8

Reducing Data Complexity

Marketing data sets often have many variables—many *dimensions*—and it is advantageous to reduce these to smaller sets of variables to consider. For instance, we might have many items on a consumer survey that reflect a smaller number of underlying concepts such as *customer satisfaction* with a service, *category leadership* for a brand, or *luxury* for a product. If we can reduce the data to its underlying dimensions, we can more clearly identify the relationships among concepts.

In this chapter we consider three common methods to reduce complexity by reducing the number of dimensions in the data. *Principal component analysis* (PCA) attempts to find uncorrelated linear dimensions that capture maximal variance in the data. *Exploratory factor analysis* (EFA) also attempts to capture variance with a small number of dimensions while seeking to make the dimensions interpretable in terms of the original variables. *Multidimensional scaling* (MDS) maps similarities among observations in terms of a low-dimension space such as a two-dimensional plot. MDS can work with metric data and with non-metric data such as categorical or ordinal data.

In marketing, PCA is often associated with *perceptual maps*, which are visualizations of respondents' associations among brands or products. In this chapter we demonstrate perceptual maps for brands using PCA. We then look at ways to draw similar perceptual inferences from factor analysis and MDS.

8.1 Consumer Brand Rating Data

We investigate dimensionality using a simulated data set that is typical of consumer *brand perception* surveys. This data reflects consumer ratings of *brands* with regard to *perceptual adjectives* as expressed on survey items with the following form:

C. Chapman, E.M. Feit, *R for Marketing Research and Analytics*, Use R!,
DOI 10.1007/978-3-319-14436-8_8

> On a scale from 1 to 10—where 1 is *least* and 10 is *most*—how *[ADJECTIVE]* is *[BRAND A]*?

In this data, an observation is one respondent's rating of a brand on one of the adjectives. Two such items might be:

1. How *trendy* is *Intelligentsia Coffee*?
2. How much of a *category leader* is *Blue Bottle Coffee*?

Such ratings are collected for all the combinations of adjectives and brands of interest.

The data here comprise simulated ratings of 10 brands ("a" to "j") on 9 adjectives ("performance," "leader," "latest," "fun," and so forth), for $N = 100$ simulated respondents. The data set is provided on this book's website. We start by loading and checking the data:

```
> brand.ratings <- read.csv("http://goo.gl/IQl8nc")
> head(brand.ratings)
  perform leader latest fun serious bargain value trendy rebuy brand
1       2      4      8   8       2       9     7      4     6     a
2       1      1      4   7       1       1     1      2     2     a
...
> tail(brand.ratings)
...
999        1      1      7   5       1       1     2      5     1     j
1000       7      4      7   8       4       1     2      5     1     j
```

Each of the 100 simulated respondents has observations on each of the 10 brands, so there are 1,000 total rows. We inspect the `summary()` and `str()` to check the data quality and structure:

```
> summary(brand.ratings)
    perform          leader          latest           fun
 Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
 1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 4.000   1st Qu.: 4.000
 Median : 4.000   Median : 4.000   Median : 7.000   Median : 6.000
...
> str(brand.ratings)
'data.frame': 1000 obs. of  10 variables:
...
 $ rebuy  : int  6 2 6 1 1 2 1 1 1 1 ...
 $ brand  : Factor w/ 10 levels "a","b","c","d",..: 1 1 1 1 1 1 1 1 1 1 ...
```

We see in `summary()` that the ranges of the ratings for each adjective are 1–10. In `str()`, we see that the ratings were read as numeric while the brand labels were properly interpreted as factors. In short, the data appear to be clean and formatted appropriately.

There are nine perceptual adjectives in this data set. Table 8.1 lists the adjectives and the kind of survey text that they might reflect.

Table 8.1. Adjectives in the `brand.rating` data and examples of survey text that might be used to collect rating data

Perceptual adjective (column name)	Example survey text
perform	*Brand* has strong performance
leader	*Brand* is a leader in the field
latest	*Brand* has the latest products
fun	*Brand* is fun
serious	*Brand* is serious
bargain	*Brand* products are a bargain
value	*Brand* products are a good value
trendy	*Brand* is trendy
rebuy	I would buy from *Brand* again

8.1.1 Rescaling the Data

It is often good practice to rescale raw data. This makes data more comparable across individuals and samples. A common procedure is to *center* each variable by subtracting its mean from every observation, and then *rescale* those centered values as units of standard deviation. This is commonly called *standardizing*, *normalizing*, or *Z scoring* the data (Sect. 7.3.3).

