7 Identifying Drivers of Outcomes: Linear Models

In this chapter we investigate linear models, which are often used in marketing to explore the relationship between an outcome of interest and other variables. A common application in survey analysis is to model satisfaction with a product in relation to specific elements of the product and its delivery; this is called "satisfaction drivers analysis." Linear models are also used to understand how price and advertising are related to sales, and this is called "marketing mix modeling." There are many other situations in which it is helpful to model an outcome, known formally as a *response* or *dependent* variable, as a function of predictor variables (also known as *explanatory* or *independent* variables). Once a relationship is estimated, one can use the model to make predictions or forecasts of the likely outcome for other values of the predictors.

In this chapter, we illustrate linear modeling with a satisfaction drivers analysis using survey data for customers who have visited an amusement park. In the survey, respondents report their levels of satisfaction with different aspects of their experience, and their overall satisfaction. Marketers frequently use this type of data to figure out what aspects of the experience *drive* overall satisfaction, asking questions such as, "Are people who are more satisfied with the rides also more satisfied with their experience overall?" If the answer to this question is "no," then the company will know to invest in improving other aspects of the experience.

An important thing to understand is that *driver* does *not* imply causation. A linear model only assumes an association among variables. Consider a survey of automobile purchasers that finds a positive association between satisfaction and price paid. If a brand manager wants customers to be more satisfied, does this imply that she should raise prices? Probably not. It is more likely that price is associated with higher quality, which then leads to higher satisfaction. Results should be interpreted cautiously and considered in the context of domain knowledge.

© Springer International Publishing Switzerland 2015
C. Chapman, E.M. Feit, *R for Marketing Research and Analytics*, Use R!,
DOI 10.1007/978-3-319-14436-8_7

Linear models are a core tool in statistics, and R provides an excellent set of functions for estimating them. As in other chapters, we review the basics and demonstrate how to conduct linear modeling in R, yet the chapter does not review everything that one would wish to know in practice. We encourage readers who are unfamiliar with linear modeling to supplement this chapter with a review of linear modeling in a statistics or marketing research textbook, where it might appear under a name such as *regression analysis*, *linear regression*, or *least-squares fitting*.

7.1 Amusement Park Data

In this section, we simulate data for a hypothetical survey of visitors to an amusement park. This data set comprises a few objective measures: whether the respondent visited on a weekend (which will be the variable `weekend` in the data frame), the number of children brought (`num.child`), and distance traveled to the park (`distance`). There are also subjective measures of satisfaction: expressed satisfaction overall (`overall`) and satisfaction with the rides, games, waiting time, and cleanliness (`rides`, `games`, `wait`, and `clean`, respectively).

Unlike earlier chapters, in this one we recommend that you *skip* the simulation section and download the data. There is no new R syntax, and this will allow you to review the models without knowing the outcome in advance. To download and check:

```
> sat.df <- read.csv("http://goo.gl/HKnl74")
> str(sat.df)
'data.frame': 500 obs. of  8 variables:
 $ weekend  : Factor w/ 2 levels "no","yes": 2 2 1 2 1 1 2 1 1 2 ...
 $ num.child: int  0 2 1 0 4 5 1 0 0 3 ...
 $ distance : num  114.6 27 63.3 25.9 54.7 ...
...
```

If you have the data, skip to Sect. 7.2 for now, and return later to review the simulation code.

7.1.1 Simulating the Amusement Park Data

To start the data simulation, we set the random number seed to make the process repeatable and declare a variable for the number of observations:

```
> set.seed(08226)
> nresp <- 500    # number of survey respondents
```

Our hypothetical survey includes four questions about a customer's satisfaction with different dimensions of a visit to the amusement park: satisfaction with rides (`rides`), games (`games`), waiting times (`wait`), and cleanliness (`clean`), along with a rating of overall satisfaction (`overall`). In such surveys, respondents often answer similarly on all satisfaction questions; this is known as the *halo effect*.

We simulate a satisfaction halo with a random variable for each customer, `halo`, that does not appear in the final data but is used to influence the other ratings:

```
> halo <- rnorm(n=nresp, mean=0, sd=5)
```

We generate responses for the satisfaction ratings by adding each respondent's halo to the value of another random variable that is specific to the survey item (satisfaction with rides, cleanliness, and so forth).

We add a constant just to adjust the range slightly, and convert the continuous values to integers using `floor()`. This gives us a final value for each satisfaction item on a 100-point scale. Although scales rating 1–5, 1–7, or 1–11 may be more common in practice, such discrete scales introduce complications that we discuss in 7.9; those would detract from our presentation here. So we assume that the data comes from a 100-point scale. Such near-continuous values might be obtained by measuring where respondents mark levels of satisfaction along a line on paper or by touching a screen.

Creating the `nresp` responses can be done in just one line per variable:

```
> rides <- floor(halo + rnorm(n=nresp, mean=80, sd=3)+1)
> games <- floor(halo + rnorm(n=nresp, mean=70, sd=7)+5)
> wait <- floor(halo + rnorm(n=nresp, mean=65, sd=10)+9)
> clean <- floor(halo + rnorm(n=nresp, mean=85, sd=2)+1)
```

By adding `halo` to the response for each question, we create positive correlation between the responses. The constants +1, +5, and +9 are arbitrary to adjust the ranges just for appearance. You can verify the correlation between variables that share the halo by using `cor()`:

```
> cor(rides, games)
[1] 0.4551851
```

Satisfaction surveys often include other questions related to the customer experience. For the amusement park data, we include whether the visit was on a weekend, how far the customer traveled to the park in miles, and the number of children in the party. We generate this data using two functions: `rlnorm(n, meanlog, sdlog)` to sample a lognormal distribution for `distance`, and `sample(x, size, replace)` to sample discrete distributions for `weekend` and number of children (`num.child`):

```
> distance <- rlnorm(n=nresp, meanlog=3, sdlog=1)
> num.child <- sample(x=0:5, size=nresp, replace=TRUE,
+                     prob=c(0.3, 0.15, 0.25, 0.15, 0.1, 0.05))
> weekend <- as.factor(sample(x=c("yes", "no"), size=nresp, replace=TRUE,
+                             prob=c(0.5,0.5)))
```

We create the overall satisfaction rating as a function of ratings for the various aspects of the visit (satisfaction with rides, cleanliness, and so forth), distance traveled, and the number of children:

```
> overall <- floor(halo + 0.5*rides + 0.1*games + 0.3*wait + 0.2*clean +
+                  0.03*distance + 5*(num.child==0) + 0.3*wait*(num.child>0) +
+                  rnorm(n=nresp, mean=0, sd=7) - 51)
```

Although this is a lengthy formula, it is relatively simple with five parts:

1. It includes `halo` to capture the latent satisfaction (also included in `rides` and the other ratings)
2. It adds the satisfaction variables (`rides`, `games`, `wait`, and `clean`) with a weight for each one
3. It includes weighted contributions for other influences such as `distance`
4. There is random normal variation using `rnorm()`
5. It uses `floor()` to produce an integer, with a constant -51 that adjusts the total to be 100-points or less

When a variable like `overall` is a linear combination of other variables plus random noise, we say that it follows a *linear model*. Although these ratings are not a model of real amusement parks, the structure exemplifies the kind of linear model one might propose. With real data, one would wish to discover the contributions from the various elements, which are the weights associated with the various predictors. In the next section, we examine how to fit such a linear model.

Before proceeding, we combine the data points into a data frame and remove unneeded objects from the workspace:

```
> sat.df <- data.frame(weekend, num.child, distance, rides, games, wait, clean,
+                      overall)
> rm(nresp, weekend, distance, num.child, halo, rides, games, wait, clean,
+    overall)
```

7.2 Fitting Linear Models with `lm()`

Every modeling effort should begin with an inspection of the data, so we start with a `summary()` of the data:

```
> summary(sat.df)
 weekend     num.child        distance            rides            games
 no :259   Min.   :0.000   Min.   :  0.5267   Min.   : 72.00   Min.   : 57.00
 yes:241   1st Qu.:0.000   1st Qu.: 10.3181   1st Qu.: 82.00   1st Qu.: 73.00
...
           Max.   :5.000   Max.   :239.1921   Max.   :100.00   Max.   :100.00
      wait           clean           overall
 Min.   : 40.0   Min.   : 74.0   Min.   :  6.00
 1st Qu.: 62.0   1st Qu.: 84.0   1st Qu.: 40.00
...
 Max.   :100.0   Max.   :100.0   Max.   :100.00
```

The data comprise eight variables from a survey of satisfaction with a recent visit to an amusement park. The first three variables describe features of the visit: weekend is a factor with two levels, no and yes; num.child is the number of children in the party, 0–5; and distance is the distance traveled to the park. The remaining five variables are satisfaction ratings for the customers' experience of the rides, games, wait times, cleanliness, and overall experience of the park, on a 100 point scale.

7.2.1 Preliminary Data Inspection

Before modeling, there are two important things to check: that each individual variable has a reasonable distribution, and that joint relationships among the variables are appropriate for modeling.

