with* `plot`*: Additional Details*

Having tried our hand with plots from the `base` package, we will now itemize in detail the basic functions and options that bring considerable flexibility to creating figures in R. Bear in mind that R actually offers the useful option of beginning with a blank slate and adding items to the graph bit-by-bit.

The Coordinate System: In Fig. 3.4, we were not worried about establishing the coordinate system because the data effectively did this for us. But often, you will want to establish the dimensions of the figure before plotting anything—especially if you are building up from the blank canvas. The most important point here is that your `x` and `y` must be of the same length. This is perhaps obvious, but missing data can create difficulties that will lead R to balk.

Plot Types: We now want to plot these series, but the `plot` function allows for different types of plots. The different types that one can include within the generic `plot` function include:

- `type="p"` This is the default and it plots the `x` and `y` coordinates as *points*.
- `type="l"` This plots the `x` and `y` coordinates as *lines*.
- `type="n"` This plots the `x` and `y` coordinates as *nothing* (it sets up the coordinate space only).
- `type="o"` This plots the `x` and `y` coordinates as *points and lines* overlaid (i.e., it "overplots").
- `type="h"` This plots the `x` and `y` coordinates as *histogram-like vertical lines*. (Also called a *spike plot*.)
- `type="s"` This plots the `x` and `y` coordinates as *stair-step like lines*.

Axes: It is possible to turn off the axes, to adjust the coordinate space by using the `xlim` and `ylim` options, and to create your own labels for the axes.

`axes=` Allows you to control whether the axes appear in the figure or not. If you have strong preferences about how your axes are created, you may turn them off by selecting `axes=F` within `plot` and then create your own labels using the separate `axis` command:

- `axis(side=1,at=c(2,4,6,8,10,12),labels=c("Feb", "Apr","June","Aug","Oct","Dec"))`

`xlim=, ylim=` For example, if we wanted to expand the space from the R default, we could enter:

- `plot(x=ind.var, y=dep.var, type="o", xlim=c(-5, 17),ylim=c(-5, 15))`

`xlab="", ylab=""` Creates labels for the x- and y-axis.

Style: There are a number of options to adjust the style in the figure, including changes in the line type, line weight, color, point style, and more. Some common commands include:

`asp=` Defines the **asp**ect ratio of the plot. Setting `asp=1` is a powerful and useful option that allows the user to declare that the two axes are measured on the same scale. See Fig. 5.1 on page 76 and Fig. 8.4 on page 153 as two examples of this option.

`lty=` Selects the type of line (solid, dashed, short-long dash, etc.).

`lwd=` Selects the line width (fat or skinny lines).

`pch=` Selects the plotting symbol, can either be a numbered symbol (`pch=1`) or a letter (`pch="D"`).

`col=` Selects the color of the lines or points in the figure.

`cex=` **C**haracter **ex**pansion factor that adjusts the size of the text and symbols in the figure. Similarly, `cex.axis` adjusts axis annotation size, `cex.lab` adjusts font size for axis labels, `cex.main` adjusts the font size of the title, and `cex.sub` adjusts subtitle font size.

Graphing Parameters: The `par` function brings added functionality to plotting in R by giving the user control over the graphing **par**ameters. One noteworthy feature of `par` is that it allows you to plot multiple calls to `plot` in a single graphic. This is accomplished by selecting `par(new=T)` while a plot window (or device) is still open and before the next call to `plot`. Be *careful*, though. Any time you use this strategy, include the `xlim` and `ylim` commands in each call to make sure the graphing space stays the same. Also be careful that graph margins are not changing from one call to the next.

3.2.3 Add-On Functions

There are also a number of add-on functions that one can use once the basic coordinate system has been created using `plot`. These include:

`arrows(x1, y1, x2, y2)` Create arrows within the plot (useful for labeling particular data points, series, etc.).

`text(x1, x2, "text")` Create text within the plot (modify size of text using the character expansion option `cex`).

`lines(x, y)` Create a plot that connects lines.

`points(x, y)` Create a plot of points.

