9 Additional Linear Modeling Topics

As we noted in Chap. 7, the range of applications and methods in linear modeling and regression is vast. In this chapter, we discuss four additional topics in linear modeling that often arise in marketing:

- Handling highly correlated observations, which pose a problem known as *collinearity*, as mentioned in Sect. 7.2.1. In Sect. 9.1 we examine the problem in detail, along with ways to detect and remediate collinearity in a data set.
- Fitting models for yes/no, or *binary* outcomes, such as purchasing a product. In Sect. 9.2 we introduce *logistic regression* models to model binary outcomes and their correlates.
- Finding a model for the preferences and responses of *individuals*, not only for the sample as a whole. In marketing, we often wish to understand individual consumers and the diversity of behavior and product interest among people. In Sect. 9.3 we consider *hierarchical linear models* (HLM) for consumer preference in ratings-based conjoint analysis data.
- In marketing, hierarchical models of individual preference are most often estimated using Bayesian methods. In Sect. 9.4 we continue the discussion of HLM by introducing *hierarchical Bayesian* (HB) methods, and we apply HB for ratings-based conjoint analysis.

Except for the two HLM sections, these topics are not especially closely related to one another; unlike other chapters in this book, they may be read independently within this chapter. Still, each section builds on models presented earlier in the book and will extend your knowledge of issues and applications for linear modeling. More importantly, each is a foundational part of a compete toolbox for marketing analysis.

C. Chapman, E.M. Feit, *R for Marketing Research and Analytics*, Use R!,
DOI 10.1007/978-3-319-14436-8_9

9.1 Handling Highly Correlated Variables

We have mentioned several times (as in Sect. 7.2.1) that highly correlated explanatory variables cause problems with linear models. In this section, we examine why that is the case and present strategies to address the problem.

We consider a question that might arise with the retail sales data in Chap. 4, which simulated 12-month online and in-store transactions by customer (see Sect. 4.1). The question is this: which variables are most predictive of online spending? If we wished to increase online spending by customers, which factors might we consider?

9.1.1 An Initial Linear Model of Online Spend

Either create the simulated retail sales data (Sect. 4.1) or load it from the book's website:

```
> cust.df <- read.csv("http://goo.gl/PmPkaG")
> summary(cust.df)
    cust.id            age          credit.score    email      distance.to.store
 Min.   :   1.0   Min.   :19.34   Min.   :543.0   no :186    Min.   :  0.2136
 1st Qu.: 250.8   1st Qu.:31.43   1st Qu.:691.7   yes:814    1st Qu.:  3.3383
...
```

Now we use `lm()` to model spend as a function of all other variables (`online.spend ~ .`). We omit customers with zero online spend; having exactly zero spend is probably related to different factors than positive spend, and we are interested here in the associations for those who spend anything. We also index `[ , -1]` to omit the customer ID column:

```
> spend.m1 <- lm(online.spend ~ .,
+                data=subset(cust.df[ , -1], online.spend > 0))
> summary(spend.m1)
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        6.718948  33.537665   0.200   0.8413
...
online.visits     -0.072269   0.204061  -0.354   0.7234
online.trans      20.610744   0.667450  30.880   <2e-16 ***
store.trans        0.135018   3.211943   0.042   0.9665
store.spend        0.001796   0.078732   0.023   0.9818
sat.service        5.638769   3.016181   1.870   0.0623 .
...
Multiple R-squared:  0.9831,  Adjusted R-squared:  0.9827
```

We have omitted much of the summary to show a few key points. First, online spend is closely related to the number of online transactions (coefficient = 20.6) but not to the number of online visits. That is puzzling. Second, the model accounts for almost all the available variance, $R^2 = 0.98$. These results should cause concern. Because online transactions are dependent on visits, shouldn't those two variables show a similar pattern? How could we be so lucky as to fit a model that nearly perfectly predicts online spending (insofar as it is assessed by R^2)? And notice that the standard error on `store.trans` is quite large, showing that its estimate is very uncertain.

If we turn to data visualization using `gpairs()` (Sect. 7.2.1), we see some problems:

```
> library(gpairs)
> gpairs(cust.df)
```

The result in Fig. 9.1 shows variables with extreme skew and pairs of variables that are very highly correlated.

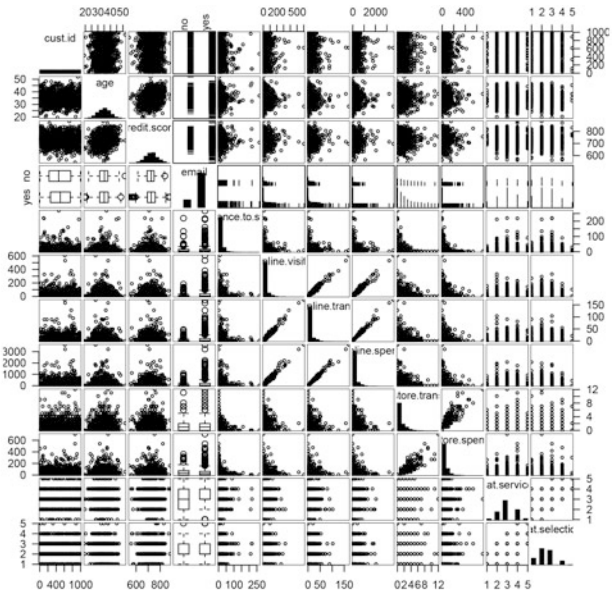

Fig. 9.1. Visualization of the customer data using `gpairs()`. Several variables have extreme skew and other pairs are nearly perfectly correlated; both situations pose problems for linear modeling.

Our first step to remediate the situation is to transform the data using a Box–Cox transformation. Building on the transformation routines we saw in Sect. 4.5.5, we write a short function that uses `BoxCox.lambda()` from the `forecast` package to select the transformation *lambda* automatically [82]. At the same time, we standardize the data with `scale()` (Sect. 7.3.3):

```
> autoTransform <- function(x) {
+   library(forecast)
+   return(scale(BoxCox(x, BoxCox.lambda(x))))
+ }
```

We select the complete cases from our data frame, dropping the customer ID column (`[, -1]`) because it is not a predictor. Then we take only the rows with positive online spend. We create a vector to index all the columns except `email` (which is not numeric), and then `lapply()` the `autoTransform()` function to each numeric column:

```
> cust.df.bc <- cust.df[complete.cases(cust.df), -1]
> cust.df.bc <- subset(cust.df.bc, online.spend > 0)
> numcols <- which(colnames(cust.df.bc) != "email")
> cust.df.bc[ , numcols] <- lapply(cust.df.bc[ , numcols], autoTransform )
```

The result is a data frame with standardized, more normally distributed values, which we can check with `summary()` and `gpairs()`:

```
> summary(cust.df.bc)    # output not shown
> gpairs(cust.df.bc)     # output not shown
```

We refit the model using the transformed data:

```
> spend.m2 <- lm(online.spend ~ ., data=cust.df.bc)
> summary(spend.m2)
...
online.visits     -0.0003913  0.0126165  -0.031    0.975
online.trans       0.9960378  0.0126687  78.622   <2e-16 ***
..
Multiple R-squared:  0.9925,  Adjusted R-squared:  0.9923
```

The coefficients are smaller now because the data have been standardized. Transforming and standardizing the data, although a good idea, have not changed the unbelievable estimate that online spend is highly related to transactions yet unrelated to visits. Indeed, the full model is no better than one that simply predicts spending from the number of transactions alone (see Sect. 6.5.1 on using `anova()` to compare models):

```
> spend.m3 <- lm(online.spend ~ online.trans, data=cust.df.bc)
> anova(spend.m3, spend.m2)
...
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1    416 3.1539
2    407 3.1139  9  0.040001 0.5809 0.8129
```

The small difference between the model fits is reflected in the high p-value ($p = 0.8129$), and thus the null hypothesis of no difference between the models cannot be rejected.

The problem here is *collinearity*: because visits and transactions are so highly related, and also because a linear model assumes that effects are additive, an effect attributed to one variable (such as transactions) is not available in the model to be attributed jointly to another that is highly correlated (visits). This will cause the standard errors of the predictors to increase, which means that the coefficient estimates will be highly uncertain or *unstable*. As a practical consequence, this may cause coefficient estimates to differ dramatically from sample to sample due to minor variations in the data even when underlying relationships are the same.

9.1.2 Remediating Collinearity

The degree of collinearity in data can be assessed as the *variance inflation factor* (VIF). This estimates how much the standard error (variance) of a coefficient in a linear model is increased because of shared variance with other variables, compared to the situation if the variables were uncorrelated or simple single predictor regression were performed.

We assess VIF in the `spend.m2` model using `vif()` from the `car` package:

```
> library(car)
> vif(spend.m2)
            age      credit.score             email distance.to.store
       1.094949          1.112784          1.046874          1.297978
  online.visits      online.trans       store.trans       store.spend
       8.675817          8.747756        125.931383        123.435407
 ...
```

A common rule of thumb is that $VIF > 5.0$ indicates the need to mitigate collinearity. In `spend.m2`, the VIF suggests that collinearity should be addressed for the `online...` and `store...` variables.

There are three general strategies for mitigating collinearity:

- Omit variables that are highly correlated.
- Eliminate correlation by extracting principal components or factors for sets of highly correlated predictors (see Chap. 8).
- Use a method that is robust to collinearity, i.e., something other than traditional linear modeling. There are too many options to consider this possibility exhaustively, but one method to consider would be a random forest approach, which only uses a subset of variables at a time (see Sect. 11.4.2).

Another option for the present data would be to construct a new measure of interest that combines the collinear variables (such as spend per transaction). For purposes here, we explore the first two options above and create models `spend.m4` and `spend.m5`.

We omit highly correlated variables for model `spend.m4` by excluding `online.trans` and `store.trans`, using `-` in the formula:

```
> spend.m4 <- lm(online.spend ~ . -online.trans -store.trans,
+                data=cust.df.bc)
> vif(spend.m4)
...
```

```
    online.visits       store.spend       sat.service     sat.selection
         1.026148          1.215208          1.507866          1.509001
> summary(spend.m4)
...
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       -0.0923395  0.0435047  -2.123   0.0344 *
age               -0.0333779  0.0178813  -1.867   0.0627 .
credit.score      -0.0084524  0.0180637  -0.468   0.6401
emailyes           0.1099655  0.0476011   2.310   0.0214 *
distance.to.store  0.0001702  0.0189271   0.009   0.9928
online.visits      0.9295374  0.0174184  53.365   <2e-16 ***
store.spend        0.0092463  0.0189552   0.488   0.6260
...
Multiple R-squared:  0.8791,  Adjusted R-squared:  0.8767
```

The VIF is now acceptable and we see that online visits are now the best predictor of online spend, although email status and age are also slightly related.

Another approach is to use the principal components of the correlated data. As you will recall from Chap. 8, principal components are uncorrelated (orthogonal). Thus, PCA provides a way to extract composite variables that are guaranteed to be free of collinearity with other variables that are included in the same PCA.