We do an initial check of the variable distributions and relationships in sat.df using gpairs() as described in Sect. 4.4.2:

```
> gpairs(sat.df)
```

The result is Fig. 7.1, where we see from the histograms that all of the satisfaction ratings are close to normally distributed, but distance has a highly skewed distribution. For most purposes it is a good idea to transform such a variable to a more normal distribution. As we discussed in Sect. 4.5.4, a common transformation for such data is a logarithmic transform; we take the log() of distance and add that to the data frame:

```
> sat.df$logdist <- log(sat.df$distance)
```

We could then run gpairs(sat.df) again (or run hist(sat.df$logdist)) to confirm that the new variable logdist is more normally distributed.

To check the relationships among variables, we examine the bivariate scatterplots shown in Fig. 7.1. They show few concerns apart from the need to transform distance. For example, the pairwise scatterplots of our continuous measures are generally elliptical in shape, which is a good indication that they are appropriate to use in a linear model. One question, however, concerns the fact that the variables in the lower right of Fig. 7.1 are positively correlated.

Why is this a concern? A common issue with marketing data and especially satisfaction surveys is that variables may be highly correlated with one another. Although we as marketers care about individual elements of customers' experiences such as their amusement park experience with rides and games, when completing a survey, the respondents might not give independent ratings to each of those items. They may instead form an overall *halo* rating and rate individual elements of the experience in light of that overall feeling.

When variables are strongly related in this way, it is difficult to assess their individual effects with statistical models. As we will see in Sect. 9.1, the effect can be so

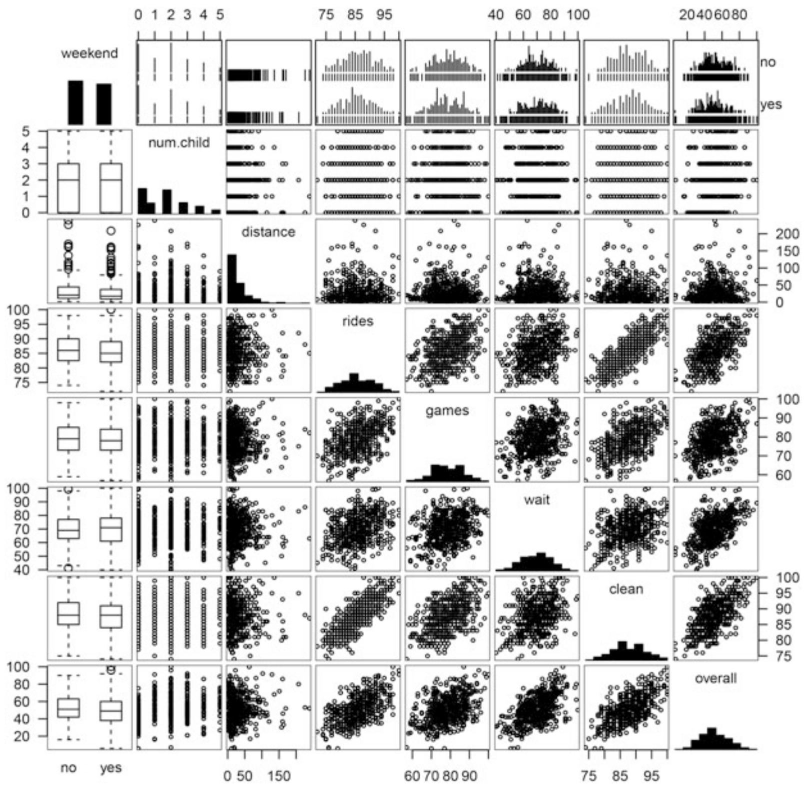

Fig. 7.1. An inspection of data using `gpairs()` before we perform further modeling. This reveals that `distance` has a highly skewed distribution and should be transformed before modeling. Additionally, several variables are positively associated and should be examined further for the strength of association.

severe that the relationships become uninterpretable without taking some action to handle the high correlations.

Given the positive associations shown in Fig. 7.1, we investigate the correlation structure further using `cor()` and `corrplot()` as demonstrated in Sect. 4.5.2:

```
> corrplot.mixed(cor(sat.df[ , c(2, 4:9)]), upper="ellipse")
```

We selected columns `c(2, 4:9)` to exclude the categorical variable `weekend` and the raw variable `distance` that we transformed as `logdist`. The result is the correlation plot shown in Fig. 7.2. We see that the satisfaction items are moderately to strongly associated with one another. However, none of the items appear to be nearly identical, as would be indicated by correlations exceeding $r > 0.8$ for several of them, or $r > 0.9$ for particular pairs. Thus, on an initial inspection, it appears to be acceptable to proceed with modeling the relationships among these variables.

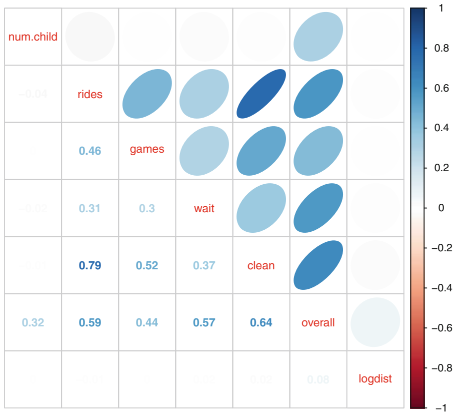

Fig. 7.2. A correlation plot for the amusement park data. Inspection of the item associations is always recommended before linear modeling, in order to check for extremely high correlations between items (such as $r > 0.9$). In the present data, `rides` and `clean` are highly related ($r = 0.79$) but not so strongly that remediation is strictly required.

In Chap. 9 we discuss how to assess this question in more detail and what to do when high correlations pose a more significant problem. In Chap. 8 we discuss strategies to find underlying dimensions that appear in highly correlated data.

7.2.2 Recap: Bivariate Association

The goal of a satisfaction drivers analysis is to discover relationships between customers' satisfaction with features of the service (or product) and their overall experience. For example, to what extent is satisfaction with the park's rides related to overall experience? Is the relationship strong or weak? One way to assess this is to plot those two variables against each other as we did in Chap. 4:

```
> plot(overall~rides, data=sat.df,
+      xlab="Satisfaction with Rides", ylab="Overall Satisfaction")
```

This creates a plot similar to the one in Fig. 7.3, except that it does not include the blue line (but we'll get to that soon). The points on the plot show that there is a tendency for people with higher satisfaction with rides to also have higher overall satisfaction.

7.2.3 Linear Model with a Single Predictor

A linear model estimates a best fit line through the cloud of points. The function to estimate a linear model is `lm(formula, data)`, where `data` is a data frame

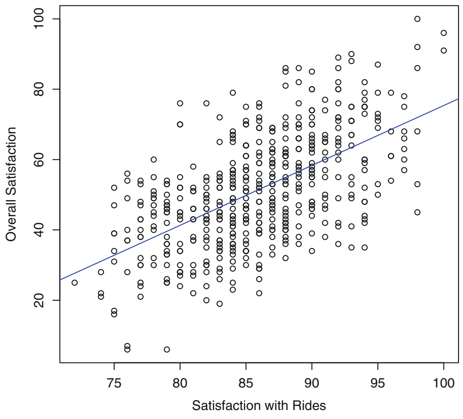

Fig. 7.3. Scatterplot comparing satisfaction with rides to overall satisfaction among recent visitors to an amusement park.

containing the data and `formula` is an R formula, as we saw in Sect. 6.5 for `anova()`. To estimate a linear model relating overall satisfaction to satisfaction with rides, we write:

```
> lm(overall ~ rides, data=sat.df)
...
Coefficients:
(Intercept)         rides
    -94.962         1.703
```

The formula above can be read as "`overall` varies with `rides`." When we call `lm()`, R finds a line that best fits the relationship of `sat.df$rides` and `sat.df$overall`. In the output, R repeats the model for reference and reports two `coefficients`, which are the intercept and the slope of the fitted line. Those can be used to determine the best estimate for any respondent's report of `overall` based on knowing his or her value for `rides`. For example, from this model we would expect that a customer who gives a rating of 95 for satisfaction with `rides` would give an `overall` rating of:

```
> -94.962 + 1.703*95
[1] 66.823
```

Using coefficients manually is not very efficient. This brings us to our next topic, `lm` objects.

7.2.4 `lm` Objects

Like most other R functions, `lm()` returns an object that we can save and use for other purposes. Typically, we assign the result of `lm()` to an object that is used in subsequent lines of code. For example, we can assign the result of `lm()` to a new object `m1`:

```
> m1 <- lm(overall ~ rides, data=sat.df)
```

We can then reuse the model by accessing m1. If we redraw the scatterplot for overall ~ rides, we can add the linear fit line using abline(m1):

```
> plot(overall ~ rides, data=sat.df,
+      xlab="Satisfaction with Rides", ylab="Overall Satisfaction")
> abline(m1, col='blue')
```

The result is shown in Fig. 7.3. abline() recognizes that it is dealing with an lm object and uses the slope and the intercept from m1 to draw the line.

We can also inspect the m1 object:

```
> str(m1)
List of 12
 $ coefficients : Named num [1:2] -95 1.7
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "rides"
 $ residuals    : Named num [1:500] -6.22 11.78 11.18 -17.93 19.89 ...
...
```

This shows us that the m1 object is a list with 12 specific members that contain everything lm() knows about the model. (To refresh yourself on list objects, see Chap. 2.) The first element of this list is $coefficients, which you can inspect:

```
> m1$coefficients
(Intercept)       rides
 -94.962246    1.703285
```

You don't have to use the full name m1$coefficients. In many places in R, it works to abbreviate long names, such as m1$coef.