`polygon()` Create a polygon of any shape (rectangles, triangles, etc.).

`legend(x, y, at = c("", ""), labels=c("", ""))` Create a legend to identify the components in the figure.

`axis(side)` Add an axis with default or customized labels to one of the sides of a plot. Set the side to 1 for bottom, 2 for left, 3 for top, and 4 for right.

`mtext(text, side)` Command to add **m**argin **text**. This lets you add an axis label to one of the sides with more control over how the label is presented. See the code that produces Fig. 7.1 on page 108 for an example of this.

3.3 Using lattice Graphics in R

As an alternative to the base graphics package, you may want to consider the lattice add-on package. These produce trellis graphics from the S language, which tend to make better displays of grouped data and numerous observations. A few nice features of the lattice package are that the graphs have viewer-friendly defaults, and the commands offer a data= option that does not require the user to list the data frame with every call to a variable.

To start, the first time we use the lattice library, we must install it. Then, on every reuse of the package, we must call it with the library command.

```
install.packages("lattice")
library(lattice)
```

To obtain a scatterplot similar to the one we drew with plot, this can be accomplished in lattice using the xyplot command:

```
xyplot(partratebusness~supplybusiness,data=health.fin,
     col="black",ylab="Lobby Participation Rate",
     xlab="Number of Health Establishments")
```

Figure 3.7a displays this graph. The syntax differs from the plot function somewhat: In this case, we can specify an option, data=health.fin, that allows us to type the name of the relevant data frame once, rather than retype it for each variable. Also, both variables are listed together in a single argument using the form, vertical.variable~horizontal.variable. In this case, we also specify the option, col="black" for the sake of producing a black-and-white figure. By default lattice colors results cyan in order to allow readers to easily separate data information from other aspects of the display, such as axes and labels (Becker et al. 1996, p. 153). Also, by default, xyplot prints tick marks on the third and fourth axes to provide additional reference points for the viewer.

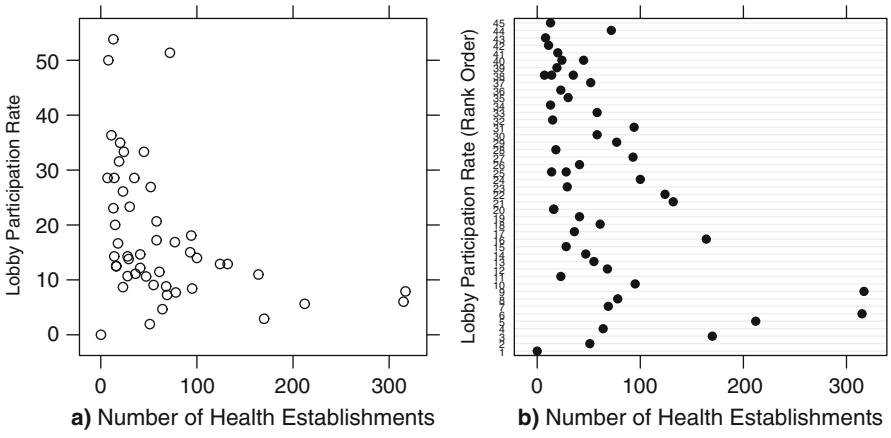

Fig. 3.7 Lobby participation rate of the health finance industry against the number of health establishments, (**a**) scatterplot and (**b**) dotplot