We use PCA to extract the first component for the online variables, and then do this again for the store variables, and add those two initial components to the data frame:

```
> pc.online <- prcomp(cust.df.bc[ , c("online.visits", "online.trans")])
> cust.df.bc$online <- pc.online$x[ , 1]
> pc.store <- prcomp(cust.df.bc[ , c("store.trans", "store.spend")])
> cust.df.bc$store <- pc.store$x[ , 1]
```

Then we fit a new model:

```
> spend.m5 <- lm(online.spend ~ email + age + credit.score +
+                               distance.to.store + sat.service +
+                               sat.selection + online + store,
+                               data=cust.df.bc)
> summary(spend.m5)
...
                    Estimate Std. Error  t value Pr(>|t|)
...
(Intercept)       -3.928e-02  2.410e-02   -1.630   0.1039
emailyes           4.678e-02  2.638e-02    1.773   0.0769 .
age               -1.695e-02  9.882e-03   -1.715   0.0871 .
...
online            -7.019e-01  6.933e-03 -101.247   <2e-16 ***
...
Multiple R-squared:  0.9631,  Adjusted R-squared:  0.9623

> vif(spend.m5)
            email               age      credit.score distance.to.store
         1.039458          1.081430          1.103206          1.224019
      sat.service     sat.selection            online             store
         1.508487          1.509001          1.032362          1.228073
```

VIF poses no problem in this model, and we see that online spend is still associated primarily with online activity (as captured in the first component of the PCA model, `online`) and perhaps slightly with email status and age. One caution when interpreting results that use principal components as explanatory variables is that the components have arbitrary numerical direction; the negative coefficient for `online` here does not imply that online activity results in lower sales.

Although this result—that online sales relate primarily to online activity—may at first appear to be uninteresting, it is better to have an obvious result than an incorrect result. This result might prompt us to collect other data, such as attitudes about our website or online shopping, to build a more complete understanding of factors associated with online spending.

9.2 Linear Models for Binary Outcomes: Logistic Regression

Marketers often observe yes/no outcomes: did a customer purchase a product? Did she take a test drive? Did she sign up for a credit card, or renew her subscription, or respond to a promotion? All of these kinds of outcomes are *binary* because they have only two possible observed states: *yes* or *no*.

At first it is tempting to fit such a model with a typical linear regression model as we saw in Chap. 7, predicting the outcome (1 = yes, 0 = no) as a linear combination of the features. That is not incorrect to do, but a more flexible and useful way to fit such outcomes is with a *logistic* model (also called a *logit* model for reasons we'll discuss below).

9.2.1 Basics of the Logistic Regression Model

The core feature of a logistic model is this: it relates the *probability* of an outcome to an *exponential function* of a predictor variable. We'll illustrate that and show the formula in a moment, but before examining that, let's consider why those are desirable properties and are improvements on a basic linear model.

By modeling the *probability* of an outcome, a logistic model accomplishes two things. First, it more directly models what we're interested in, which is a probability or proportion, such as the likelihood of a given customer to purchase a product, or the expected proportion of a segment who will respond to a promotion. Second, it limits the model to the appropriate range for a proportion, which is $[0,1]$. A basic linear model as generated with `lm()` does not have such a limit and could estimate a nonsensical probability such as 1.05 or -0.04.

We ask indulgence to consider the formula here because it is instrumental in understanding how the model works. The equation for the logistic function is:

$$logistic: p(y) = \frac{e^{v_x}}{e^{v_x}+1} \tag{9.1}$$

In this equation, the outcome of interest is y, and we compute its likelihood $p(y)$ as a function of v_x. We typically estimate v_x as a function of the features (x) of a product, such as price. v_x can take any real value, so we are able to treat it as a continuous function in a linear model. In that case, v_x is composed from one or more coefficients of the model and indicates the importance of the corresponding features of the product.

This formula gives a value between $[0,1]$. The likelihood of y is less than 50 % when v_x is negative, is 50 % when $v_x = 0$, and is above 50 % when v_x is positive. We compute this first by hand, and then switch to the equivalent, built-in `plogis()` function:

```
> exp(0) / (exp(0) + 1)  # computing logistic by hand; could use plogis()
[1] 0.5
> plogis(-Inf)           # infinitely low = likelihood 0
[1] 0
> plogis(2)              # moderate probability = 88% chance of outcome
[1] 0.8807971
> plogis(-0.2)           # weak likelihood
[1] 0.450166
```

Such a model is known as a *logit* model, which determines the value of v_x from the logarithm of the relative probability of occurrence of y:

$$logit: v_x = \log\left(\frac{p(y)}{1-p(y)}\right) \tag{9.2}$$

Again, R includes a built-in function `qlogis()` for the logit function:

```
> log(0.88 / (1-0.88))    # moderate high likelihood
[1] 1.99243
> qlogis(0.88)            # equivalent to hand computation
[1] 1.99243
```

In practice, the expressions *logit model* and *logistic regression* are used interchangeably.

9.2.2 Data for Logistic Regression of Season Passes

We considered an amusement park example in Chap. 7. Suppose that we now have data on the sales of season tickets to the park. The data consist of a table of season ticket *pass sales* (with values of *yes* or *no*), on the basis of two factors: the *channel* used to extend the offer (email, postal mail, or in-person at the park) and whether it was *promoted* in a bundle offering the season ticket with another feature such as free parking, or not. The marketing question is this: are customers more likely to purchase the season pass when it is offered in the bundle (with free parking), or not?

In this section, we see how to simulate such data, and how to create a full data frame from tabulated data. If you wish to load the data from the website instead of working through the data creation, you can retrieve it with:

```
> pass.df <- read.csv("http://goo.gl/J8MH6A")
> pass.df$Promo <- factor(pass.df$Promo, levels=c("NoBundle", "Bundle"))
> summary(pass.df)
  Channel          Promo            Pass
 Email: 633   NoBundle:1482   NoPass :1567
 Mail :1328   Bundle  :1674   YesPass:1589
 Park :1195
```

Note that the second command above is required for reasons we describe in Sect. 9.2.5. Be sure to run it after loading the CSV and check that the `summary()` matches the above.

We encourage you to read the rest of this simulation section and the R language lessons it contains. But if you loaded the data and prefer to skip ahead to analysis, you could continue with Sect. 9.2.6.

9.2.3 Sales Table Data

Suppose that we have been given sales data as shown in Table 9.1.

Table 9.1. Counts of sales of season tickets broken out by promotion status (bundled or not bundled with a promotion), and channel by which a customer was reached (mail, at the park, by email)

	Bought season pass (count)			Did not buy season pass (count)	
	Bundle	NoBundle		Bundle	NoBundle
Mail	242	359	Mail	449	278
Park	639	284	Park	223	49
Email	38	27	Email	83	485

There are several ways to analyze tabular data as shown in Table 9.1, including chi-square analysis (Sect. 6.2), but a versatile approach when the data set is not too large is to convert it to long form and recreate a data frame of individual observations. This lets us use a full range of approaches such as linear modeling with minimal hassle.

To convert the data into such format, we first recreate the cross-tab data table in R. We begin this by reading the values from Table 9.1 one column and row at a time, putting them into a vector:

```
> pass.tab <- c(242, 639, 38, 359, 284, 27, 449, 223, 83, 278, 49, 485)
```

Next we add *dimensions* to the vector, which reformats it as a $3 \times 2 \times 2$ array, and set it to be an object of `class "table"`:

```
> dim(pass.tab) <- c(3, 2, 2)
> class(pass.tab) <- "table"
```

We add the marginal labels to the table by setting its `dimnames` attribute:

```
> dimnames(pass.tab) <- list(Channel=c("Mail", "Park", "Email"),
+                            Promo=c("Bundle", "NoBundle"),
+                            Pass=c("YesPass", "NoPass") )
```

We describe more about `class`, `table`, and `dimnames` in optional Sect. 9.2.4 below. For now, we inspect the resulting table and confirm that it matches Table 9.1:

```
> pass.tab
, , Pass = YesPass
       Promo
Channel Bundle NoBundle
  Mail     242      359
  Park     639      284
  Email     38       27
...
```

We now have the data in R and are ready to create a full data frame from the table. Before that, we take a brief detour into the R language to understand the commands we just used.

9.2.4 Language Brief: Classes and Attributes of Objects*

In this optional section, we explore how the R language understands data types. If you just want to continue with the logistic regression model, you could skip ahead to Sect. 9.2.5.

Every object in R has an associated `class`, which functions use to determine how to handle the object. For example, a vector of real numbers has a class of `numeric`, while a data frame is a `data.frame`. The class of an object may be inspected directly by `class()`:

```
> class(c(1, pi, exp(1)))
[1] "numeric"
> class(data.frame(1:10))
[1] "data.frame"
```

When we examine `str()`, the first thing listed is the class of the object and its raw values:

```
> str(pass.tab)
 table [1:3, 1:2, 1:2] 242 639 38 359 284 27 449 223 83 278 ...
 - attr(*, "dimnames")=List of 3
  ..$ Channel: chr [1:3] "Mail" "Park" "Email"
  ..$ Promo  : chr [1:2] "Bundle" "NoBundle"
  ..$ Pass   : chr [1:2] "YesPass" "NoPass"
```

This code shows that `pass.tab` is an object of class `table` that comprises values `242 639 ....`

The `is.*()` set of functions tests whether an object is of some class (abbreviated here with `*`). For example:

```
> is.table(pass.tab)
[1] TRUE
> is.character(pass.tab)
[1] FALSE
```

Class membership is non-exclusive. For example, tables are composed of counts, and counts are numeric:

```
> is.numeric(pass.tab)
[1] TRUE
```

The `as.*()` functions attempt to treat (convert, or *coerce*) objects as other classes:

```
> as.numeric(pass.tab)
 [1] 242 639  38 359 284  27 449 223  83 278  49 485
> as.character(pass.tab)
 [1] "242" "639" "38"  "359" "284" "27"  "449" "223" "83"  "278" "49"  "485"
```

This shows how we could extract the vector of counts from our park table, and how we might reformat them as character strings for printing, chart labeling, and similar purposes.

In addition to `class`, objects can have other *attributes*. An attribute is a property of an object other than its data, and typically tells R something important about the object. Common attributes that we have used throughout the book are `names` for the names of columns, `dim` for the dimensions of a matrix or data frame, and `class` to specify the type of object. Each of these can be queried for an object:

```
> names(pass.tab)
NULL
> dim(pass.tab)
[1] 3 2 2
> class(pass.tab)
[1] "table"
```

In this case, the `names` for `pass.tab` are `NULL` because it is not a data frame or other object for which names are useful. However, we see that it has `dim` and `class` attributes. A table also has names for its rows and columns, which are known as `dimnames`:

```
> dimnames(pass.tab)
$Channel
[1] "Mail"  "Park"  "Email"
$Promo
[1] "Bundle"   "NoBundle"
...
```

Thus, `Channel`, the first dimension of the table, has elements `"Mail"`, `"Park"`, and `"Email"`.