As with other types of R objects, there is a summary() function for lm objects that summarizes features of the fitted model, reporting much more than the short output we saw from lm() above:

```
> summary(m1)
...
Residuals:
    Min      1Q  Median      3Q     Max
-33.597 -10.048   0.425   8.694  34.699

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -94.9622     9.0790  -10.46   <2e-16 ***
rides         1.7033     0.1055   16.14   <2e-16 ***
---
Signif. codes: ***  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1

Residual standard error: 12.88 on 498 degrees of freedom
Multiple R-squared:  0.3434,  Adjusted R-squared:  0.3421
F-statistic: 260.4 on 1 and 498 DF,  p-value: < 2.2e-16
```

This summarizes the principal information to review for a linear model. More advanced models are reported similarly, so it is useful to become familiar with this format. In addition to listing the model that was estimated, we get information about coefficients, residuals, and the overall fit.

The most important section is labeled `Coefficients` and shows the model coefficients in the `Estimate` column. The coefficient for `rides` is 1.70, so each additional rating point for `rides` is estimated to result in an increase of 1.7 points of `overall` rating. (In case you're wondering, the coefficient for the `(Intercept)` shows where the linear model line crosses the y-axis, but this is usually not interpretable in a satisfaction drivers analysis—for instance, there is no such thing as a possible negative rating on our scale—so it is generally ignored by marketing analysts.)

The `Std. Error` column indicates uncertainty in the coefficient estimate, under the assumption that the data are a random sample of a larger population. The "`t value`", *p*-value ("`Pr(>|t|)`"), and significance codes indicate a *Wald test*, which assesses whether the coefficient is significantly different than zero. A traditional estimate of a 95 % confidence interval for the coefficient estimate is that it will fall within $\pm 1.96 \times std.error$. In this case, $1.7033 \pm 1.96 \times 0.1055 = (1.495, 1.910)$. So we are confident—assuming the model is appropriate and the data are representative—that the coefficient for `ride` is 1.495–1.910.

Once again, R does not make you compute things by hand. `confint()` reports confidence intervals:

```
> confint(m1)
                   2.5 %     97.5 %
(Intercept) -112.800120 -77.124371
rides          1.495915   1.910656
```

This confirms our computation by hand that the best estimate for the relationship `overall` $\sim$ `rides` is 1.496–1.911 (with slight differences due to rounding). It is a best practice to report the range of an estimate, not just the single best point.

The `Residuals` section in the `summary(m1)` output tells us how closely the data follow the best fit line. A *residual* is the difference between the model-predicted value of a point and its actual value. In Fig. 7.3, this is the vertical distance between a plotted point (actual value) and the blue line (predicted value).

In the summary of `m1`, we see that the residuals are quite wide, ranging from -33.597 to 34.699, which means our predictions can be quite a bit off for any given data point (more than 30 points on the rating scale). The quartiles of the residuals suggest that they are fairly symmetric around 0. As we discuss in Sect. 7.2.5, that is a good sign that the model is unbiased (although perhaps imprecise).

In the last section of the output, `summary(m1)` provides measures of how well the model fits the data. The first is the *residual standard error*, an estimate of the standard error of the residuals. Like the residuals, this is a measure of how close the data points are to the best estimate line. (You can directly check this by examining the standard deviation of the residuals using `sd(m1$residuals)`, which will be similar.)

The second line reports the estimate of *R-squared*, a measure of how much variation in the dependent variable is captured by the model. In this case, the R-squared is 0.3434, indicating that about a third of the variation in overall satisfaction is explained by variation in satisfaction with rides. When a model includes only a single predictor, R-squared is equal to the square of the correlation coefficient r between the predictor and the outcome:

```
> cor(sat.df$overall, sat.df$rides)^2
[1] 0.3433799
```

Finally, the line labeled `F-statistic:` provides a statistical test of whether the model predicts the data better than simply taking the average of the outcome variable and using that as the single prediction for all the observations. In essence, this test tells whether our model is better than a model that predicts overall satisfaction using no predictors. (For reasons we will not describe in detail, this is the same test reported by the `anova()` function that we saw in Chap. 5; you could find the same value with `anova(m1)`. Check a statistics textbook for a description of the *F-test* in more detail.) In the present case, the `F-statistic` shows a p-value $<<$ .05, so we reject the null hypothesis that a model without predictors performs as well as model `m1`.

7.2.5 Checking Model Fit

Because it is easy to fit linear models, too many analysts fit models and report results without considering whether the models are *reasonable*. However, there are a variety of ways to assess model fit and adequacy that are easy to perform in R. While we can't possibly cover this material comprehensively, we would like to give you a few pointers that will help you assess model adequacy.

There are several assumptions when a linear model is fitted to data. The first is that the relationship between the predictors and the outcomes is *linear*. If the relationship is not linear, then the model will make systematic errors. For example, if we generate data where `y` is a function of the square of `x` and then fit a linear model `y ~ x`, this will draw a straight line through a cloud of points that is curved.

```
> x <- rnorm(500)
> y <- x^2 + rnorm(500)
> toy.model <- lm(y~x)
```

If you inspect the model by typing `summary(toy.model)`, you will see that the fitted coefficient for `x` is -0.01159 and the Wald significance test indicates that the coefficient is not significantly different from zero. Without model checking, a sloppy analyst might conclude that x is not related to y. However, if we plot x versus y and then draw our fitted line on the plot, we can see more clearly what is going on.

```
> plot(y~x)
> abline(toy.model)
```

The resulting plot is shown on the left side of Fig. 7.4. The plot shows that our fitted linear model (illustrated with a blue line) completely misses the curvature in the relationship between x and y.

Another assumption of a linear model is that prediction errors—the parts of the data that do not exactly fit the model—are normally distributed and look like random noise with no pattern. One way to examine this is to plot the model's *fitted values* (the predictions) versus the *residuals* (the prediction errors).

```
> plot(toy.model$fitted.values, toy.model$residuals)
```

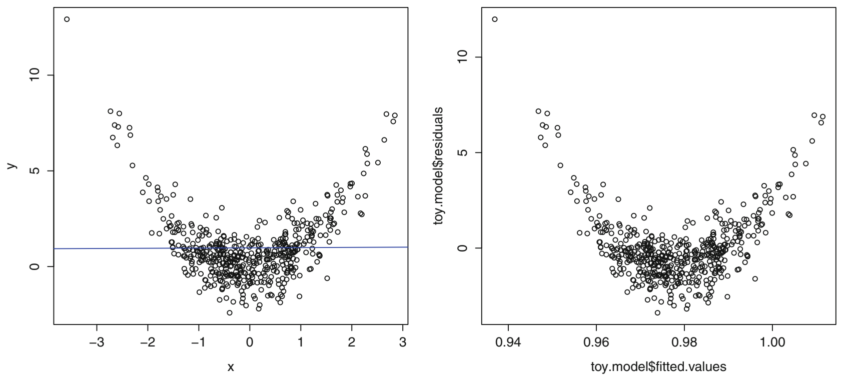

Fig. 7.4. Fitting a linear model when the true relationship is nonlinear (*as shown on the left*) results in unusual residual patterns (*shown on the right*).

This results in the plot on the right side of Fig. 7.4 and you can see from the plot that there is a clear pattern in the residuals: our model under-predicts the value of y near zero and over-predicts far from zero. When you come across this problem in real data, the solution is usually to transform x; you can use the methods described in Sect. 4.5.4 to find a transformation that is suitable. If you begin by inspecting scatterplots as we recommend in Sect. 7.2.1, you will be unlikely to commit such a simple error. Still, it is good to know that later checks can help prevent errors as well.

We can look at this same diagnostic plot for our satisfaction drivers data. R suggests four specific plots to assess the fit of linear model objects and you can look at all four simply by using plot() with any lm object. To see all four plots at once, we type par(mfrow=c(2,2)) first:

```
> par(mfrow=c(2,2))
> plot(m1)
```

In Fig. 7.5, the first plot (in the upper left corner) shows the fitted values versus residuals for `m1`, just as we produced manually for our toy `y ~ x` model. In Fig. 7.5 there is no obvious pattern between the fitted values for overall satisfaction and the residuals; this is consistent with the idea that the residuals are due to random error, and supports the notion that the model is adequate.

The second plot in the lower left of Fig. 7.5 is similar to the first, except that instead of plotting the *raw* residual value, it plots the *square root* of the standardized residual. Again, there should be no clear pattern; if there were it might indicate a nonlinear relationship. Observations with high residuals are flagged as potential outliers, and R labels them with row numbers in case we wish to inspect them in the data frame.

A common pattern in residual plots is a *cone* or *funnel*, where the range of errors gets progressively larger for larger fitted values. This is called *heteroskedasticity* and is a violation of linear model assumptions. A linear model tries to maximize fit to the line; when values in one part of the range have a much larger spread than those in another area, they have undue influence on the estimation of the line. Sometimes a transformation of the predictor or outcome variable will resolve heteroskedasticity (see Sect. 4.5.3).