In R, data could be standardized in this way with a mathematical expression using `mean()` and `sd()`:

```
> x <- 1:1000
> x.sc <- (x - mean(x)) / sd(x)
> summary(x.sc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.7290 -0.8647  0.0000  0.0000  0.8647  1.7290
```

As we saw in Sect. 7.3.3, a simpler way is to use `scale()` to rescale all variables at once. We never want to alter raw data, so we assign the raw values first to a new data frame `brand.sc` and alter that:

```
> brand.sc <- brand.ratings
> brand.sc[, 1:9] <- scale(brand.ratings[, 1:9])
> summary(brand.sc)
    perform            leader            latest              fun
 Min.   :-1.0888   Min.   :-1.3100   Min.   :-1.6878   Min.   :-1.84677
 1st Qu.:-1.0888   1st Qu.:-0.9266   1st Qu.:-0.7131   1st Qu.:-0.75358
 Median :-0.1523   Median :-0.1599   Median : 0.2615   Median :-0.02478
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.: 0.7842   3rd Qu.: 0.6069   3rd Qu.: 0.9113   3rd Qu.: 0.70402
 Max.   : 1.7206   Max.   : 2.1404   Max.   : 1.2362   Max.   : 1.43281
...
```

In this code we name the new data frame with extension "`.sc`" to remind ourselves that observations have been scaled. We operate on columns 1–9 because the 10th column is a factor variable for brand. We see that the mean of each adjective is

correctly 0.00 across all brands because the data is rescaled. Observations on the adjectives have a spread (difference between `min` and `max`) of roughly 3 standard deviation units. This means the distributions are *platykurtic*, flatter than a standard normal distribution, because we would expect a range of more than 4 standard deviation units for a sample of this size. (Platykurtosis is a common property of survey data, due to floor and ceiling effects.)

We use `corrplot()` for initial inspection of bivariate relationships among the variables:

```
> library(corrplot)
> corrplot(cor(brand.sc[, 1:9]), order="hclust")
```

As before, we plot columns 1–9 because the 10th column is the non-numeric brand label. In `corrplot()`, the argument `order="hclust"` reorders the rows and columns according to variables' similarity in a hierarchical cluster solution (see Sect. 11.3.2 for more on hierarchical clustering). The result is shown in Fig. 8.1, where we see that the ratings seem to group into three clusters of similar variables, a hypothesis we examine in detail in this chapter.

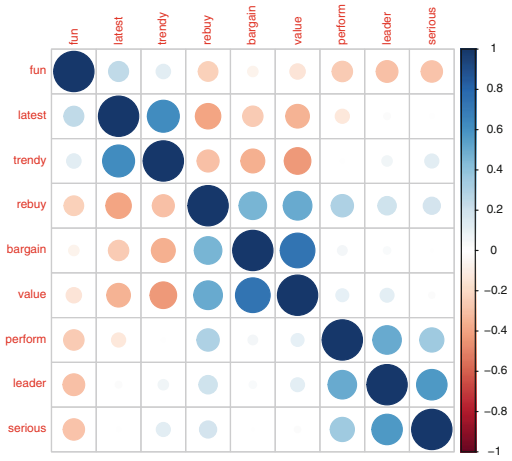

Fig. 8.1. Correlation plot for the simulated consumer brand ratings. This visualization of the basic data appears to show three general clusters that comprise *fun/latest/trendy*, *rebuy/bargain/value*, and *perform/leader/serious*, respectively.

8.1.2 Aggregate Mean Ratings by Brand

Perhaps the simplest business question in these data is: "What is the average (mean) position of the brand on each adjective?" We can use `aggregate()` (see Sects. 3.4.5 and 5.2.1) to find the mean of each variable by brand:

```
> brand.mean <- aggregate(. ~ brand, data=brand.sc, mean)
> brand.mean
   brand     perform     leader     latest        fun     serious     bargain
1      a -0.88591874 -0.5279035  0.4109732  0.6566458 -0.91894067  0.21409609
2      b  0.93087022  1.0707584  0.7261069 -0.9722147  1.18314061  0.04161938
...
```

Before proceeding, we perform a bit of housekeeping on the new `brand.mean` object. We name the rows with the brand labels that `aggregate()` put into the `brand` column, and then we remove that column as redundant:

```
> rownames(brand.mean) <- brand.mean[, 1] # use brand for the row names
> brand.mean <- brand.mean[, -1]          # remove brand name column
```