You can see all the attributes of an object with `attributes()`:

```
> attributes(pass.tab)
$dim
[1] 3 2 2
$class
[1] "table"
...
```

Attributes may be changed using the assignment operator (`<-`). We often use this feature to set `names` of data frames, using `names(DATA) <- c("name1", "name2", ...)`. In the code above, we converted `pass.tab` from a simple vector to a table by assigning `class(pass.tab) <- "table"` and setting its `dim` attribute. As you might imagine, this must be done very carefully! Setting an inappropriate class or dimension of an object will render it useless (but you can usually just change it back to make things work again).

We'll see another use for classes in Sect. 12.3.3, where we use objects' classes to determine how to handle multiple data types inside a function. To learn more about the R class and attribute system, review the R language reference [128] and Wickham's *Advanced R* [163].

9.2.5 Finalizing the Data

We have the data in a table `pass.tab`, which is suitable for analysis as is. However, because most data sets come in the form of an extended data frame with one observation per respondent, we expand it from a table to a complete data frame so the analysis will match typical data structures.

We use `expand.dft()` from the `vcdExtra` package [56] to expand the table to a data frame:

```
> library(vcdExtra)   # install if needed
> pass.df <- expand.dft(pass.tab)
> str(pass.df)
'data.frame': 3156 obs. of  3 variables:
 $ Channel: Factor w/ 3 levels "Email","Mail",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ Promo  : Factor w/ 2 levels "Bundle","NoBundle": 1 1 1 1 1 1 1 1 1 1 ...
 $ Pass   : Factor w/ 2 levels "NoPass","YesPass": 2 2 2 2 2 2 2 2 2 2 ...
```

We now have a data frame with 3,156 observations for whether a customer purchases a `Pass`, by `Channel`, with and without promotion (`Promo`).

We can use `table()` on this data to create cross-tabs other than those in Table 9.1. For example, to see purchases of a pass (`Pass`) by promotion bundle (`Promo`):

```
> table(pass.df$Pass, pass.df$Promo)
          Bundle NoBundle
  NoPass     755      812
  YesPass    919      670
```

Statistical modeling is a detail-oriented process, and before building a model from the data, there is one minor detail to attend to: the factors in `pass.df` are alphabetized—which is how R handles factor names by default—but that is counter-intuitive. We might think that `NoBundle` should have a lower implicit value (such as "bundle = 0") than `Bundle` (which might be "bundle = 1"). However, in the table we just saw, `NoBundle` appears in the second column because it has a higher value thanks to alphabetic ordering.

In a regression model, that would mean that a positive effect of `Bundle` would have a *negative* value (think about it). Rather than having to remember such convoluted logic ("we see a negative effect for *no bundle*, which really means a *positive* effect for bundle after we reverse the signs ..."), it is easier just to set the order straight by reassigning that variable with the factor `levels` in the order we want:

```
> pass.df$Promo <- factor(pass.df$Promo, levels=c("NoBundle", "Bundle"))
> table(pass.df$Pass, pass.df$Promo)
        NoBundle Bundle
  NoPass     812    755
  YesPass    670    919
```

With the data ordered sensibly (*Bundle* > *NoBundle*, *YesPass* > *NoPass*), we proceed with modeling.

9.2.6 Fitting a Logistic Regression Model

A logistic regression model in R is fit as a *generalized linear model* (GLM) using a process similar to linear regression that we saw in Chap. 7 with `lm()`, but with the difference that a GLM can handle dependent variables that are not normally distributed. Thus, GLM can be used to model data counts (such as number of purchases) or time intervals (such as time spent on a website) or binary variables (e.g., did/didn't purchase). The common feature of all GLM models is that they relate normally distributed predictors to a non-normal outcome using a function known as a *link*. This means that they are able to fit models for many different distributions using a single, consistent framework.

In the present case, we model a binary outcome, and the appropriate distribution is a *binomial* distribution (see Sect. 6.3). There are multiple functions and packages that can estimate a GLM in R, but the most common is the `glm(...)` function. `glm()` takes an argument `family=` that specifies the distribution for the outcome variable. For a binary outcome, set `family=binomial`. The default link function for a binomial model is the logit function that we saw in Sect. 9.2.1, so we do not have to specify that. (But, as an example, if we wished to use a probit link function instead, we could specify `family=binomial(link="probit")`, and similarly for other link functions.)

Our marketing question was, "does the promotion bundle have an effect on season pass sales?" and we model this initially with a logistic regression of `Pass` on `Promo`, using `glm(..., family=binomial)` and syntax otherwise identical to `lm()`:

```
> pass.m1 <- glm(Pass ~ Promo, data=pass.df, family=binomial)
```

The initial model appears to confirm that the bundle is effective:

```
> summary(pass.m1)
...
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.19222    0.05219  -3.683 0.000231 ***
PromoBundle  0.38879    0.07167   5.425 5.81e-08 ***
...
```

There is a positive coefficient for the bundle condition, and the effect is statistically significant.

What does a coefficient of 0.3888 mean? We can use it to calculate the association of pass sales, as associated with the promotion bundle factor, by examining the ratio of success (using `plogis()`) to non-success ($1 - success$). A manual way to do this is to use `plogis()` directly:

```
> plogis(0.3888) / (1-plogis(0.3888))      # ratio of outcome % to alternative %
[1] 1.475209
```

This shows that the effect of `Bundle` is an estimated *odds ratio* of 1.475, meaning that customers are 1.475 times more likely to purchase the pass when it is offered in the bundle. Another way to think about this is that the bundle increases the purchase likelihood by 47.5 %. An easier and equivalent way to calculate this is to exponentiate the coefficient:

```
> exp(0.3888)                                    # identical
[1] 1.475209
```

We can find the odds ratios from the model by extracting the coefficients with `coef()` and using `exp()`:

```
> exp(coef(pass.m1))
(Intercept) PromoBundle
  0.8251232   1.4751962
```

We can obtain a confidence interval for the odds ratio using `exp(confint (model))`:

```
> exp(confint(pass.m1))
                2.5 %    97.5 %
(Intercept) 0.744749 0.9138654
PromoBundle 1.282055 1.6979776
```

The odds ratio for the promotion bundle is estimated to be 1.28–1.70, a significant positive effect. This demonstrates that the promotion is highly effective, right? Not necessarily, because the effects are estimated *under the assumption that the model is the one we want to interpret*. But is the model `Pass` $\sim$ `Promo` really the one we should interpret?

9.2.7 Reconsidering the Model

If we explore the data further, we notice something interesting. Consider a table of season pass purchases by channel:

```
> table(pass.df$Pass, pass.df$Channel)
          Email Mail Park
  NoPass    568  727  272
  YesPass    65  601  923
```

The channel that was most successful in selling season tickets was at the park, regardless of whether the promotion was offered.

A good way to visualize tables is with *mosaic* plots, which lay out "tiles" whose areas correspond to counts in a table. The `vcd` package [113] provides several ways to create mosaic plots (including the rather obvious `mosaic()` function). We use a so-called *doubledecker* plot here as it makes the relationships particularly clear in the present data:

```
> library(vcd)     # install if needed
> doubledecker(table(pass.df))
```

The result is shown in Fig. 9.2, where we see that the three channels have somewhat different effects. Sales of season passes are very successful at the park, and very unsuccessful by email. This implies that our model `Pass ~ Promo` may be inadequate and needs to account for the effect of `Channel`.

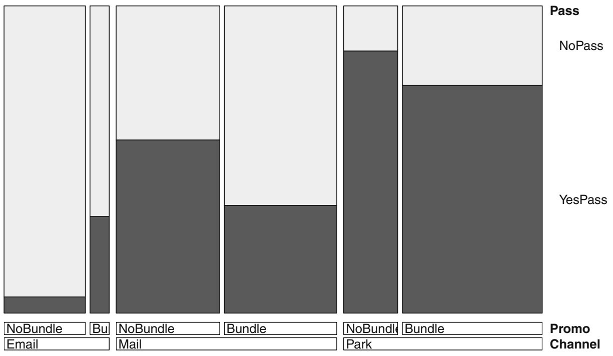

Fig. 9.2. A mosaic plot created with `doubledecker()` [113] for sales of season passes by channel and promotion in simulated amusement park data. Season passes ("YesPass," *plotted as dark areas*) are sold most frequently at the park and least frequently by email. The promotion bundle ("Bundle," the second column within each channel) is associated with higher sales through the email channel, but lower sales in regular mail and at the park, thus showing an interaction effect.

We model a main effect of channel by adding `+ Channel` to the model formula:

```
> pass.m2 <- glm(Pass ~ Promo + Channel, data=pass.df, family=binomial)
> summary(pass.m2)
...
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.07860    0.13167 -15.787  < 2e-16 ***
PromoBundle -0.56022    0.09031  -6.203 5.54e-10 ***
ChannelMail  2.17617    0.14651  14.854  < 2e-16 ***
ChannelPark  3.72176    0.15964  23.313  < 2e-16 ***
...
```

The resulting model now estimates a strong *negative* contribution of the promotion bundle. We compute the odds ratios and their confidence intervals:

```
> exp(coef(pass.m2))
(Intercept) PromoBundle ChannelMail ChannelPark
  0.1251054   0.5710846   8.8125066  41.3371206
> exp(confint(pass.m2))
...
                  2.5 %     97.5 %
PromoBundle  0.47793969  0.6810148
```

```
ChannelMail  6.65770550 11.8328173
ChannelPark 30.42959274 56.9295369
```

In this model, promotion is associated with a 32–53 % lower likelihood of purchasing a season pass. On the other hand, offers in person at the park are associated with season ticket sales 30–56× higher in this model.

But is this the appropriate model? Should we also consider an interaction effect, where `Promo` might have a different effect by `Channel`? Our data exploration suggests a possible interaction effect, especially because of the dramatically different pattern for the influence of `Bundle` in the `Email` channel in Fig. 9.2.

We add an interaction term using the `:` operator, as noted in Sect. 7.5:

```
> pass.m3 <- glm(Pass ~ Promo + Channel + Promo:Channel,
+               data=pass.df, family=binomial)
> summary(pass.m3)
...
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)               -2.8883     0.1977 -14.608  < 2e-16 ***
PromoBundle                2.1071     0.2783   7.571 3.71e-14 ***
ChannelMail                3.1440     0.2133  14.743  < 2e-16 ***
ChannelPark                4.6455     0.2510  18.504  < 2e-16 ***
PromoBundle:ChannelMail   -2.9808     0.3003  -9.925  < 2e-16 ***
PromoBundle:ChannelPark   -2.8115     0.3278  -8.577  < 2e-16 ***
...
```

The interaction of promotion with channel is statistically significant, and is strongly negative for the mail and in-park channels, as opposed to the baseline (omitted) email channel in these simulated data.

In the odds ratios, we see that the promotion is only 2–11 % as effective through the mail and in-park channels as it is in email:

```
> exp(confint(pass.m3))
Waiting for profiling to be done...
                             2.5 %       97.5 %
...
PromoBundle:ChannelMail  0.02795867   0.09102369
PromoBundle:ChannelPark  0.03135437   0.11360965
```

We now have a much better answer to our question. Is the promotion bundle effective? It depends on channel. There is good reason to continue the promotion campaign by email, but its success there does not necessarily imply success at the park or through a regular mail campaign. In case you're wondering how the statistical model is advantageous to simply interpreting Fig. 9.2, one answer is that the model estimates confidence intervals and statistical significance for the effect.