The third result of `plot()` for `lm` objects is a *Normal QQ plot*, as in the upper right of Fig. 7.5. A QQ plot helps you see whether the residuals follow a normal distribution, another key assumption (see Sect. 3.4.3). It compares the values that residuals would be *expected* to take if they are normally distributed, versus their actual values. When the model is appropriate, these points are similar and fall close to a diagonal line; when the relationship between the variables is nonlinear or otherwise does not match the assumption, the points deviate from the diagonal line. In the present case, the QQ plot suggests that the data fits the assumption of the model.

The final plot in the lower right panel of Fig. 7.5 again helps to identify potential *outliers*, observations that may come from a different distribution than the others. Outliers are a problem because, if they are far from other points, they unduly influence the fitted line. We do not want one or a very few observations to have a large effect on the coefficients. The lower right plot in Fig. 7.5 plots the *leverage* of each point, a measure of how much influence the point has on the model coefficients. When a point has a high residual and high leverage, it indicates that the point has both a different pattern (residual) and undue influence (leverage). One measure of the leverage of a data point is *Cook's distance*, an estimate of how much predicted (y) values would change if the model were re-estimated with that point eliminated from the data. If you have observations with high Cook's distance, this chart would show dotted lines for the distances; in the present case, there are none.

Still, in the lower right of Fig. 7.5, three points are automatically labeled with row numbers because they are potentially problematic outliers based on high standardized residual distance and leverage on the model. We do not recommend routinely

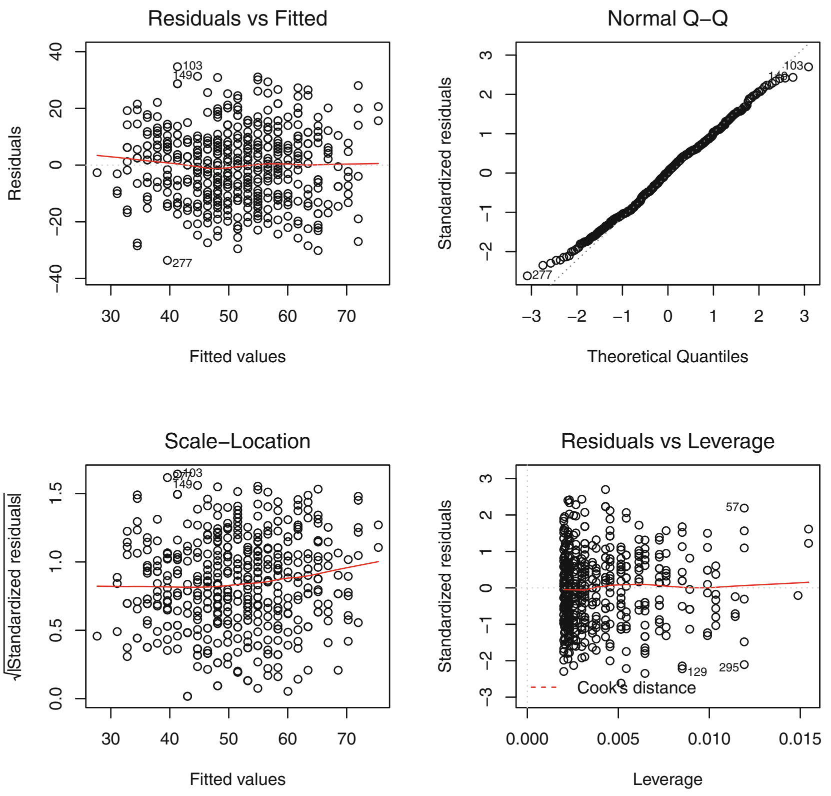

Fig. 7.5. Diagnostic plots for the model relating overall satisfaction to satisfaction with rides.

removing outliers, yet we do recommend to inspect them and determine whether there is a problem with the data. We inspect the identified points by selecting those rows:

```
> sat.df[c(57, 129, 295),]
    weekend num.child distance rides games wait clean overall  logdist
57      yes         2 63.29248    98    87   89   100     100 4.147767
129     yes         0 11.89550    76    77   51    77       6 2.476161
295      no         0 11.74474    98    83   63    92      45 2.463406
```

In this case, none of the data points is obviously invalid (for instance, with values below 1 or greater than 100), although row 129 might be checked for input correctness; an overall rating of 6 on the survey would be unusual although perhaps accurate. We generally do not omit outliers except when they represent obvious errors in the data. In the present case, we would keep all of the observations.

Overall, Fig. 7.5 looks good and suggests that the model relating overall satisfaction to satisfaction with rides is reasonable.

But we've only examined a single variable so far. In the next section, we consider multiple predictors. For brevity, in the following sections we omit the checks of model adequacy that were shown in this section, but we encourage you to check and interpret `plot()` for the models.

7.3 Fitting Linear Models with Multiple Predictors

Now that we've covered the basics of linear models using just one predictor, we turn to the problem of assessing multiple drivers of satisfaction. Our goal is to sort through all of the features of the park—rides, games, wait times, and cleanliness—to determine which ones are most closely related to overall satisfaction.

To estimate our first multiple variable model, we call `lm` with a formula describing the model:

```
> m2 <- lm(overall ~ rides + games + wait + clean, data=sat.df)
> summary(m2)
...
Residuals:
    Min      1Q  Median      3Q     Max
-29.944  -6.841   1.072   7.167  28.618

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -131.40919    8.33377 -15.768  < 2e-16 ***
rides          0.52908    0.14207   3.724 0.000219 ***
games          0.15334    0.06908   2.220 0.026903 *
wait           0.55333    0.04781  11.573  < 2e-16 ***
clean          0.98421    0.15987   6.156 1.54e-09 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1

Residual standard error: 10.59 on 495 degrees of freedom
Multiple R-squared:  0.5586,  Adjusted R-squared:  0.5551
F-statistic: 156.6 on 4 and 495 DF,  p-value: < 2.2e-16
```

Looking first at the model fit statistics at the bottom of the output, we see that our prediction was improved by including all the satisfaction items in the model. The R-squared increased to 0.5586, meaning that about half of the variation in overall ratings is explained by the ratings for specific features. The residual standard error is now 10.59, meaning that the predictions are more accurate. Our residuals also appear to be symmetric. As noted above, we recommend also to inspect the model using `plot()` to confirm that there are no patterns in the residuals indicative of nonlinearity or outliers, although we omit that step here.

Next we examine the model coefficients. Each coefficient represents the strength of the relationship between satisfaction with that feature and overall satisfaction, conditional on the values of the other predictors. All four features are identified as being statistically significant (p-value, shown as `Pr(>|t|)`, $< .05$). Rather than just comparing the numbers in the output, it can be helpful to visualize the coefficients. We use the `coefplot` package [99] to do this, calling `coefplot()` for our model, and adding `intercept=FALSE` to plot just the individual item coefficients:

```
> library(coefplot)    # install if necessary
> coefplot(m2, intercept=FALSE, outerCI=1.96, lwdOuter=1.5,
+          ylab="Rating of Feature",
+          xlab="Association with Overall Satisfaction")
```

We use `coefplot()` arguments to set the outer confidence interval to a width of 1.96 standard errors (using `outerCI=1.96`, which corresponds to a 95 % confidence interval) and to increase the size of the plotted lines slightly with `lwdOuter=1.5`.

The result is shown in Fig. 7.6 where we see that satisfaction with cleanliness is estimated to be the most important feature associated with overall satisfaction, followed by satisfaction with the rides and wait times. Satisfaction with games is estimated to be relatively less important.

A plot of coefficients is often a key output from a satisfaction drivers analysis. Sorting the plot so that the coefficients are in order based on their estimated coefficient may make it easier to quickly identify the features that are most closely related to overall satisfaction if you have a large number of predictors.

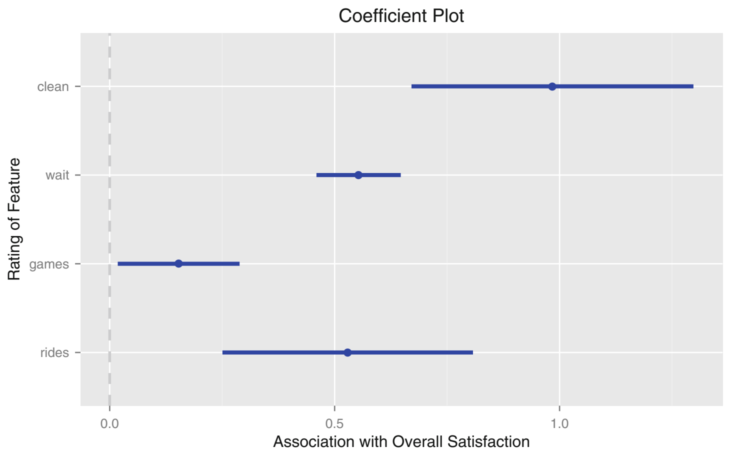

Fig. 7.6. A coefficient plot produced with `coefplot()` for an initial multivariate `lm()` model of satisfaction in the amusement park data. In the model, satisfaction with cleanliness is most strongly associated with overall satisfaction, and rides and wait times are also associated.

7.3.1 Comparing Models

Now that we have two model objects, `m1` and `m2` we might ask which one is better. One way to evaluate models is to compare their R-squared values.

```
> summary(m1)$r.squared
[1] 0.3433799
> summary(m2)$r.squared
[1] 0.558621
```

Based on the R-squared values we can say that `m2` explains more of the variation in satisfaction than `m1`. However, a model with more predictors usually has a higher R^2, so we could instead compare *adjusted* R-squared values, which control for the number of predictors in the model.

```
> summary(m1)$adj.r.squared
[1] 0.3420614
> summary(m2)$adj.r.squared
[1] 0.5550543
```

The adjusted R-squared still suggests that the `m2` explains more of the variation in overall satisfaction, even accounting for the fact that `m2` uses more predictors.