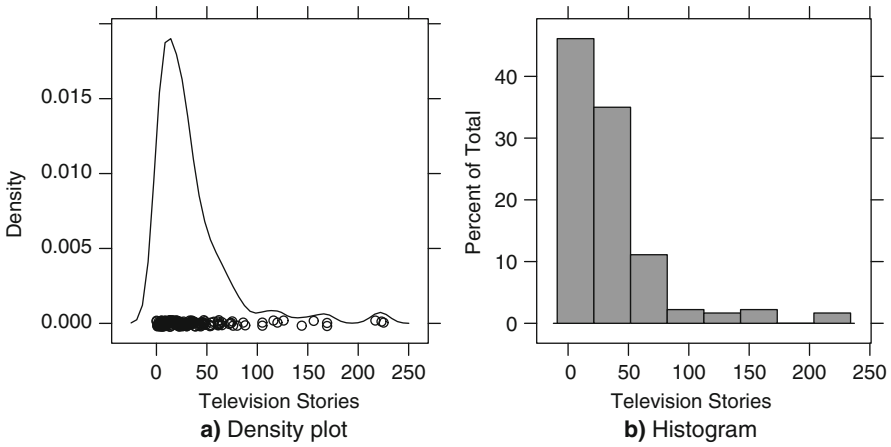

Fig. 3.8 (**a**) Density plot and (**b**) histogram showing the univariate distribution of the monthly count of television news stories related to energy

The `lattice` package also contains functions that draw graphs that are similar to a scatterplot, but instead use a rank-ordering of the vertical axis variable. This is how the `stripplot` and `dotplot` commands work, and they offer another view of a relationship and its robustness. The `dotplot` command may be somewhat more desirable as it also displays a line for each rank-ordered value, offering a sense that the scale is different. The `dotplot` syntax looks like this:

```
dotplot(partratebusness~supplybusiness,
     data=health.fin,col="black",
     ylab="Lobby Participation Rate (Rank Order)",
     xlab="Number of Health Establishments")
```

Figure 3.7b displays this result. The `stripplot` function uses similar syntax.

Lastly, the `lattice` library again gives us an option to look at the distribution of a single variable by plotting either a histogram or a density plot. Returning to the presidential time series data we first loaded in Sect. 3.1, we can now draw a density plot using the following line of code:

```
densityplot(~Energy,data=pres.energy,
     xlab="Television Stories",col="black")
```

This is presented in Fig. 3.8a. This output shows points scattered along the base, each representing the value of an observation. The smoothed line across the graph represents the estimated relative density of the variable's values.

Alternatively, a histogram in `lattice` can be drawn with the `histogram` function:

```
histogram(~Energy, data=pres.energy,
     xlab="Television Stories", col="gray60")
```

This is printed in Fig. 3.8b. In this case, color is set to `col="gray60"`. The default again is for cyan-colored bars. For a good grayscale option in

this case, a medium gray still allows each bar to be clearly distinguished. A final interesting feature of `histogram` is left to the reader: The function will draw conditional histogram distributions. If you still have the `post.nixon` variable available that we created earlier, you might try typing `histogram(~Energy|post.nixon,data=pres.energy)`, where the vertical pipe (`|`) is followed by the conditioning variable.

3.4 Graphic Output

A final essential point is a word on how users can export their R graphs into a desired word processor or desktop publisher. The first option is to save the screen output of a figure. On Mac machines, user may select the figure output window and then use the dropdown menu *File→Save As...* to save the figure as a PDF file. On Windows machines, a user can simply right-click on the figure output window itself and then choose to save the figure as either a metafile (which can be used in programs such as Word) or as a postscript file (for use in LaTeX). Also by right-clicking in Windows, users may copy the image and paste it into Word, PowerPoint, or a graphics program.

A second option allows users more precision over the final product. Specifically, the user can write the graph to a graphics device, of which there are several options. For example, in writing this book, I exported Fig. 3.5a by typing:

```
postscript("lin.partrate.eps",horizontal=FALSE,width=3,
     height=3,onefile=FALSE,paper="special",pointsize=7)
plot(y=health.fin$partratebusness,x=health.fin$supplybusiness,
     ylab="Lobby Participation Rate",
     xlab="Number of Health Establishments")
abline(finance.linear)
dev.off()
```

The first line calls the `postscript` command, which created a file called `lin.partrate.eps` that I saved the graph as. Among the key options in this command are `width` and `height`, each of which I set to three inches. The `pointsize` command shrank the text and symbols to neatly fit into the space I allocated. The `horizontal` command changes the orientation of the graphic from landscape to portrait orientation on the page. Change it to `TRUE` to have the graphic adopt a landscape orientation. Once `postscript` was called, all graphing commands wrote to the file *and not to the graphing window*. Hence, it is typically a good idea to perfect a graph before writing it to a graphics device. Thus, the `plot` and `abline` commands served to write all of the output to the file. Once I was finished writing to the file, the `dev.off()` command closed the file so that no other graphing commands would write to it.