9.2.8 Additional Discussion

Before moving to the topic of hierarchical models, we have a few observations for the current section:

- Although we performed logistic regression here with categorical predictors (factor variables) due to the structure of the amusement park sales data, we could also use continuous predictors in `glm()`. Just add those to the right-hand side of the model formula as we did with `lm()` in Chap. 7.
- We saw that the estimated effect of promotion in these data was positive when we estimated one model, yet negative when we estimated another, and this shows that it is crucial to explore data thoroughly before modeling or interpreting a model. For most marketing data, no model is ever definitive. However, through careful data exploration and consideration of multiple models, we may increase our confidence in our models and the inferences drawn from them.
- The data here are an example of *Simpson's paradox*, which is when the estimate of an aggregate effect is misleading and markedly different than the effect seen in underlying categories. A famous example occurred in graduate admissions at the University of California at Berkeley, where an apparent bias in admissions was due instead to the fact that different departments had different overall admission rates and numbers of applicants [12]. In R, the Berkeley data are available as the table `UCBAdmissions` in the standard `datasets` package.

Logistic regression is powerful method and one that is a particularly good fit for many marketing problems that have binary outcomes. To learn more, see Sect. 9.6. For modeling product choice among sets of alternatives, we cover choice models in Chap. 13.

9.3 Hierarchical Linear Models

In Chap. 7 we saw how to estimate a linear model for data for a sample of respondents. What if we want to estimate the values in the model for *each* respondent? As marketers, it can be very useful to determine individual-level effects such as which customers are more interested in a product or service, who among them want which features, and who are most or less sensitive to price. We can use such information to see the diversity of preference or for purposes such as customer targeting or segmentation (see Chap. 11).

To estimate both a population-level effect and an individual-level effect, we can use a *hierarchical* linear model (HLM). The model is hierarchical because it proposes that individual effects follow a distribution across the population. There are various algorithms to fit such models, but the general approach is that the algorithm fits the overall model to all the data, and then attempts to determine best fit for each individual within that overall estimate (and repeats as necessary).

In general, a data set for HLM at an individual level needs multiple observations per individual. Such observations may come from responses over time (as in transactions or a customer relationship management system (CRM)) or from multiple responses at one time (as in a survey with repeated measures). We consider the case of conjoint analysis, where a respondent rates multiple items on a survey at one time.

How is this different from simply adding the individual, store, or other grouping variable as a factor variable in the model? The key difference is that a factor variable would add a single term that adjusts the model up or down according to the individual. In HLM, however, we can estimate *every* coefficient—or any that we wish—for each individual.

There are other uses for hierarchical models besides customer-level estimation. For example, one might wish to estimate differences by a factor such as geographic region, store, salesperson, product, or promotion campaign. Each of these might provide many responses that could be grouped and allow estimation of a group-level effect within an overall hierarchy. We can't cover every application of HLM here—hierarchical models are the subject of entire books (e.g., Gelman and Hill [60])—yet we hope this discussion will help you to understand when and how they may be useful, and how to begin with them in R.

9.3.1 Some HLM Concepts

A few words of jargon are required. Hierarchical models distinguish two types of effects. One type is *fixed* effects, which are effects that are the same for every respondent. In a standard linear model (Chap. 7) all effects are fixed effects. For instance, in Sect. 9.1.2, we saw that online spend was highly associated with online transactions and slightly associated with age. Both of those estimates are fixed effects that predict the same pattern of association for everyone in the sample.

An HLM also estimates *random* effects, which are additional adjustments to the model coefficients estimated for each individual (or group). These are known as "random" because they are estimated as random variables that follow a distribution around the fixed estimates. However, for the estimate of each individual, they are *best* estimates according to the model, not random guesses in that sense.

Such models are also known as *multilevel* models, where individuals and the full sample are at different levels. They are a subset of models known as *mixed effect* models, where *mixed* reflects the fact that the total effect for each respondent has (at least) two effects that are combined: the overall fixed effect plus the individual-level random effect.

A final variation on mixed effects models is a *nested* model, where a factor of interest might occur only within subgroups of the total sample. For example, if we consider sales in response to different promotions that each occur at different stores, we might model both the effect of store (as a random effect, such that there are different sales intercepts for different stores) and the effect of promotion within store as a nested effect. We do not examine a nested model here, yet they may also be fit using the `lme4` package used below.

9.3.2 Ratings-Based Conjoint Analysis for the Amusement Park

For a hierarchical model, we return to the fictional amusement park from Sect. 7.1. The park is now considering designs for a new roller coaster and hopes to find out which roller coaster features appeal to its customers. They are considering coasters with various possible levels of maximum *speed* (40, 50, 60 or 70 mph), *height* (200, 300, or 400 ft), *construction* type (wood or steel), and *theme* (dragon or eagle). The stakeholders wish to know which combination of features would be most popular according to customers' stated preference.

One way to examine this is a survey that asks customers to rate different roller coasters (illustrated with photographs or videos for more realism). For example:

> On a 10-point scale, where 10 is the best and 1 is the worst, how would you rate a roller coaster that is made of **wood**, is **400 ft** high, has a maximum speed of **50 mph**, with a **dragon theme**?

Customers' ratings could be analyzed with a linear model where the ratings are predicted from the different features of the roller coasters. This would tell us the contribution of each feature to the total rating.

Additionally, we wish to understand these preferences at an individual level, such that we can see the distribution of preference or identify individuals for potential marketing actions. To do this, we use a HLM that estimates both the overall fixed effect and the individual-level random effect.

In the following section we simulate consumers' ratings for such a survey. The code is brief and illustrative of the data, but if you wish to skip the simulation, you can load the data from the book's website:

```
> conjoint.df <- read.csv("http://goo.gl/G8knGV")
> conjoint.df$speed  <- factor(conjoint.df$speed)
> conjoint.df$height <- factor(conjoint.df$height)
> summary(conjoint.df)
    resp.id           rating       speed     height      const
 Min.   :  1.00   Min.   : 1.000   40: 800   200:1400   Steel:1400
 1st Qu.: 50.75   1st Qu.: 3.000   50:1200   300:1200   Wood :1800
...
```

Given this data, you may skip to Sect. 9.3.4.

9.3.3 Simulating Ratings-Based Conjoint Data

In this section we simulate responses for a hypothetical conjoint analysis survey with 200 respondents who each rate the same set of 16 roller coaster profiles. If you have worked through the data simulation in previous chapters, this code should be relatively simple in structure, although a few functions are new.

We set the structure: 200 respondents who rate 16 designs, each with 4 roller coaster attributes:

```
> set.seed(12814)
> resp.id <- 1:200 # respondent ids
> nques <- 16      # number of conjoint ratings per respondent
> speed <- sample(as.factor(c("40", "50", "60", "70")), size=nques,
+                           replace=TRUE)
> height <- sample(as.factor(c("200", "300", "400")), size=nques, replace=TRUE)
> const <- sample(as.factor(c("Wood", "Steel")), size= nques, replace=TRUE)
> theme <- sample(as.factor(c("Dragon", "Eagle")), size=nques, replace=TRUE)
```

In this example we assume that all respondents rate the same set of designs. Depending on your study's goal, you might instead want to have a different, random set for each respondent. A single set of designs is convenient for printed surveys, while an online study could easily have a different set for every respondent; we will see an example in Chap. 13.

Next we create a model matrix for the combinations of features to rate. We draw multivariate random normal values for respondents' preferences using `mvrnorm()` from the `MASS` package [157]:

```
> profiles.df <- data.frame(speed, height, const, theme)
> profiles.model <- model.matrix(~ speed + height + const + theme,
+                                data=profiles.df)
> library(MASS)     # a standard library in R
> weights <- mvrnorm(length(resp.id),
+                    mu=c(-3, 0.5, 1, 3, 2, 1, 0, -0.5),
+                    Sigma=diag(c(0.2, 0.1, 0.1, 0.1, 0.2, 0.3, 1, 1)))
```

`model.matrix()` converts the list of design attributes (`profiles.df`) into coded variables; it is similarly used by functions such as `lm()` to convert factors into variables for regression equations. You can compare `profiles.model` to `profiles.df` to see how this works. We use `mvrnorm()` to draw unique preference `weights` for each respondent. Estimating those later is the key feature that distinguishes a hierarchical model from a standard linear model.

Given the designs to be rated and individuals' preferences, we compile the simulated individual ratings. For each respondent, we multiply the preference weights by the design matrix to get the total preference (utility) for each design, adding some random noise with `rnorm()`. We convert the utility to a 10-point rating scale using `cut()` (see Sect. 12.4.1), and add the respondent's result to the overall data frame:

```
> conjoint.df <- NULL   # make sure there's no data yet
> for (i in seq_along(resp.id)) {
+   # create one respondent's ratings of the 16 items, plus error
+   utility <- profiles.model %*% weights[i, ] + rnorm(16)  # preference
+   rating <- as.numeric(cut(utility, 10))   # put on a 10-point scale
+   conjoint.resp <- cbind(resp.id=rep(i, nques), rating, profiles.df)
+   conjoint.df <- rbind(conjoint.df, conjoint.resp)
+ }
```

Building a data frame using `rbind()` repeatedly instead of preallocating a whole matrix is not efficient, but it is easy to understand and it is fast enough for this data set. For large data sets, it would be better to preallocate the data frame for the size needed and fill in the rows. With a bit of matrix manipulation, one might instead create the whole data frame at once; but a simple, readable method like the one here may be more effective overall if it's easier and more reliable to code.

9.3.4 An Initial Linear Model

We begin as always with a quick `summary` of our conjoint data to check it (create or load the data as described in Sect. 9.3.2 if needed):

```
> summary(conjoint.df)
    resp.id          rating       speed     height      const
 Min.   :  1.00   Min.   : 1.000   40: 800   200:1400   Steel:1400
 1st Qu.: 50.75   1st Qu.: 3.000   50:1200   300:1200   Wood :1800
...
```

Ratings of the designs range from 1 (strongly disprefer) to 10 (strong prefer). We also see the counts of the features that were shown in various combinations: `speed`, `height`, `const`, and `theme`.

Our goal is to determine how the four features relate to the ratings. At an aggregate level, we might use `by()` to find the average rating for levels of each attribute. For example, the averages by `height` are:

```
> by(conjoint.df$rating, conjoint.df$height, mean)
conjoint.df$height: 200
[1] 3.657857
------------------------------------------------------------
conjoint.df$height: 300
[1] 7.254167
------------------------------------------------------------
conjoint.df$height: 400
[1] 5.05
```

The average rating for designs with 300 foot height is 7.25 points on the 10-point scale, compared to 3.66 and 5.05 for heights of 200 and 400 ft. So, respondents prefer the middle of our height range.