To compare the predictions of the models visually, we plot the fitted versus actual values for each:

```
> plot(sat.df$overall, fitted(m1), col='red',
+      xlim=c(0,100), ylim=c(0,100),
+      xlab="Actual Overall Satisfaction", ylab="Fitted Overall Satisfaction")
> points(sat.df$overall, fitted(m2), col='blue')
> legend("topleft", legend=c("model 1", "model 2"),
+         col=c("red", "blue"), pch=1)
```

If the model fits the data perfectly, it would fall along a 45° line in this plot, but, of course, it is nearly impossible to fit customer satisfaction data perfectly. By comparing the red and the blue points in the resulting plot in Fig. 7.7, you can see that the blue cloud of points is more tightly clustered along a diagonal line, which shows that `m2` explains more of the variation in the data than `m1`.

For a more formal test, which is possible because the models here are nested (see Sect. 6.5.1), we can use `anova()` function to determine whether `m2` explains more of the variation than `m1`:

```
> anova(m1, m2)
Analysis of Variance Table

Model 1: overall ~ rides
Model 2: overall ~ rides + games + wait + clean
  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
1    498 82612
2    495 55532  3     27080 80.463 < 2.2e-16 ***
```

The low *p*-value indicates that the additional predictors in `m2` significantly improve the fit of the model. If these two models were the only ones under consideration, we would interpret `m2` instead of `m1`.

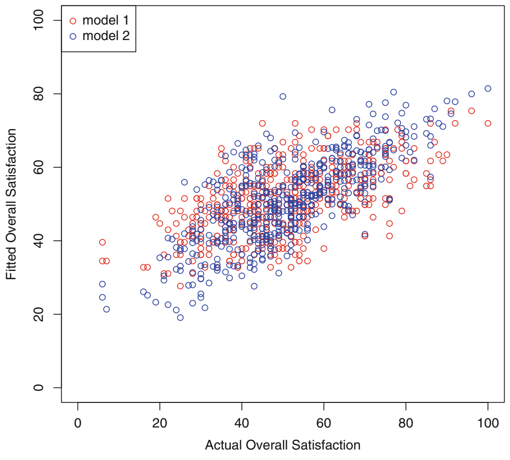

Fig. 7.7. Comparison of fitted versus actual values for linear models `m1` and `m2`.

We should also point out that the coefficient for `rides` changed from `m1` to `m2`. The value in `m1` was $1.70 \times rides$, while in `m2` it is $0.529 \times rides$. Why is this happening? The reason is because `rides` is not independent of all the other variables; Fig. 7.1 shows that customers who are more satisfied with the rides tend to be more satisfied with the wait times and games. When those variables are added as predictors in model `m2`, they now perform some of the work in predicting the overall rating, and the contribution of `rides` is a smaller share of the total model.

Neither coefficient nor `rides` is more correct in itself because a coefficient is not right or wrong but part of a larger model. Which model is preferable? Because model `m2` has better overall fit, we would interpret its coefficient for `rides`, but only in the context of the total model. In the sections below, we see that as the structure of a model changes, the coefficients generally change as well (unless the variables are entirely uncorrelated).

7.3.2 Using a Model to Make Predictions

As we saw for the single variable case, we could use the model coefficients to predict the `overall` outcome for different combinations of the explanatory variables. For example, if we wanted to predict the overall rating for a customer who rated the four separate aspects as 100 points each, we could multiply those ratings by the coefficients and add the intercept:

```
> coef(m2)["(Intercept)"] + coef(m2)["rides"]*100 + coef(m2)["games"]*100 +
+     coef(m2)["wait"]*100 + coef(m2)["clean"]*100
```

```
(Intercept)
   90.58612
```

The best estimate is 90.586 using model `m2`. Because `coef(m2)` is a named vector, we access the individual coefficients here using their names.

The prediction equation above is clunky to type, and there are more efficient ways to compute model predictions. One way is to use matrix operations to multiply coefficients by a vector of predictor values:

```
> coef(m2)%*%c(1, 100, 100, 100, 100)
          [,1]
[1,] 90.58612
```

We could also use `predict(object, newdata)` where `newdata` is a data frame with the same column names as the data that was used to estimate the model. For example, if we want to find the predictions for the first ten customers in our data set, we would pass the first ten rows of `sat.df` to `predict`:

```
> predict(m2, sat.df[1:10,])
       1        2        3        4        5        6        7 ...
46.60864 54.26012 51.17289 50.30434 52.94625 27.87214 36.27435 ...
```

This predicts satisfaction for the first ten customers. The predictions for observations used to estimate the model are also stored in the model object, and can be accessed with `fitted()`:

```
> fitted(m2)[1:10]
       1        2        3        4        5        6        7 ...
46.60864 54.26012 51.17289 50.30434 52.94625 27.87214 36.27435 ...
```

7.3.3 Standardizing the Predictors

Thus far, we have interpreted raw coefficients in order to evaluate the contributions of ratings on the shared 100-point scale. However, if the variables have different *scales*, such as a survey where `rides` is rated on a 1–10 scale while cleanliness is rated 1–5 scale, then their coefficient values would not be directly comparable. In the present data, this occurs with the `distance` and `logdist` variables, which are not on a 100-point scale.

When you wish to compare coefficients, it can be helpful to *standardize* data on a common scale before fitting a model (and *after* transforming any variables to a more normal scale). The most common standardization converts values to zero-centered *units of standard deviation*. This subtracts a variable's `mean` from each observation and then divides by the standard deviation (`sd()`). This could be done using math, such as:

```
> (sat.df$rides - mean(sat.df$rides)) / sd(sat.df$rides)
  [1]  0.21124774  0.21124774 -0.15486620  0.39430471 -0.33792317  ...
```

This process is so common that R includes the `scale()` function to perform it:

```
> scale(sat.df$rides)
                [,1]
  [1,]  0.21124774
  [2,]  0.21124774
  [3,] -0.15486620
...
```

In the remainder of the chapter, we do not want to worry about the scale of our variables, only their relative contributions, so we create a scaled version of `sat.df` called `sat.std`:

```
> sat.std <- sat.df[ , -3]  # sat but remove distance
> sat.std[ , 3:8] <- scale(sat.std[ , 3:8])
> head(sat.std)
  weekend num.child      rides       games         wait       clean    overall
1     yes         0  0.2112477 -0.69750817 -0.918784090  0.21544189 -0.2681587
2     yes         2  0.2112477 -0.08198737  0.566719693 -0.17555973  0.8654385
3      no         1 -0.1548662  0.16422095  0.009655775  0.01994108  0.6135280
...
```

In this code, we first copied `sat.df` to the new data frame `sat.std`, dropping the untransformed values of `distance` with `[ , -3]` because we use `logdist` instead. Then we standardized each of the numeric columns. We do not standardize `weekend` because it is a factor variable rather than numeric. We leave `num.child` as is for now because we have not yet analyzed it.

Note that we do not alter the original data frame `sat.df` when standardizing it. Instead, we copy it to a new data frame and alter the new one. This process makes it easier to recover from errors; if anything goes wrong with `sat.std`, we can just run these few commands again to recreate it.

The question of standardizing values depends primarily on how you want to use a model's coefficients. If you want to interpret coefficients in terms of the original scales, then you would not standardize data first. However, in driver analysis we are usually more concerned with the relative contribution of different predictors and wish to compare them, and standardization assists with this. Additionally, we often transform variables before analysis such that they are no longer on the original scale.

After standardizing, you should check the results. A standardized variable should have a mean of 0 and values within a few units of the mean. Checking the `summary()`:

```
> summary(sat.std)
 weekend     num.child           rides.V1              games.V1
 no :259   Min.   :0.000   Min.   :-2.5346068   Min.   :-2.6671747
 yes:241   1st Qu.:0.000   1st Qu.:-0.7040371   1st Qu.:-0.6975082
           Median :2.000   Median : 0.0281908   Median :-0.0819874
           Mean   :1.738   Mean   : 0.0000000   Mean   : 0.0000000
...
```

We see that sat.std matches expectation. Note that the column names from summary() have an extra .V1 in the output; this indicates that the column has a more complex data type than a simple vector. Specifically, scale() converts objects to one-dimensional matrices (instead of vectors). This has no significance for our model fitting; we just have to be aware of the occasionally confusing addition to the names.

There is a technical point we should mention when standardizing variables. If the outcome and predictors are all standardized, their means will be zero and thus the intercept will be zero. However, that does *not* imply that the intercept could be removed from the model. The model is estimated to minimize error in the overall fit, which includes error for the intercept. This implies that the intercept should remain in a model after standardization if it would be there otherwise (as it usually should be; see Sect. 7.5.1).

7.4 Using Factors as Predictors

While m2 above was reasonable, we can continue to improve it. It is typical to try many models before arriving at a final one.