The resulting matrix is now nicely formatted with brands by row and adjective means in the columns:

```
> brand.mean
      perform     leader     latest        fun     serious     bargain
a -0.88591874 -0.5279035  0.4109732  0.6566458 -0.91894067  0.21409609
b  0.93087022  1.0707584  0.7261069 -0.9722147  1.18314061  0.04161938
...
```

A *heatmap* is a useful way to examine such results because it colors data points by the intensities of their values. We use `heatmap.2()` from the `gplots` package [158] with colors from the `RColorBrewer` package [121] (install those if you need them):

```
> library(gplots)
> library(RColorBrewer)
> heatmap.2(as.matrix(brand.mean),
+           col=brewer.pal(9, "GnBu"), trace="none", key=FALSE, dend="none",
+           main="\n\n\n\n\nBrand attributes")
```

`heatmap.2()` is a complex function. In the code above, we coerce `brand.mean` to be a matrix as `heatmap.2()` expects. We color the map using greens and blues from `RColorBrewer`'s "GnBu" palette and turn off a few options that otherwise clutter the heatmap (`trace`, `key`, and `dendrogram`). We improve title alignment by adding blank lines with `\n` before the title text.

The resulting heatmap is shown in Fig. 8.2. In this chart's green-to-blue (`"GnBu"`) palette a green color indicates a low value and dark blue indicates a high value; lighter colors are for values in the middle of the range. The brands are clearly perceived differently with some brands rated high on performance and leadership (brands *b* and *c*) and others rated high for value and intention to rebuy (brands *f* and *g*). By default, `heatmap.2()` sorts the columns and rows in order to emphasize similarities and patterns in the data, which is why the rows and columns in Fig. 8.2 are ordered in an unexpected way. It does this using a form of hierarchical clustering (see Sect. 11.3.2).

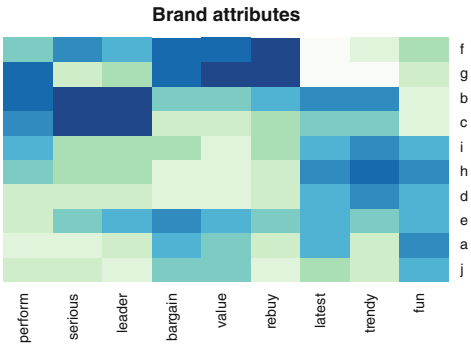

Fig. 8.2. A heatmap for the mean of each adjective by brand. Brands *f* and *g* are similar—with high ratings for *rebuy* and *value* but low ratings for *latest* and *fun*. Other groups of similar brands are *b/c*, *i/h/d*, and *a/j*.

Looking at Figs. 8.1 and 8.2 we could guess at the groupings and relationships of adjectives and brands. For example, there is similarity in the color pattern across columns for the *bargain/value/rebuy*; a brand that is high on one tends to be high on another. But it is better to formalize such insight, and the remainder of this chapter discusses how to do so.

8.2 Principal Component Analysis and Perceptual Maps

PCA recomputes a set of variables in terms of linear equations, known as *components*, that capture linear relationships in the data [87]. The first component captures as much of the variance as possible from all variables as a single linear function. The second component captures as much variance as possible that remains after the first component. This continues until there are as many components as there are variables. We can use this process to reduce data complexity by retaining and analyzing only a subset of those components—such as the first one or two components—that explain a large proportion of the variation in the data.

8.2.1 PCA Example

We explore PCA first with a simple data set to see and develop intuition about what is happening. We create highly correlated data by copying a random vector `xvar` to a new vector `yvar` while replacing half of the data points. Then we repeat that procedure to create `zvar` from `yvar`:

```
> set.seed(98286)
> xvar <- sample(1:10, 100, replace=TRUE)
> yvar <- xvar
> yvar[sample(1:length(yvar), 50)] <- sample(1:10, 50, replace=TRUE)
> zvar <- yvar
> zvar[sample(1:length(zvar), 50)] <- sample(1:10, 50, replace=TRUE)
> my.vars <- cbind(xvar, yvar, zvar)
```

yvar will be correlated with xvar because 50 of the observations are identical while 50 are newly sampled random values. Similarly, zvar keeps 50 values from yvar (and thus also inherits some from xvar, but fewer). We compile those three vectors into a matrix.