We could examine each individual feature in that way, but a more comprehensive linear model considers all of the effects in combination. To start, we'll estimate a regular linear model without a hierarchical component using `lm()` (Chap. 7):

```
> ride.lm <- lm(rating ~ speed + height + const + theme, data=conjoint.df)
> summary(ride.lm)
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.07307    0.08102  37.932  < 2e-16 ***
speed50      0.82077    0.10922   7.515 7.35e-14 ***
speed60      1.57443    0.12774  12.326  < 2e-16 ***
speed70      4.48697    0.15087  29.740  < 2e-16 ***
height300    2.94551    0.09077  32.452  < 2e-16 ***
height400    1.44738    0.12759  11.344  < 2e-16 ***
constWood   -0.11826    0.11191  -1.057    0.291
themeEagle  -0.75454    0.11186  -6.745 1.81e-11 ***
...
```

In this abbreviated output, the coefficients indicate the association with preference (the `rating`). The highest rated roller coaster on average would have a top speed of 70 mph, a height of 300 ft, steel construction, and the dragon theme (steel and dragon because wood and eagle have negative values). We estimate an overall rating for this most-desired coaster; it would be the intercept + speed70 + height300 (steel and dragon are included in the intercept), or $3.07 + 4.49 + 2.94 = 10.46$ points on our 10-point rating scale.

But wait! That's not possible; our scale is capped at 10 points. This shows that simply interpreting the "average" result can be misleading. The coefficients are estimated on the basis of designs that mostly combine both desirable and undesirable attributes, and are not as reliable at the extremes of preference. Additionally, it could happen that few people prefer that exact combination even though the individual features are each best on average.

Consider that the coefficient for `constWood` is near zero. Are people indifferent between wood and steel coasters, or do they have strong preferences that cancel out when averaged? If people are strongly but almost equally divided, that's important for us to know as marketers; it might suggest that we construct different rides that appeal to two different groups. On the other hand, if they are truly indifferent, we could choose between steel and wood on the basis of cost and other factors.

To understand our respondents better, we turn next to a hierarchical model that will estimate both the overall average preference level and individual preferences within the group.

9.3.5 Hierarchical Linear Model with `lme4`

The linear model `ride.lm` has only fixed effects that are estimated at the sample level. In an HLM, we add one or more individual-level effects to those.

The simplest HLM allows individuals to vary only in terms of the constant intercept. For example, we might expect that individuals vary in their usage of a rating scale such that some will rate our roller coaster designs higher or lower than the average respondent. This would be an individual-level random effect for the intercept term.

To estimate an HLM with fixed effects plus a per-respondent intercept, we change the `lm()` model from above in three ways. First, instead of `lm()`, we use a hierarchical estimation function, `lmer()` from the `lme4` package [8].

Second, in the formula for `lmer()`, we specify the term(s) for which to estimate random effects. For the intercept, that is signified as simply "`1`". Third, we specify the grouping variable, for which a random effect will be estimated for each unique group. In our conjoint data, the group in the set of responses for a single respondent, which is identified in the data frame by respondent number, `resp.id`. With `lme4`, we specify the random effect and grouping variable with syntax using a vertical bar ("`|`") as `+ (predictors | group)`, or in this case for the intercept only, `+ (1 | resp.id)`.

We estimate this model using `lme4`, where the only difference from the call to `lm()` above is the addition of a term for random intercept by respondent:

```
> library(lme4)
> ride.hlm1 <- lmer(rating ~ speed + height + const + theme + (1 | resp.id),
+                   data=conjoint.df)
> summary(ride.hlm1)
...
Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.3970 -0.6963  0.0006  0.6700  3.3689

Random effects:
 Groups   Name        Variance Std.Dev.
 resp.id  (Intercept) 0.3352   0.5789
 Residual             3.5358   1.8804
Number of obs: 3200, groups:  resp.id, 200

Fixed effects:
            Estimate Std. Error t value
(Intercept)  3.07307    0.08759   35.08
speed50      0.82077    0.10439    7.86
speed60      1.57443    0.12209   12.90
speed70      4.48697    0.14421   31.11
height300    2.94551    0.08676   33.95
height400    1.44738    0.12195   11.87
constWood   -0.11826    0.10696   -1.11
themeEagle  -0.75454    0.10692   -7.06
...
```

In this output, we see that the fixed effects are identical to those estimated by `lm()` above. But now we have also estimated a unique intercept term adjustment for each respondent. The output section labeled "Random effects" shows 3,200 total observations (survey questions) grouped into 200 respondents for which a random effect was estimated (such as the effect for `(Intercept)`).

`fixef()` is an easy way to extract just the fixed (population level) effects:

```
> fixef(ride.hlm1)
(Intercept)      speed50      speed60      speed70    height300    height400 ...
  3.0730724    0.8207718    1.5744257    4.4869715    2.9455084    1.4473848 ...
```

The 200 per-respondent random effect estimates for intercept, which `summary (ride.hlm1)` does not display because there could be many of them, are accessed with `ranef()` (and we additionally use `head()` to shorten the output):

```
> head(ranef(ride.hlm1)$resp.id)
  (Intercept)
1 -0.65085634
2 -0.04821158
3 -0.31186866
...
```

The complete effect for each respondent comprises the overall fixed effects that apply to everyone, plus the individually varying random effects (in this case, just the intercept). Those are accessed using `coef()`:

```
> head(coef(ride.hlm1)$resp.id)
  (Intercept)   speed50  speed60  speed70 height300 height400  constWood ...
1    2.422216 0.8207718 1.574426 4.486971  2.945508  1.447385 -0.1182553 ...
2    3.024861 0.8207718 1.574426 4.486971  2.945508  1.447385 -0.1182553 ...
3    2.761204 0.8207718 1.574426 4.486971  2.945508  1.447385 -0.1182553 ...
...
```

It is possible to estimate random effects for multiple grouping factors (hierarchical levels), so these effects must be extracted for the grouping level of interest by selecting the coefficient matrix named `$resp.id`.

In `coef(ride.hlm1)$resp.id`, each respondent has the overall sample-level value of the effect on all coefficients except for intercept, and the final intercept coefficient is the same as the fixed effect plus the random effect. For example, for respondent 1, the intercept is $3.07(\texttt{fixef}) - 0.65(\texttt{ranef}) = 2.42(\texttt{coef})$.

9.3.6 The Complete Hierarchical Linear Model

The most common hierarchical model in marketing practice is to estimate a random effect parameter for every coefficient of interest for every respondent. This is easy to do with the `lme4` syntax; simply add all the variables of interest to the predictors in the random effects specification `(predictors | group)`.

For the conjoint data, we write the random effects part of the formula as `(speed + height + const + theme | resp.id)`. Before estimating that model, we should note that this is a much more complex model than the intercept model above. Whereas the random intercept-only HLM estimated 8 fixed parameters and 200 random effects, the full model will estimate 8 fixed effects plus $8 * 200$ random effects. And it will do this for a total data frame of 3,200 observations.

This fact has two implications. First, the estimation can be rather slow, taking several minutes for the present model at the time of writing. Second, there are so many parameters that even 3,200 observations is not a lot, and one can expect some difficulty finding a stable *converged* model.

With those facts in mind, we estimate the full model as follows (this will take some time, perhaps several minutes):

```
> ride.hlm2 <- lmer(rating ~ speed + height + const + theme +
+                            (speed + height + const + theme | resp.id),
+                   data=conjoint.df,
+                   control=lmerControl(optCtrl=list(maxfun=100000)))
```

Compared to model `ride.hlm1` above, this model has two changes. First, we added all four roller coaster factors to be estimated for random effects. Second, we added a `control` argument to `lmer()`, which increases the `maxfun` number of iterations to attempt convergence from 10,000 iterations (the default) to 100,000. This allows the model to converge better, although still not completely as we see in the resulting warnings when it finishes:

```
Warning messages:
1: In optwrap(optimizer, devfun, getStart(start, rho$lower, rho$pp) ...
2: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv ...
```

Despite the warnings, we proceed with data analysis here because it is quite slow to run the model to convergence and the exact results are for illustration, not for an important business decision. For a model of importance, we recommend to run to convergence whenever possible.

If you run into warnings, we suggest five potential remedies. First, increase the `control maxfun` argument by a factor of 2, 5, or 10 to see if convergence results (and repeat that if necessary). Second, check whether the `max|grad|` (maximum absolute value of the gradient in the optimization function; cf. [8]) is small, such as $max < 0.001$; if so, you may be okay. Alternatively, if $max >> .01$, such as $max = 0.10$, increase the iterations. Third, do a web search for the warnings you receive and consider the suggestions offered on R discussion forums. Fourth, consider using a different optimization function (see `lme4` documentation [8]). Fifth, consider collecting more data, or evaluate your data for internal consistency. Again, we skip these steps now primarily for convenience.

Fixed effects are extracted with `fixef()`:

```
> fixef(ride.hlm2)
(Intercept)     speed50     speed60     speed70   height300   height400 ...
  3.0730724   0.8207718   1.5744257   4.4869715   2.9455084   1.4473848 ...
```

This part of the `ride.hlm2` model is identical to the model estimated for `ride.hlm1` above, so the coefficients are identical.

The random effects now include an estimate for each parameter for each respondent. Again, because we grouped by `resp.id` and could have had multiple grouping factors, we request the `$resp.id` portion of the random effects using `ranef()`:

```
> head(ranef(ride.hlm2)$resp.id)
  (Intercept)     speed50     speed60     speed70   height300
1  -1.1199673 -0.20603467 -0.12507535  0.10294883  0.10742700
2  -1.0104334  0.24975368 -0.08225264  0.16262789  0.05610339
3  -1.0352111 -0.21870984  0.31082035 -0.29288693  0.34166296
...
```

Notice that the random intercepts are no longer identical to those estimated in model `ride.hlm1`, because we added seven explanatory variables and the predicted outcome rating points are distributed differently across the predictors.

We obtain the total coefficients per respondent with `coef()`:

```
> head(coef(ride.hlm2)$resp.id)
  (Intercept)   speed50  speed60  speed70 height300 height400  constWood
1    1.953105 0.6147371 1.449350 4.589920  3.052935 1.4473264  0.1060510
2    2.062639 1.0705254 1.492173 4.649599  3.001612 2.5206374  1.4178018
3    2.037861 0.6020619 1.885246 4.194085  3.287171 1.3337777  0.4858052
...
```

As a final sanity check to confirm that the model matches expectations, we choose a respondent (ID 196) and see that the coefficients are indeed the sum of the fixed and random effects:

```
> fixef(ride.hlm2) + ranef(ride.hlm2)$resp.id[196, ]
    (Intercept)   speed50  speed60  speed70 height300 height400 constWood ...
196    2.143063 0.7534565 1.271094 4.594383   2.94959  1.212746  2.580269 ...
> coef(ride.hlm2)$resp.id[196, ]
    (Intercept)   speed50  speed60  speed70 height300 height400 constWood ...
196    2.143063 0.7534565 1.271094 4.594383   2.94959  1.212746  2.580269 ...
```

In this code, the random effect and coefficient values for respondent 196 are retrieved by indexing that row within the corresponding `$resp.id` matrix.

9.3.7 Summary of HLM with `lme4`

This concludes our discussion of classical hierarchical models; in the next section, we consider the Bayesian approach to HLM, which uses the same general conceptual model but a different estimation method.