For the next step, we wonder whether satisfaction is different for customers who come on the weekend, travel farther, or have more children. We add these predictors to the model using the standardized data:

```
> m3 <- lm(overall ~ rides + games + wait + clean +
+                    weekend + logdist + num.child, data = sat.std)
> summary(m3)
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.37271    0.04653  -8.009 8.41e-15 ***
rides        0.21288    0.04197   5.073 5.57e-07 ***
games        0.07066    0.03026   2.335   0.0199 *
wait         0.38138    0.02777  13.734  < 2e-16 ***
clean        0.29690    0.04415   6.725 4.89e-11 ***
weekendyes  -0.04589    0.05141  -0.893   0.3725
logdist      0.06470    0.02572   2.516   0.0122 *
num.child    0.22717    0.01711  13.274  < 2e-16 ***
...
Multiple R-squared:  0.6786,  Adjusted R-squared:  0.674
F-statistic: 148.4 on 7 and 492 DF,  p-value: < 2.2e-16
```

The model summary shows a substantial improvement in fit (R-squared of 0.6786) and the coefficients for logdist and num.child are significantly greater than zero, suggesting that people who travel further and have more children have higher overall satisfaction ratings.

Notice that the coefficient for weekend is labeled weekendyes, which seems a bit unusual. Recall that weekend is a factor variable, but a factor doesn't fit naturally in our linear model; you can't multiply yes by a number. R handles this by converting the data to a numeric value where 1 is assigned to the value of `yes` and 0 to `no`. It labels the output so that we know which direction the coefficient applies to. So, we can interpret the coefficient as meaning that on average those who come on the weekend rate their overall satisfaction −0.046 standard units (standard deviations) lower than those who come on a weekday. A convenient feature of R is that it does this automatically for factor variables, which are common in marketing.

In fact, we used a linear model with a factor as a predictor in Chap. 5, when we compared groups using ANOVA. An ANOVA model is a linear model with a factor as a predictor, and the command we learned in Chap. 5, `aov()`, internally calls `lm()` to fit the model. `aov(overall ~ weekend, data=sat.std)` and `lm(overall ~ weekend, data=sat.std)` fit the same model, although the result is reported differently because of tradition.

If they are the same, which should one use? We generally prefer to use `lm` because it is a more flexible method and allows us to include both numeric and factor predictors in the same model. (For those of you who were wondering, this explains why we used the linear modeling function `lmBF` to fit a Bayesian ANOVA model in Chap. 5.)

When your data includes factors, you must be careful about the data type. For example, `num.child` is a numeric variable, ranging 0–5, but it doesn't necessarily make sense to treat it as a number, as we did in `m3`. In doing so, we implicitly assume that satisfaction goes up or down linearly as a function of the number of children, and that the effect is the same for each additional child. (Anyone who has taken a group of children to an amusement park might guess that this is an unreasonable assumption.)

We correct this by converting `num.child` to a factor and re-estimating the model:

```
> sat.std$num.child.factor <- factor(sat.std$num.child)
> m4 <- lm(overall ~ rides + games + wait + clean +
+                   weekend + logdist + num.child.factor, data=sat.std)
> summary(m4)
...
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       -0.69100    0.04488 -15.396  < 2e-16 ***
rides              0.22313    0.03541   6.301 6.61e-10 ***
...
num.child.factor1  1.01610    0.07130  14.250  < 2e-16 ***
num.child.factor2  1.03732    0.05640  18.393  < 2e-16 ***
num.child.factor3  0.98000    0.07022  13.955  < 2e-16 ***
num.child.factor4  0.93154    0.08032  11.598  < 2e-16 ***
num.child.factor5  1.00193    0.10369   9.663  < 2e-16 ***
...
Multiple R-squared:  0.7751,  Adjusted R-squared:   0.77
F-statistic: 152.9 on 11 and 488 DF,  p-value: < 2.2e-16
```

We now see that there are five fitted coefficients for `num.child.factor`: one for parties with one child, one for parties with two children, etc. There is not a coefficient for `num.child.factor0`, because it is the baseline level to which the other coefficients are added when they apply. We interpret each coefficient as the difference between that level of the factor and the baseline level. So, parties with 1 child rate their overall satisfaction on average 1.016 standard deviations higher than parties without children.

Internally, R has created a new variable `num.child.factor1` that is equal to 1 for those cases where `num.child.factor` represents one child (a factor level of "1"), and is 0 otherwise. Similarly, `num.child.factor2` is 1 for cases with two children, and 0 otherwise, and so forth. The coefficient for `num.child.factor2` is 1.037, meaning that people with two children rate their overall satisfaction on average a full standard deviation higher than those with no children.

A striking thing about `m4` is that the increase in overall satisfaction is about the same regardless of how many children there are in the party—about one standard deviation higher for any number of children. This suggests that we don't actually need to estimate a different increase for each number of children. In fact, if the increase is the same for one child as for five children, attempting to fit a model that scales increasingly per child would result in a less accurate estimate.

Instead, we declare a new variable called `has.child` that is `TRUE` when the party has children in it and `FALSE` when the party does `not` have children. We then estimate the model using that new factor variable. We also drop `weekend` from the model because it doesn't seem to be a significant predictor:

```
> sat.std$has.child <- factor(sat.std$num.child > 0)
> m5 <- lm(overall ~ rides + games + wait + clean + logdist + has.child,
+                    data=sat.std)
> summary(m5)
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -0.70195    0.03906 -17.969  < 2e-16 ***
rides          0.22272    0.03512   6.342 5.12e-10 ***
...
has.childTRUE  1.00565    0.04683  21.472  < 2e-16 ***
...
Multiple R-squared:  0.7741,  Adjusted R-squared:  0.7713
F-statistic: 281.5 on 6 and 493 DF,  p-value: < 2.2e-16
```

Is this still a good model? The change in R-squared between model `m4` and `m5` is negligible, suggesting that our simplification did not deteriorate the model fit.

Model `m5` estimates overall satisfaction to be about one standard deviation higher for parties with children. However, one might now wonder how children influence other aspects of the ratings. For instance, is the relationship between satisfaction and waiting times different for parties with and without children? One might guess from experience that wait time would be more important to parties with children. To explore this question, we need to incorporate *interactions* into the model.

7.5 Interaction Terms

We can include an interaction of two terms by using the `:` operator between variables in a formula. For instance, to estimate `overall` as a function of `rides` plus the interaction of `wait` and `has.child`, we could write the formula as `overall ~ rides + wait:no.child`. There are other ways in R to write interaction terms (see Sect. 7.5.1) but we prefer to specify them explicitly in this way.

We create a new model with interactions between the satisfaction ratings and two variables that describe the visit: `no.child` and `weekend`:

```
> m6 <- lm(overall ~ rides + games + wait + clean +
+                    weekend + logdist + has.child +
+                    rides:has.child + games:has.child + wait:has.child +
+                    clean:has.child + rides:weekend + games:weekend +
+                    wait:weekend + clean:weekend, data=sat.std)
> summary(m6)
...
Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
...
rides:has.childTRUE  0.057837   0.073070   0.792  0.42902
games:has.childTRUE -0.064043   0.052797  -1.213  0.22572
wait:has.childTRUE   0.350649   0.047241   7.423 5.21e-13 ***
clean:has.childTRUE -0.001854   0.079710  -0.023  0.98146
rides:weekendyes     0.061784   0.067750   0.912  0.36225
games:weekendyes     0.018511   0.049036   0.377  0.70597
wait:weekendyes      0.035168   0.044463   0.791  0.42936
clean:weekendyes    -0.027305   0.071005  -0.385  0.70074
...
```

The model object `m6` now includes eight interaction terms between ratings for features of the park and `no.child` and `weekend`. Only one of these interactions is significant: the `wait:no.child` interaction. This suggests we could drop the non-significant interactions to create a new model `m7`:

```
> m7 <- lm(overall ~ rides + games + wait + clean + logdist + has.child +
+                    wait:has.child, data=sat.std)
> summary(m7)
...
Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        -0.69316    0.03684 -18.814  < 2e-16 ***
rides               0.21264    0.03313   6.419 3.24e-10 ***
games               0.04870    0.02394   2.034   0.0425 *
wait                0.15095    0.03688   4.093 4.98e-05 ***
clean               0.30244    0.03485   8.678  < 2e-16 ***
logdist             0.02919    0.02027   1.440   0.1504
has.childTRUE       0.99830    0.04416  22.606  < 2e-16 ***
wait:has.childTRUE  0.34688    0.04380   7.920 1.59e-14 ***
...
Multiple R-squared:  0.7996,  Adjusted R-squared:  0.7968
F-statistic: 280.5 on 7 and 492 DF,  p-value: < 2.2e-16
```

In these results, we see that attending the park with children is a predictor of higher satisfaction, and waiting time is more important predictor among those with children (`wait:has.childTRUE`) than those without children. We don't know the reason for this, but perhaps children go on more rides and their parents are therefore more influenced by wait times.

One might further tune the model by considering whether `logdist` is still needed; we'll leave that to the reader and assume that model `m7` is the final model.

What do we do with these results as marketers? We identify several possible marketing interventions. If we want to increase satisfaction overall, we could perhaps do so by trying to increase the number of visitors with children. Alternatively, if we want to appeal to visitors without children, we might engage in further research to understand why their ratings are lower. If we are allocating budget to personnel, the importance of cleanliness suggests continuing to allocate resources there (as opposed, say, to games). We might also want to learn more about the association between children and waiting time, and whether there are things we could do to make waiting less frequent or more enjoyable.