Of course postscript graphics are most frequently used by writers who use the desktop publishing language of LaTeX. Writers who use more traditional word processors such as Word or Pages will want to use other graphics devices. The

available options include: `jpeg`, `pdf`, `png`, and `tiff`.[5] To use any of these four graphics devices, substitute a call for the relevant function where `postscript` is in the previous code. Be sure to type `?png` to get a feel for the syntax of these alternative devices, though, as each of the five has a slightly different syntax.

As a special circumstance, graphs drawn from the `lattice` package use a different graphics device, called `trellis.device`. It is technically possible to use the other graphics devices to write to a file, but unadvisable because the device options (e.g., size of graph or font size) will not be passed to the graph. In the case of Fig. 3.7b, I generated the output using the following code:

```
trellis.device("postscript",file="dotplot.partrate.eps",
     theme=list(fontsize=list(text=7,points=7)),
     horizontal=FALSE,width=3,height=3,
     onefile=FALSE,paper="special")
dotplot(partratebusness~supplybusiness,
     data=health.fin,col='black',
     ylab="Lobby Participation Rate (Rank Order)",
     xlab="Number of Health Establishments")
dev.off()
```

The first argument of the `trellis.device` command declares which driver the author wishes to use. Besides `postscript`, the author can use `jpeg`, `pdf`, or `png`. The second argument lists the file to write to. Font and character size must be set through the `theme` option, and the remaining arguments declare the other preferences about the output.

This chapter has covered univariate and bivariate graphing functions in R. Several commands from both the `base` and `lattice` packages have been addressed. This is far from an exhaustive list of R's graphing capabilities, and users are encouraged to learn more about the available options. This primer should, however, serve to introduce users to various means by which data can be visualized in R. With a good sense of how to get a feel for our data's attributes visually, the next chapter turns to numerical summaries of our data gathered through descriptive statistics.

3.5 Practice Problems

In addition to their analysis of energy policy coverage introduced in this chapter, Peake and Eshbaugh-Soha (2008) also study drug policy coverage. These data similarly count the number of nightly television news stories in a month focusing on drugs, from January 1977 to December 1992. Their data is saved in comma-separated format in the file named `drugCoverage.csv`. Download their data from the Dataverse named on page vii or the chapter content link on page 33. The variables in this data set are: a character-based time index showing month and year

[5] My personal experience indicates that `png` often looks pretty clear and is versatile.

(**Year**), news coverage of drugs (**drugsmedia**), an indicator for a speech on drugs that Ronald Reagan gave in September 1986 (**rwr86**), an indicator for a speech George H.W. Bush gave in September 1989 (**ghwb89**), the president's approval rating (**approval**), and the unemployment rate (**unemploy**).

1. Draw a histogram of the monthly count of drug-related stories. You may use either of the histogram commands described in the chapter.
2. Draw two boxplots: One of drug-related stories and another of presidential approval. How do these figures differ and what does that tell you about the contrast between the variables?
3. Draw two scatterplots:
   (a) In the first, represent the number of drug-related stories on the vertical axis, and place the unemployment rate on the horizontal axis.
   (b) In the second, represent the number of drug-related stories on the vertical axis, and place presidential approval on the horizontal axis.
   (c) How do the graphs differ? What do they tell you about the data?
   (d) Bonus: Add a linear regression line to each of the scatterplots.
4. Draw two line graphs:
   (a) In the first, draw the number of drug-related stories by month over time.
   (b) In the second, draw presidential approval by month over time.
   (c) What can you learn from these graphs?
5. Load the `lattice` library and draw a density plot of the number of drug-related stories by month.
6. Bonus: Draw a bar graph of the frequency of observed unemployment rates. (*Hint:* Try using the `table` command to create the object you will graph.) Can you go one step further and draw a bar graph of the percentage of time each value is observed?