In this section, we hope to have convinced you that, when you have multiple observations for an individual or other grouping factor of interest, you should consider a hierarchical model that estimates both sample-level and individual- or group-level effects. These models are relatively straightforward to estimate using the `lme4` package.

Besides *customer*-level models, which are most common in marketing, other factors for which one might wish to estimate a hierarchical model include *store*, *country*, *geographic region*, *advertising campaign*, *advertising creative*, *channel*, *bundle*, and *brand*.

If this section has inspired you to consider adding hierarchical modeling to your toolbox, see "Learning More" (Sect. 9.6) for pointers to other resources.

9.4 Bayesian Hierarchical Linear Models*

This is an optional section that you may skip if you are not interested in the Bayesian approach to estimate hierarchical models.

Hierarchical models may be fit with classical estimation procedures (such as the `lme4` package we saw above), yet they are particularly well suited to Bayesian estimation, which gives a best estimate for each individual even when there are few individual observations.

The method we use here is known as a *hierarchical Bayes* approach; *hierarchical* because it models individuals in relationship to an overarching distribution, and *Bayes* because it uses Bayesian estimation techniques to fit the models (see Sects. 6.6.1 and 6.6.2 for an introduction).

In this section, we apply a hierarchical Bayes (HB) method to estimate the HLM for ratings-based (metric) conjoint analysis, using the same data set that we analyzed with classical hierarchical models in Sect. 9.3 above. Before continuing this section you should:

- Review the concepts of Bayesian linear models and MCMC estimation in Sect. 7.8
- Review the concepts of HLM in Sects. 9.3 and 9.3.1
- Review the description of the amusement park conjoint analysis data in Sect. 9.3.2

Download the simulated amusement park conjoint analysis data as follows, or see Sect. 9.3.2:

```
> conjoint.df <- read.csv("http://goo.gl/G8knGV")
> conjoint.df$speed  <- factor(conjoint.df$speed)
> conjoint.df$height <- factor(conjoint.df$height)
> summary(conjoint.df)
    resp.id           rating        speed      height       const
 Min.   :  1.00   Min.   : 1.000   40: 800   200:1400   Steel:1400
 1st Qu.: 50.75   1st Qu.: 3.000   50:1200   300:1200   Wood :1800
...
```

9.4.1 Initial Linear Model with `MCMCregress()`*

We start by estimating a non-hierarchical model, which allows us to check that our basic estimation procedures are working before we attempt a complex model. We model respondents' ratings of roller coaster designs as a function of roller coaster features using `MCMCregress()` to fit a simple linear model as we did in Sect. 7.8:

```
> library(MCMCpack)
> set.seed(97439)
> ride.mc1 <- MCMCregress(rating ~ speed + height + const + theme,
+                         data=conjoint.df)
> summary(ride.mc1)
...
               Mean      SD  Naive SE Time-series SE
(Intercept)  3.0729 0.08112 0.0008112      0.0008112
speed50      0.8208 0.11061 0.0011061      0.0011126
speed60      1.5754 0.12889 0.0012889      0.0012889
speed70      4.4873 0.15002 0.0015002      0.0015002
height300    2.9444 0.09122 0.0009122      0.0009337
height400    1.4461 0.12934 0.0012934      0.0013367
constWood   -0.1187 0.11310 0.0011310      0.0011310
themeEagle  -0.7533 0.11308 0.0011308      0.0011308
sigma2       3.8705 0.09737 0.0009737      0.0009737
...
```

As expected, the overall effects are nearly identical to those estimated by the classical linear models in Sect. 9.3.5, so we are ready to add the hierarchical component to the model.

9.4.2 Hierarchical Linear Model with `MCMChregress()`*

We estimate a hierarchical model using `MCMChregress(fixed, random, group, data, r, R)`. Note the **h** for *hierarchical* buried in that function name. This is a slightly different syntax than `lme4` uses (as we reviewed in Sect. 9.3.5), as it separates the fixed and random effect specifications. The key arguments we use here are:

`fixed` : formula for fixed effects at the higher level that are the same for all respondents

`random` : formula for random effects that are estimated for each respondent

`group` : name of the column with identifiers that group observations for the random effects

`data` : the data frame with observations

`r, R` : pooling arguments. We'll just set them for now; see below for detail

For `fixed` effects we specify the primary model to estimate: `rating ~ speed + height + const + theme`. For `random` effects, the most common models in marketing estimate all parameters of the model for every respondent, so we specify `random = ~ speed + height + const + theme`. Because we are estimating by individual, `group` is the respondent identifier, `"resp.id"`.

Estimation of this model may take several minutes to run. Here is the final code:

```
> set.seed(97439)
> ride.mc2 <- MCMChregress(fixed = rating ~ speed + height + const + theme,
+                          random = ~ speed + height + const + theme,
+                          group="resp.id", data=conjoint.df, r=8, R=diag(8))

Running the Gibbs sampler. It may be long, keep cool :)  ...
```

While the model runs, let's examine the two arguments `r` and `R`. A hierarchical model assumes that each respondent has a set of preferences (coefficients) drawn from a larger distribution that defines the range of possible preferences. The model is slow because it makes *thousands* of estimates of both the individuals' coefficients and the higher-order distributions that best describe those individuals.

Of course there are only a few observations for each respondent, and a model for a single person cannot be estimated very well with such limited data. To improve estimation, the MCMC model *pools* information across respondents, allowing estimates to be more or less similar to one another based on the data. If several respondents dislike a feature, it's more likely (but not certain) that another randomly selected respondent will also dislike it; this expected similarity is used to improve estimates given sparse data.

That degree of pooling across respondents is adjusted by the final two arguments `r` and `R`. For most analyses, you can set `r` equal to the number of parameters in your model and `R` equal to a diagonal matrix with values along the diagonal equal to the number of parameters in your model, and the algorithm will determine the optimal level of pooling from the data. This can be done with the simple function `diag(K)`, where `K` is the same number as `r`. However, if you plan to run hierarchical Bayesian models regularly, you will wish to learn more about pooling; check the references in Sect. 9.6.

By now, `MCMChregress()` from above should have finished, and we can review its result:

```
> str(ride.mc2)
List of 2
 $ mcmc  : mcmc [1:1000, 1:1674] 3.04 2.87 2.9 3.06 2.98 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:1674] "beta.(Intercept)" "beta.speed50" "beta.speed60" "beta.
     speed70" ...
  ..- attr(*, "mcpar")= num [1:3] 1001 10991 10
 $ Y.pred: num [1:3200] 4.94 2.69 5.73 6.24 4.67 ...
```

The output of `MCMChregress` is a list with two items. The first item in this list is an `mcmc` object containing the draws from the posterior distribution of the parameters. A notable thing is that `ride.mc2$mcmc` contains 1,674 columns. Why so many? The model estimates a set of 8 coefficients—the preferences for each attribute of our roller coasters—for every one of the 200 respondents. That's 1,600 parameters plus a few more that describe the overall population distribution. For each of those parameters, it drew 1,000 estimates from the posterior distribution for every respondent (see Sect. 6.6.2).

Let's look at the first 8 columns, estimated coefficients for the overall, population-level preferences:

```
> summary(ride.mc2$mcmc[ ,1:8])
...
                    Mean     SD Naive SE Time-series SE
beta.(Intercept)  3.0739 0.1694 0.005356       0.005457
beta.speed50      0.8168 0.1398 0.004422       0.004422
beta.speed60      1.5691 0.1618 0.005117       0.005569
beta.speed70      4.4849 0.1862 0.005889       0.005889
beta.height300    2.9474 0.1235 0.003904       0.003681
beta.height400    1.4578 0.1796 0.005680       0.005680
beta.constWood   -0.1128 0.1952 0.006172       0.005615
beta.themeEagle  -0.7542 0.1857 0.005871       0.005871
...
```

These estimates are nearly identical to the result of non-hierarchical `MCMCregress` in model `ride.mc1` above. `speed70` is still preferred and worth 4.5 points on our rating scale, preference for wood construction is near zero, and so forth. Where is the difference? Why did we wait several minutes for these results? The answer is in the coefficients it estimated for *individual* respondents.

Let's look at an example respondent; we pull and summarize the posterior draws for the parameters that are associated with respondent 196. We do this by finding columns that are named with "196" (the `resp.id` that we want). We accomplish that by indexing the columns with the results of the `grepl()` function that identifies elements of a character vector (in this case, column names) containing a particular string:

```
> summary(ride.mc2$mcmc[ , grepl(".196", colnames(ride.mc2$mcmc), fixed=TRUE)])
...
                       Mean     SD Naive SE Time-series SE
b.(Intercept).196 -1.03806 0.6780  0.02144        0.02144
b.speed50.196      0.44049 0.5434  0.01718        0.01718
b.speed60.196      0.10442 0.6335  0.02003        0.02003
b.speed70.196      0.03807 0.7167  0.02266        0.02357
b.height300.196   -0.35414 0.5441  0.01721        0.01797
b.height400.196   -0.55132 0.7357  0.02327        0.02327
b.constWood.196    2.57915 0.8370  0.02647        0.02647
b.themeEagle.196  -1.41955 0.8220  0.02599        0.02599
...
```

Respondent 196 strongly prefers wood coasters; her ratings for them are 2.5 points higher on our 10-point scale than those for steel construction (the default level). On the other hand, she dislikes the eagle-themed design, rating it -1.4 points lower on average than the dragon theme. These preferences are rather different than the population averages above.

How could we use this information? The ideal roller coaster for respondent 196, according to her responses, would be a dragon-themed wood coaster with a top speed of 50 mph and a height of 200 ft (the default level not shown). Although individual customization is impractical for roller coasters, a plausible marketing use would be to segment respondents' preferences to determine a mix of coasters (see Chap. 11). For instance, we might ask which new coaster would maximize preference over and above the coasters the park already has; in other words, we could investigate a product line extension. More immediately, if we have respondents' contact information, we could tailor marketing communications to this and similar respondents and tell them about wooden coasters at the park.

The MCMC output also informs our confidence of estimates. One could use the standard error of the mean estimate, but we recommend instead to use the values from the `Quantiles` section of the output. Let's look at the population estimates again, but focus on the quantiles::

```
> summary(ride.mc2$mcmc[ ,1:8])
...
2. Quantiles for each variable:

                     2.5%     25%     50%      75%   97.5%
beta.(Intercept)  2.7389  2.9594  3.0764  3.18818  3.4099
beta.speed50      0.5421  0.7251  0.8114  0.91274  1.0801
beta.speed60      1.2604  1.4636  1.5725  1.68365  1.8804
beta.speed70      4.1213  4.3599  4.4834  4.60792  4.8599
beta.height300    2.7114  2.8642  2.9501  3.03263  3.1779
beta.height400    1.0898  1.3429  1.4589  1.58500  1.8017
beta.constWood   -0.5219 -0.2464 -0.1105  0.01628  0.2698
beta.themeEagle  -1.0999 -0.8745 -0.7571 -0.63284 -0.3609
```

This tells us that the fixed effect estimate for `speed70` had a value between 4.12–4.86 in 95 % of the draws from the posterior distribution. Thus, we can use these values to express the credible interval for the parameters we report. An advantage of Bayesian statistics is that confidence in estimates can be stated directly, without resorting to discussion of null hypotheses.