There are many more such questions one could pose from results like these; a crucial step in analysis is to think carefully about the implications and where one might be able to make a product or market intervention. When considering actions to take, it is especially important to remember that the model assesses association, not causation. Possible changes in outcome should be viewed as hypotheses suggested by the model, to be confirmed separately.

To share these results with others, it is helpful to create a new satisfaction drivers plot using `coefplot()`:

```
> library(coefplot)    # install if needed
> coefplot(m7, intercept=FALSE, outerCI=1.96, lwdOuter=1.5,
+           ylab="Rating of Feature",
+           xlab="Association with Overall Satisfaction")
```

The result is Fig. 7.8 summarizing the relative contribution of each element on overall satisfaction.

When including interaction terms in a model, there are two important points. First, it is especially important to consider standardizing the predictors when modeling interactions in order to have an interpretable and comparable scale for coefficients. Second, one should always include main effects (such as `x + y`) when including an interaction effect (`x:y`). If you don't estimate the main effects, you won't know whether a purported interaction is in fact due to an interaction, or is instead due to one of the individual variables' unestimated main effects.

7.5.1 Language Brief: Advanced Formula Syntax*

This section is optional for those who wish to construct more complex formulas with interaction effects. As in the examples above, we generally write formulas

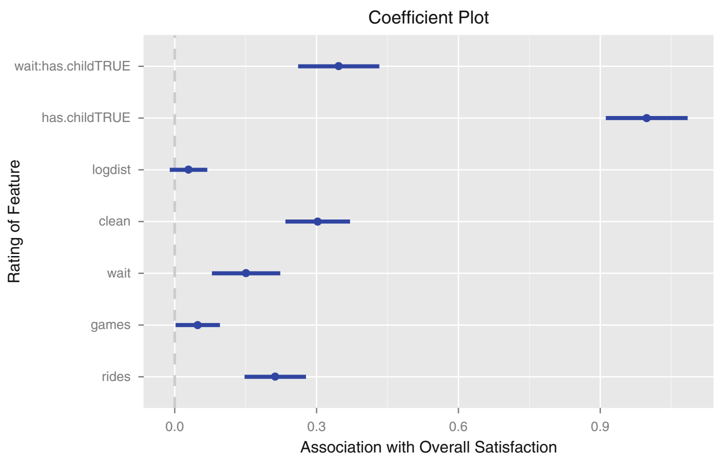

Fig. 7.8. Satisfaction drivers for visitors to an amusement park (simulated). The model reveals that the variable most strongly (and positively) associated with satisfaction is visiting the park with children. Satisfaction with waiting time is a stronger predictor of overall satisfaction among visitors with children than those without, as shown in the `wait:has.childTRUE` interaction. Of the individual park features, satisfaction with cleanliness is most associated with overall satisfaction.

using only + (for main effects) and : (specific interactions), but the following may help create more compact formulas when you have many variables or interactions.

As we've seen, you can include an interaction between `x` and `z` by including `x:z` in the formula. If you want to include two variables along with their interaction, you can use `x*z`, which is the equivalent to writing `x + z + x:z`.

To include *all* of the predictors in your data frame in the model, use a `.` , writing write `y ~ .`. You can also *omit* any variable using `-x`. Thus, `y ~ . - x` means "include all the variables except x."

The intercept can be removed from a model by including `-1` in the formula. This is ill-advised in general linear models with continuous predictors, because it forces the line to go through the origin (0, 0), which alters the other coefficients. However, it can be helpful in some kinds of models, such as those with purely categorical predictors.

Table 7.1 summarizes the common options for formula syntax and their interpretation in terms of a linear equation (where β is a model coefficient with β_0 for the intercept, β_1 for the first predictor, and so forth; ε is the error term).

Table 7.1. Syntax for including interactions in model formulas

R formula syntax	Linear model	Description
`y ~ x`	$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$	`y` is a linear function of `x`
`y ~ x - 1`	$y_i = \beta_1 x_i + \beta_2 z_i + \varepsilon_i$	Omit the intercept
`y ~ x + z`	$y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \varepsilon_i$	`y` is a linear combination of `x` and `z`
`y ~ x:z`	$y_i = \beta_0 + \beta_1 x_i z_i + \varepsilon_i$	Include the interaction between `x` and `z`
`y ~ x*z`	$y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \beta_3 x_i z_i \varepsilon_i$	Include `x`, `z` and the interaction between them
`y ~ (u + v + w)^3`	$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \beta_4 u_i v_i + \beta_5 u_i w_i + \beta_6 v_i w_i + \beta_7 u_i v_i w_i + \varepsilon_i$	Include `u`, `v`, and `w`, and all interactions among them up to three-way (`u:v:w`)
`y ~ (u+v+w)^3 - u:v`	$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + +\beta_3 w_i + \beta_5 u_i w_i + \beta_6 v_i w_i + \beta_7 u_i v_i w_i + \varepsilon_i$	Include these variables and all interactions up to three-way, but remove the `u:v` interaction

7.6 Caution! Overfitting

Now that we've seen the complete process of creating a model, from initial data inspection to the potential implications, we have a caution about linear models. As you become more comfortable with linear models, you may want to put more and more predictors into your equation. Be careful about that.

A typical satisfaction drivers survey might include dozens of different features. As you add predictors to a model, estimates of the coefficients become less precise due to both the number of effects and associations among the variables. This shows up in the `lm()` output as larger standard errors of the coefficients, indicating lower confidence in the estimates. This is one reason we like to plot confidence intervals for coefficients, as in Fig. 7.8.

Despite the potentially low confidence in estimates, as you add variables to a model, the value of R^2 will become higher and higher. On a first impression, that might seem as if the model is getting better and better. However, if the estimates of the coefficients are imprecise, then the utility of the model will be poor; it could lead to making the wrong inferences about relationships in your data.

This process of adding too many variables and ending up with a less precise or inappropriate model is called *overfitting*. One way to avoid it is to keep a close eye on the standard errors for the coefficients; small standard errors are an indicator that there is sufficient data to estimate the model. Another approach is to select a subset of the data to *hold out* and not use to estimate the model. After fitting the model, use `predict()` on the hold out data and see how well it performs. Overfitted models will perform poorly when predicting outcomes for holdout data. Stepwise model selection is a traditional approach to select variables while attempting to avoid overfitting; the `step()` function we saw in Sect. 6.5.3 works for `lm` objects the same as for `aov` models.

We recommend to keep models as parsimonious as possible. Although it is tempting to create large, impressive, omnibus models, it is usually more valuable in marketing practice to identify a few interventions with clear and confident interpretations.

7.7 Recommended Procedure for Linear Model Fitting

We followed a lengthy process to arrive at the final model `m7`, and it is helpful to recount the general steps we recommend in creating such a linear model.

1. Inspect the data to make sure it is clean and has the structure you expect, following the outline in Sect. 3.3.3.
2. Check the distributions of the variables to make sure they are not highly skewed (Sect. 7.2.1). If one is skewed, consider transforming it (Sect. 4.5.4).
3. Examine the bivariate scatterplots and correlation matrix (Sect. 7.2.1) to see whether there are any extremely correlated variables (such as $r > 0.9$, or several with $r > 0.8$). If so, omit some variables or consider transforming them if needed; see Sect. 9.1 for further discussion.
4. If you wish to estimate coefficients on a consistent scale, standardize the data with `scale()` (Sect. 7.3.3).
5. After fitting a model, check the residual quantiles in the output. The residuals show how well the model accounts for the individual observations (Sect. 7.2.4).
6. Check the standard model plots using `plot()`, which will help you judge whether a linear model is appropriate or whether there is nonlinearity, and will identify potential outliers in the data (Sect. 7.2.4).
7. Try several models and compare them for overall interpretability and model fit by inspecting the residuals' spread and overall R^2 (Sect. 7.3.1). If the models are nested, you could also use `anova()` for comparison (Sect. 6.5.1) .
8. Report the confidence intervals of the estimates with your interpretation and recommendations (Sect. 7.3).

7.8 Bayesian Linear Models with `MCMCregress()`*

In this section, we review how the satisfaction analysis could be performed with Bayesian methods. This is an optional section; if you're not familiar with Bayesian methods, you could skip this section or review the basics in Sect. 6.6.

Like lm() above, Bayesian inference for a linear model attempts to estimate the most likely coefficients relating the outcome to the explanatory variables. However, the Bayesian method does this by sampling the posterior distribution of estimated model parameters (Sect. 6.6.2), using a procedure known as Markov-chain Monte Carlo (MCMC).