9.4.3 Inspecting Distribution of Preference*

We wondered above whether respondents were just indifferent to wooden versus steel coasters, or had significant differences. To investigate this in the estimated model, we need to do a bit of work. First, we extract out all the coefficients labeled `b.constWood`, which are the individual-level estimates for preference for wood construction. There are 200 columns for these coefficients, one for each customer in our data set.

Those values each represent a *difference* for the individual relative to the overall population, so we add the values to the baseline population estimate, `beta.constWood`. Because we have 1,000 sets of estimates from the MCMC draws, we compute the total (individual plus population mean) for each of the 1,000 draws from the posterior distribution, and summarize those totals. (Do *not* summarize first and then add.)

Although this process may sound complex, it is accomplished in a single, albeit cryptic, command:

```
> ride.constWood <- summary(ride.mc2$mcmc[ , grepl("b.constWood",
+                                              colnames(ride.mc2$mcmc))]
+                            + ride.mc2$mcmc[ , "beta.constWood"])
```

Deconstructing this code, it finds the columns in `mcmc` draws with "b.constWood" in their names; those are the individual differences in preference. It adds the population value, `beta.constWood`, to obtain the total preference for each respondent. Then it summarizes the result. (You might try parts of this code in the R console to see how this works.)

The result is that `ride.constWood` contains estimates from the posterior distribution for individual-level preference of wood over steel coasters. We plot these to see the distribution of individuals' preferences for wood coasters:

```
> hist(ride.constWood$statistics[,1],
+      main="Preference for Wood vs. Steel",
+      xlab="Rating points", ylab="Count of Respondents", xlim=c(-4,4))
```

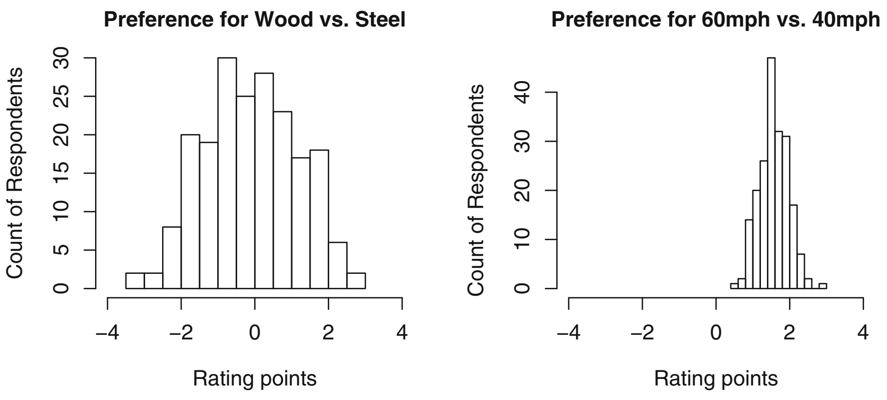

Fig. 9.3. Histograms of individual respondent preferences in a ratings-based conjoint analysis model.

We compare that to the distribution of preference for 60 mph speed (versus baseline 40mph):

```
> ride.speed60 <- summary(ride.mc2$mcmc[,grepl("b.speed60",
+                                           colnames(ride.mc2$mcmc))]
+                         + ride.mc2$mcmc[,"beta.speed60"])
> hist(ride.speed60$statistics[,1],
+      main="Preference for 60 vs. 40\,mph",
+      xlab="Rating points", ylab="Count of Respondents", xlim=c(-4,4))
```

The resulting charts are shown in Fig. 9.3. In the first, we see a wide range across individuals in preference of wood versus steel construction; some respondents have negative preference for wood, and thus prefer steel, while others prefer wood. The magnitude is very strong for some, corresponding to a difference in rating of up to 4 points. By comparison, in the second chart, preference for 60 mph coasters over 40 mph is less diverse; all respondents prefer the faster speed.

This degree of variation among respondents is known as *heterogeneity*, and in addition to estimating the parameters (coefficients) for the population (`beta.<predictor name>` as we saw above), `MCMChregress()` also estimates their variance and covariance across the population of respondents. The results are named `VCV.<predictor name>.<predictor name>` in the output, where "VCV" abbreviates *variance covariance*. When the two predictor names are the same, this gives the variance estimate for a single parameter; when they are different, it is the covariance of two parameters.

For example, we can find the population mean and variance of the wood and 60 mph parameters:

```
> summary(ride.mc2$mcmc[,c("beta.constWood", "VCV.constWood.constWood",
+ "beta.speed60","VCV.speed60.speed60")])
...
                           Mean     SD Naive SE Time-series SE
beta.constWood          -0.1128 0.1952 0.006172       0.005615
VCV.constWood.constWood  2.3458 0.3749 0.011855       0.014056
beta.speed60             1.5691 0.1618 0.005117       0.005569
VCV.speed60.speed60      0.5782 0.1351 0.004273       0.004939
...
```

The estimated variance for `constWood` is quite large at 2.34, demonstrating that there is large heterogeneity between respondents in preference for wooden roller coasters. On the other hand, the variance of the estimates for `speed60` is much smaller at 0.58. This reflects the difference in distributions that we saw in the histograms in Fig. 9.3.

You might wish to predict respondents' interest in one or more fully specified roller coaster designs, as opposed to interest in individual features. Such assessment is typical in conjoint analysis to predict product interest and is often known as *market simulation*. However, there is not yet an appropriate `predict()` function for MCMC models as there is for `lm()`. To obtain estimates of overall preference for a design, there are two choices. One option is to calculate the net level of interest by adding the columns of the MCMC draws that match your design (plus the baseline population estimates), and then summarize the level of interest for each respondent. Another option is to use a market simulation routine that compares preference between choices, such as the relative preference for your design versus some other design; an example is available in Chapman et al. [25]. We discuss preference share estimation further in Chap. 13.

One other thing we should mention with regard to this model—as is illustrated in our data simulation and Fig. 9.3 as well as in the model's assumptions—is that individuals' estimates (random coefficients) are assumed to follow a multivariate normal distribution. This means that the model assumes most people's preferences are in the middle of the distribution. If you have reason to suspect that there are separate groups with divergent and strong preferences, you might consider a mixture or latent class model, which is outside the scope of this chapter (see [137], Chap. 5).

We hope this introduction to hierarchical Bayesian models has demonstrated their value in understanding individual customers. Hierarchical modeling has become widespread in marketing because it allows us both to obtain model estimates at an individual level and to understand the diversity across customers. We'll have more to say about such models for conjoint analysis, in the form of choice-based conjoint analysis, in Chap. 13. These models are also common in CRM applications, where the goal is to estimate a likely response or outcome of some sort for individual customers. We suggest to consider a Bayesian approach anytime that you are interested to fit a linear model.

9.5 A Quick Comparison of Frequentist & Bayesian HLMs*

This is an optional section for those who completed both of the previous sections. In those sections we modeled the same data set using classical methods (Sect. 9.3) and Bayesian methods (Sect. 9.4). We saw that the estimates of the fixed effects are nearly identical in the two models (Sect. 9.4.1). What about the random, individual-level effects? How similar are they?

Before examining those effects, let's try to apply a bit of intuition to the problem. First, we might consider that the fixed effects, even with 3,200 total observations are not exactly identical between the two methods. Second, we should expect that the individual-level effects, with only 16 observations per respondent would have much more variance (because variance is inversely proportional to the square root of the number of observations). When we consider that we are estimating 8 random effects per respondent given only 16 observations, we should expect a lot of uncertainty in the estimates. Third, we should understand that neither model can be regarded as *true*, but only expected to be (one hopes) an unbiased estimate.

To compare the models here, you need to fit both the `ride.hlm2` and `ride.mc2` models as we did above (Sects. 9.3.6 and 9.4.2, respectively).

We've seen that the mean fixed effect estimates are quite similar. We can check that visually by plotting the eight parameters of each against those from the other model. First we get the fixed effects from each, then we plot them against one another and add a 45° line to see how closely they align (Fig. 9.4).

```
> fix.hlm <- fixef(ride.hlm2)
> fix.hb  <- colMeans(ride.mc2$mcmc[ , 1:8])
> plot(fix.hlm, fix.hb)
> abline(0,1)
```

Figure 9.5 shows that the fixed effects are nearly identical in the two models. Note that we use the abbreviation "HLM" to refer to the model estimated by `lme4` in order to distinguish it from "HB" for the Bayesian model, although both models are HLM yet estimated with different methods.

The random effects have to be compared within respondent. We'll do this for just one respondent, ID 196 whom we considered above. First, let's just consider the mean estimates of each random effect. We extract those using `ranef()` for the `lme4` model (Sect. 9.3.6) and `colMeans()` to take the mean effect estimated in the draws of the MCMC model (Sect. 9.4.2):

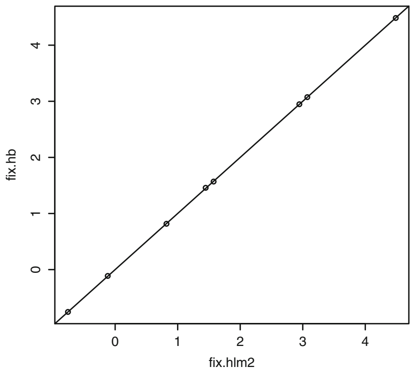

Fig. 9.4. Fixed effects from the two hierarchical models, classical and Bayesian. The Bayesian method estimates (y-axis; estimated using `MCMCpack`) are nearly identical to the classical method estimates (x-axis; estimated using `lme4`) for these simulated data.

```
> ranef(ride.hlm2)$resp.id[196, ]
    (Intercept)      speed50     speed60   speed70    height300   height400
196   -0.9300097 -0.06731524 -0.3033319 0.107412 0.004081423 -0.2346389
    constWood themeEagle
196   2.698524   -1.438102
> colMeans(ride.mc2$mcmc[ , grepl(".196", colnames(ride.mc2$mcmc),
+                                fixed=TRUE)])
b.(Intercept).196     b.speed50.196     b.speed60.196     b.speed70.196
      -1.03806213        0.44049447        0.10441996        0.03807113
  b.height300.196   b.height400.196   b.constWood.196  b.themeEagle.196
      -0.35414215       -0.55131679        2.57914806       -1.41954714
```

There are some overall similarities in the two sets of estimates for respondent 196, such as the strong negative effect for the eagle theme, relative to the same fixed effect, and strong positive for a wooden roller coasts. However, there are small to modest differences in some of the mean estimates. The MCMC process should prompt you to recall that Bayesian methods estimate not only a point estimate (the mean effect estimate reported above), but also a posterior *distribution* that reflects uncertainty.

One might compare estimates in various ways; in this case, we compare them visually. We'll do this by overlaying distribution curves for the two sets of estimates. In the case of the HB estimates, we have 1,000 MCMC draws for each parameter, so we plot the `density()` estimate of those draws. For the HLM estimates, we construct a similar density estimate in the following way: we obtain the mean effect from `ranef()` and the standard deviation of the estimation from the "`postVar`" (variance) attribute of the `ranef()` random effect estimates for one respondent, and use those parameters to draw random points from that distribution.

Doing this process one time—plotting the density of the MCMC draws and then adding a distribution plot for the mean and standard deviation of the HLM estimate—would give us a comparison of one set of parameters such as the preference for one speed or design. We iterate that to compare multiple parameters. We do that for parameters 2–5, the first four non-intercept parameters, as follows:

```
> par(mfrow=c(2,2))        # make a 2x2 plot surface
> plot.xlim <- c(-3, 3)    # define limits for the x-axis
> for (i in 2:5) {         # first four parameters only, for convenience
+   # plot the MCMC density for random effect i
+   mcmc.col <- which(grepl(".196", colnames(ride.mc2$mcmc), fixed=TRUE))[i]
+   plot(density(ride.mc2$mcmc[ , mcmc.col]), xlab="",
+        ylim=c(0, 1.4), xlim=plot.xlim,
+        main=paste("HB & lmer density:",
+                   colnames(ride.mc2$mcmc)[mcmc.col] ))
+   # add the HLM density for random effect i
+   hlm2.est <- ranef(ride.hlm2)$resp.id[196, i]             # mean estimate
+   hlm2.sd <-  sqrt(attr(ranef(ride.hlm2, condVar=TRUE)$resp.id,
+                         "postVar")[ , , 196][i, i])
+   seq.pts <- seq(from=plot.xlim[1], to=plot.xlim[2], length.out=1000) # range
+   # .. find density at x-axis points using dnorm() and add that to the plot
+   points(seq.pts, dnorm(seq.pts, mean=hlm2.est, sd=hlm2.sd),
+          col="red", pch=20, cex=0.05)
+   legend("topright", legend=c("red = lmer", "black = HB"),
+          text.col=c("red", "black"))
+ }
```

This code is lengthy but should not be difficult for you to deconstruct by this point. The two significant new elements here are that it uses `attr(..., "postVar")` to obtain the variance of the random effect estimate for the HLM model, and uses `dnorm()` to obtain a density estimate for 1,000 points that match the HLM parameter distribution estimate, which it adds to the plot with `points()`.

The resulting chart in Fig. 9.5 shows that the density estimates from the two methods are largely overlapping. It is also congruent with our intuition above, as the results are different but not enormously so, and there is no reason to suspect either method is highly discrepant because the distributions are generally similar in range and central

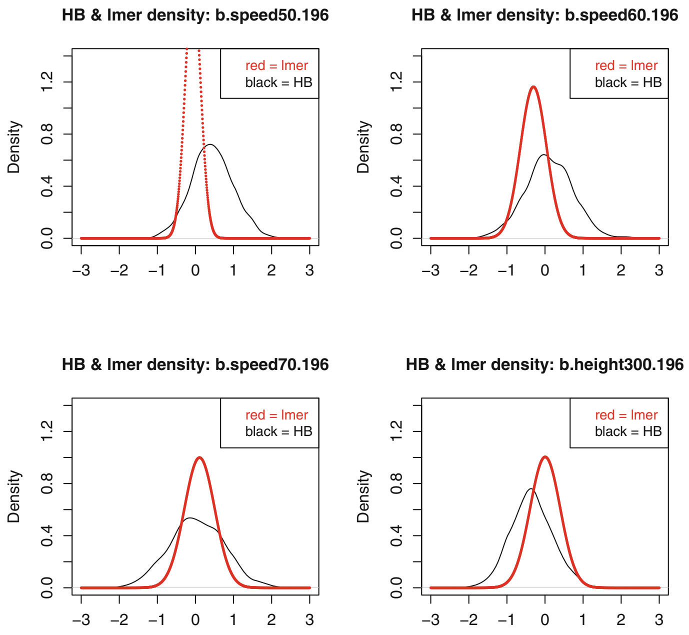

Fig. 9.5. A comparison of the estimates for four of the model parameters for respondent ID 196 in the MCMC and `lmer` results. The estimates for each respondent have substantial uncertainty but the distributions are generally similar and largely overlapping.

tendency, with just slightly higher variance in the MCMC estimates. Of course this is a comparison of only four parameters for a single respondent.

We could compare similarly across all 200 respondents, either graphically or statistically, but will leave that as an exercise for the reader. If we did so, what would we expect to see? Given that the fixed effects are nearly identical, we would expect that deviations between the models in the random effects would be close to zero and symmetric around zero. If you want to try this on your own, we can give you a preview: the median difference between the models' mean estimates of the random effects, across all 200 individuals, for the 8 parameters, ranges from -0.015 to 0.020, with a median of 0.003.

Given that the models are similar but not identical, you might wonder which is better, the classical or the Bayesian? The models themselves do not answer that; you would need to consider your assumptions, the degree to which you believe each model is appropriate (see Sect. 6.6.1), and if possible, which works better for your situation in regard to other metrics such as external validity. As we have noted, the models tend to show increasingly similar estimates with larger samples, while the Bayesian methods may yield more intuitive or useful estimates with small numbers of observations.

9.6 Learning More*

The topics in this chapter are drawn from the vast range of topics related to linear modeling, and the best general recommendation is to learn about those topics broadly, as in Harrell [74] on strategies and issues for effective regression modeling and Dobson [34] on GLM. The following notes provide further guidance on specific topics.

9.6.1 Collinearity

The best way to learn more about collinearity and how to detect and address it is to become more fluent in linear modeling in general. Good texts for learning broadly about regression modeling are Harrell [74], and Fox and Weisberg [51].

9.6.2 Logistic Regression

Logistic regression models are especially common in health sciences (modeling improvement after treatment, for instance), and much of that literature is approachable for marketers with modest translation. Hosmer et al. [78] is a standard text on such models and demonstrates the importance of model building and assessment. Binary outcomes are also often the subject of models in the machine learning community. We consider machine learning models in the context of classification in Chap. 11. A general text on those methods is Kuhn and Johnson [97].

9.6.3 Hierarchical Models

The best overall didactic text on hierarchical models is Gelman and Hill [60], which provides outstanding conceptual explanation and a breadth of models with detailed code in R. The one, comparatively minor limitation of Gelman and Hill is that its level of detail and discussion can make it difficult to determine what to do when confronted with an immediate modeling need.

Support for hierarchical models (also known as *mixed effects* models) is an evolving area in R. Besides the `lme4` package that we used, another common package is `nlme`, which has a somewhat dated companion book, Pinheiro and Bates [123]. A more up-to-date and didactic text is Galecki and Burzykowski [57].

9.6.4 Bayesian Hierarchical Models

We have provided only an introduction to hierarchical Bayes models and their importance, and have not covered the implementation issues and problems that may arise. To learn more about such models, there are technical introductions at varying levels of mathematical sophistication from Kruschke [94], Gelman et al. [61], and Rossi et al. [137]. Gelman and Hill [60] discuss hierarchical models from both Bayesian and non-Bayesian perspectives, with examples in R.

Many Bayesian texts, including several of those noted above, discuss the implementation of MCMC samplers (as in `MCMCpack`). There is a caveat: they show how to write an MCMC sampler in detail, such as the internal workings of `MCMChregress()`. That is a valuable and reusable skill but a very technical one. For some readers, it may be similar to having an automotive engineer teach you how to drive a sedan; it is highly informative but occasionally overwhelming.

`MCMCpack` includes functions for several other families of Bayesian models. A general framework that handles both mixed effects and multiple response data, using the MCMC approach, is available in the `MCMCglmm` package [68]. If you want to do hierarchical logistic regression in a Bayesian framework, you could consider `MCMCglmm` (see also Chap. 13).

9.7 Key Points

We covered a lot of material in this chapter. Following are some important lessons.

9.7.1 Collinearity

- Collinearity occurs when two or more variables are highly associated. Including them in a linear model can result in confusing, nonsensical, or misleading results, because the model cannot differentiate the contribution from each of them (Sect. 9.1).
- The *VIF* provides a measure of shared variance among variables in a model. A rule of thumb is that collinearity should be addressed for a variable when $VIF > 5$ (Sect. 9.1.2).
- Common approaches to fixing collinearity include omitting highly correlated variables, and using principal components or factor scores (see Chap. 8) instead of individual items (Sect. 9.1.2).

9.7.2 Logistic Regression

- *Logistic regression* relates a binary outcome such as purchase to predictors that may include continuous and factor variables, by modeling the variables' association with the probability of the outcome (Sect. 9.2.1).

- A logistic regression model, also known as a *logit model*, is a member of the *generalized* linear models family, and is fit using `glm( , family=binomial)` (Sect. 9.2.6).
- Coefficients in a logit model can be interpreted in terms of *odds ratios*, the degree to which they are associated with the increased or decreased likelihood of an outcome. This is done simply by exponentiating the coefficients with `exp()` (Sect. 9.2.6).
- A statistically significant result does not always mean that the model is appropriate. It is important to explore data thoroughly and to construct models on the basis of careful consideration (Sect. 9.2.7).

9.7.3 Hierarchical Linear Models

- In common marketing discussion, a *hierarchical model* estimates both group level effects and individual differences in effects. Such models are popular in marketing because they provide insight into differences among customers (*heterogeneity*) and distribution of preference. HLM are exemplified when we estimate the importance of effects for individuals as well as for an overall population (Sect. 9.3).
- Effects that are associated with all observations are known as *fixed* effects, and those that differ across various grouping levels are known as *random* effects (Sect. 9.3.1).
- These models are also known as *mixed effect* models, because the total effect for each person is composed of the effect for the overall population (the fixed effect) plus the per-individual (random) effect. We estimated an HLM using `lmer()` from the `lme4` package (Sect. 9.3.5).
- The difference between estimating hierarchical effects, as opposed to including the grouping variable as a factor in a standard linear model, is that a hierarchical model estimates *every* specified effect for each individual or group, not only a single adjustment term.
- The formula for a mixed effect model includes a grouping term, `+ ( ... | group)`. Common models have a different *intercept* by group using `(1 | group)` or different intercepts and slopes for predictors within each group using `(predictor | group)` (Sects. 9.3.5, 9.3.6). To estimate an individual-level model, the grouping term is typically the respondent identifier.
- Hierarchical models can be used to group observations at other levels than the individual level. For example, we might wish to group by store, advertising campaign, salesperson, or some other factor, if we want to estimate effects that are specific to such a grouping (Sect. 9.3.7).

- A common marketing application of HLM is conjoint analysis, to estimate both overall preference and individual differences in preference. In this chapter, we demonstrated ratings-based, or *metric* conjoint analysis (Sect. 9.3.2).

9.7.4 Bayesian Methods for Hierarchical Linear Models

- Hierarchical models in marketing are often estimated with Bayesian methods that are able to pool information and produce best estimates of both group and individual effects using potentially sparse data (Sect. 9.4.2).
- A Bayesian HLM can be estimated using `MCMChregress()` in the `MCMCpack` package (Sect. 9.4.2).
- Model coefficients from a hierarchical model are inspected using summaries of the many estimates that are collected in an `mcmc` object (Sects. 9.4.2, 9.4.3).