The package MCMCpack includes MCMCregress(), which estimates Bayesian linear models using samples from the posterior distribution; it makes a Bayesian estimation of the model as easy as calling lm(). We call MCMCregress() to estimate the model m7 from above, supplying an identical formula and data frame as we used earlier with lm() (Sect. 7.5):

```
> library(MCMCpack)
...
> m7.bayes <- MCMCregress(overall ~ rides + games + wait + clean + logdist +
+                                    has.child + wait:has.child, data=sat.std)
> summary(m7.bayes)
Iterations = 1001:11000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                       Mean      SD  Naive SE Time-series SE
(Intercept)        -0.69331 0.03702 0.0003702      0.0003702
rides               0.21262 0.03351 0.0003351      0.0003301
games               0.04885 0.02400 0.0002400      0.0002400
wait                0.15096 0.03683 0.0003683      0.0003683
clean               0.30205 0.03515 0.0003515      0.0003515
logdist             0.02891 0.02029 0.0002029      0.0002029
has.childTRUE       0.99837 0.04441 0.0004441      0.0004441
wait:has.childTRUE  0.34733 0.04358 0.0004358      0.0004358
sigma2              0.20374 0.01306 0.0001306      0.0001306

2. Quantiles for each variable:

                        2.5%      25%      50%      75%    97.5%
(Intercept)        -0.764177 -0.71841 -0.69345 -0.66861 -0.62004
rides               0.145773  0.19015  0.21290  0.23499  0.27833
games               0.001507  0.03285  0.04876  0.06453  0.09668
wait                0.079481  0.12629  0.15060  0.17602  0.22353
clean               0.233243  0.27832  0.30218  0.32581  0.37076
logdist            -0.010923  0.01539  0.02885  0.04262  0.06869
has.childTRUE       0.910071  0.96896  0.99857  1.02800  1.08498
wait:has.childTRUE  0.261291  0.31780  0.34720  0.37724  0.43211
sigma2              0.179781  0.19454  0.20311  0.21213  0.23094
```

What does this tell us? The important thing to understand is that MCMCregress() has drawn 10,000 samples from the estimated distribution of possible coefficients for model m7. It then describes those 10,000 sets of estimates in two ways: using central tendency statistics (mean and standard deviation, in the output section labeled "1."), and again using distribution quantiles (in output section "2.").

We can compare the values to those from `lm()` in Sect. 7.5 above. There, we saw that `rides` had an estimated coefficient of 0.2126; here, the mean of the Bayesian estimates is 0.2126 and the median is 0.2129. Similarly, `lm()` estimated `wait:has.child` as 0.9983; the mean Bayesian estimate is 0.9984 and the median is 0.9986. The coefficients estimated by the classical and Bayesian models are nearly identical.

Despite the similar model coefficients, there are two notable differences between this output and the output from `lm()`. First, it includes `2. Quantiles ...` because the Bayesian posterior distribution may be asymmetric; the distribution of estimates could be skewed if that provided a better fit to the data.

Second, the Bayesian output does not include statistical tests or *p*-values; null hypothesis tests are not emphasized in the Bayesian paradigm. Instead, to determine whether a parameter is likely to be non-zero (or to compare it to any other value), check the 2.5 and 97.5 %'iles and directly interpret the credible interval. For instance, in the quantiles above, the 2.5–97.5 %'iles for `logdist` range $(-0.01092, 0.06869)$ and we conclude that the coefficient for `logdist` is not credibly different from 0 at a level of 95 % confidence. However, all of the other coefficients are different from zero.

Note that `MCMCregress()` is similar to `lmBF()` in the `BayesFactor` package that we used in Sect. 6.6. Both functions produce draws from the posterior of a linear model, which you can then summarize using the `summary()`. We used `MCMCregress()` here because `lmBF()` does not estimate interaction coefficients (at the time of writing). It is common in R that different packages do similar things, yet may be better or worse for a specific problem.

If the Bayesian estimates are so similar to those from `lm()`, what is the advantage? The results here are similar for two reasons. First, we have plenty of data and a well-behaved model. Second, classical methods such as `lm()` are eminently suited to estimation of linear models. In Chap. 9 we examine hierarchical Bayesian models, in which more advantages of the Bayesian approach emerge; we later continue that investigation with choice models in Chap. 13.

We also believe, as noted in Sect. 6.6.1, that inferences such as hypothesis testing are clearer and more interpretable in the Bayesian approach. In fitting models, it is not always the case that classical and Bayesian estimates are so similar, and when they differ, we are more inclined to trust the Bayesian estimates.

7.9 Learning More*

In this chapter we've given an overview of linear modeling in R and its application to satisfaction drivers analysis. The same modeling approach could be applied to many other marketing applications, such as advertising response (or *marketing mix*) modeling [18], customer retention (or *churn*) modeling, and pricing analysis.

We covered traditional normal linear models in this chapter, which relate continuous or near-continuous outcomes to predictors. Other models apply in cases where the variables are different in structure, such as binary outcomes or counts. However, the process of estimating those is similar to the steps here. Such models include poisson and binomial regression model for outcomes that are counts, hazard regression for event occurrence (also known as timing regression or survival modeling), and logistic regression for binary outcomes (see Sect. 9.2). R covers all of these models with the *generalized linear model* (GLM) framework, an elegant way of representing many families of models, and such models can be estimated with the `glm()` function. To learn more about generalized models, consult an introduction to GLM such as Dobson [34].

In our synthetic satisfaction drivers data, hypothetical customers rated satisfaction on a 100-point scale, making it reasonable for us to analyze the data as if the ratings were continuous. However, many survey studies collect ratings on a 5- or 7-point scale, which may be questionable to fit with a linear model. Although many analysts use `lm()` for outcomes on 5- or 7-point scales, an alternative is a *cut-point model*, such as an ordered logit or probit model. Such a model will fit the data better and won't make nonsensical predictions like a rating of 6.32 on a 5-point scale (as `lm()` might). These models can be fit with the `polr()` function from the `MASS` package [157].

A more sophisticated model for ordinal ratings data is a Bayesian scale-usage heterogeneity model, as described by Rossi, Allenby, and McCullough [137]. This models that different customers (and cultures) may use scales in different ways; some customers may give systematically higher or lower scores than others due to differences in interpreting the rating scale. When this is modeled, it is possible to find a better estimate of the underlying satisfaction levels. A Bayesian estimation procedure for such models is implemented in the `bayesm` package [136].

In this chapter, we used models in which an effect has uniform influence. For example, we assumed that the effect of satisfaction with cleanliness is a single influence that is the same for every respondent (or, more precisely, whose *average* influence is the same, apart from random individual variation). You might instead consider a model in which the effect varies for different people, with both a group-level and an individual-level effect, known as a *hierarchical* model. We examine ways to estimate individual-level effects using hierarchical models in Chap. 9.

Finally, many data sets have variables that are highly correlated (known as *collinearity*), and this can affect the stability and trustworthiness of linear modeling. In Sect. 9.1 we introduce additional ways to check for collinearity and strategies to mitigate it. One approach is to reduce the number of dimensions under consideration by extracting underlying patterns from the correlated variables; we review such *principal component* and *factor analytic* procedures in Chap. 8.

7.10 Key Points

There are many applications for linear models in marketing: satisfaction drivers analysis, advertising response modeling, customer churn modeling, and so forth. Although these use different kinds of data, they are all implemented in similar ways in R. The following points are some of the important considerations for such analyses. We also summarized the basic process of linear modeling in Sect. 7.7.

- Linear models relate continuous scale *outcome* variables to *predictors* by finding a straight line that best fits the points. A basic linear model function in R is `lm(formula, data)`. `lm()` produces an object that can be used with `plot()`, `summary()`, `predict()`, and other functions to inspect the model fit and estimates of the coefficients.
- Before modeling, it is important to check the data quality and the distribution of values on each variable. For distributions, approximately normal distributions are generally preferred, and data such as counts and revenue often need to be transformed. Also check that variables do not have excessive correlation (Sect. 7.2.1).
- To interpret coefficients on a standardized scale, such that they are comparable to one another, you will either need predictors that are on identical scales or that have been standardized to be on a uniform scale. The most common standardization is conversion to units of standard deviation, performed by `scale()` (Sect. 7.3.3).
- A linear model assumes that the relationship between predictors and an outcome is linear and that errors in fit are symmetric with similar variability across their range (a property known as *homoskedasticity*). Results may be misleading when these assumptions do not match the data. `plot()` of a model can help you assess whether these assumptions are reasonable for your data (Sect. 7.2.5).
- The `summary()` function for `lm` objects provides output that analysts review most frequently, reporting model coefficients along with their standard errors and *p*-values for hypothesis tests assessing whether the coefficients differ from zero (Sect. 7.2.4).
- Factor variables may be included in a model simply by adding the name of the factor to the model formula. R automatically converts the factor into dummy-coded 0/1 values for each level. You must check the direction shown in the output to ensure you interpret these correctly (Sect. 7.4).
- An interaction is a predictor that is the product of two other predictors, and thus assesses the degree to which the predictors reinforce (or cancel) one another. You can model an interaction between `x` and `y` by including `x:y` in a model formula (Sect. 7.5).

- Model building is the process of adding and removing predictors from a model to find a set of predictors that fits the data well. We can compare the fit of different models using the R-squared value or, if models are nested (see Sect. 6.5) by using the more formal ANOVA test (`anova()`) (Sect. 7.3.1).

- You can fit a Bayesian version of a linear model using `MCMCregress()` from the `MCMCpack` package. The usage is nearly identical to `lm()`. The resulting coefficient estimates are assessed as expressing the most likely values (known as credible intervals) under the assumption that the model is appropriate (Sect. 7.8).

- We recommend to interpret coefficients in terms of their estimated ranges, such as confidence intervals in the case of `lm()` (Sect. 7.2.4) or credible intervals from Bayesian estimates (Sect. 7.8). A plot of the coefficient ranges for `lm` objects can be created with the `coefplot` package (Sect. 7.3).