We check one of the three possible bivariate plots along with the correlation matrix. If we simply plotted the raw data, there would be many overlapping values because the responses are discrete (integers 1–10). To separate and visualize multiple points with the same values, we jitter() them (Sect. 4.6.1):

```
> plot(yvar ~ xvar, data=jitter(my.vars))
> cor(my.vars)
          xvar      yvar      zvar
xvar 1.0000000 0.5969717 0.2496469
yvar 0.5969717 1.0000000 0.5231468
zvar 0.2496469 0.5231468 1.0000000
```

The bivariate plot in Fig. 8.3 shows a clear linear trend for yvar vs. xvar on the diagonal. In the correlation matrix, xvar correlates highly with yvar and less so with zvar, as expected, and yvar has strong correlation with zvar (using the rules of thumb from Sect. 4.5).

Using intuition, what would we expect the components to be from this data? First, there is shared variance across all three variables because they are positively correlated. So we expect to see one component that picks up that association of all three variables. After that, we expect to see a component that shows that xvar and zvar are more differentiated from one another than either is from yvar. That implies that yvar has a unique position in the data set as the only variable to correlate highly with both of the others, so we expect one of the components to reflect this uniqueness of yvar.

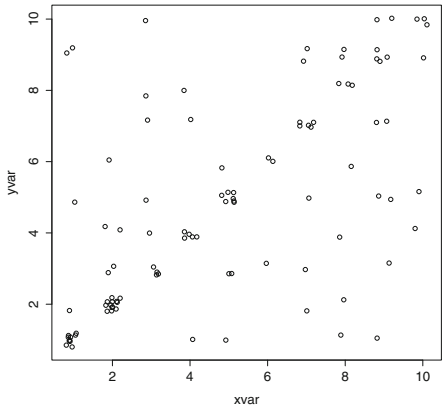

Fig. 8.3. Scatterplot of correlated data with discrete values, using jitter() to separate the values slightly for greater visual impact of overlapping points.

Let's check the intuition. We use `prcomp()` to perform PCA:

```
> my.pca <- prcomp(my.vars)
> summary(my.pca)
Importance of components:
                          PC1    PC2    PC3
Standard deviation     3.9992 2.4381 1.6269
Proportion of Variance 0.6505 0.2418 0.1077
Cumulative Proportion  0.6505 0.8923 1.0000
```

There are three components because we have three variables. The first component accounts for 65 % of the explainable linear variance, while the second accounts for 24 %, leaving 11 % for the third component. How are those components related to the variables? We check the rotation matrix, which is helpfully printed by default for a PCA object:

```
> my.pca
Standard deviations:
[1] 3.999154 2.438079 1.626894

Rotation:
            PC1          PC2         PC3
xvar -0.6156755  0.63704774  0.4638037
yvar -0.6532994 -0.08354009 -0.7524766
zvar -0.4406173 -0.76628404  0.4676165
```

Interpreting PCA rotation loadings is difficult because of the multivariate nature—factor analysis is a better procedure for interpretation, as we will see later in this chapter—but we examine the loadings here for illustration and comparison to our expectations. In component 1 (PC1) we see loading on all 3 variables as expected from their overall shared variance (the negative direction is not important; the key is that they are all in the same direction).

In component two, we see that `xvar` and `zvar` are differentiated from one another as expected, with loadings in opposite directions. Finally, in component 3, we see residual variance that differentiates `yvar` from the other two variables and is consistent with our intuition about `yvar` being unique.

In addition to the loading matrix, PCA has computed scores for each of the principal components that express the underlying data in terms of its loadings on those components. Those are present in the PCA object as the `$x` matrix, where the columns (`[ , 1]`, `[ , 2]`, and so forth) may be used to obtain the values of the components for each observation. We can use a small number of those columns in place of the original data to obtain a set of observations that captures much of the variation in the data.

A less obvious feature of PCA, but implicit in the definition, is that extracted PCA components are *uncorrelated* with one another, because otherwise there would be more linear variance that could have been captured. We see this in the `scores`

returned for observations in a PCA model, where the off-diagonal correlations are effectively zero (approximately 10^{-15} as shown in R's scientific notation):

```
> cor(my.pca$x)    # components have zero correlation
              PC1           PC2           PC3
PC1 1.000000e+00  4.808932e-16  1.768720e-15
PC2 4.808932e-16  1.000000e+00 -1.174441e-15
PC3 1.768720e-15 -1.174441e-15  1.000000e+00
```

8.2.2 Visualizing PCA

A good way to examine the results of PCA is to map the first few components, which allows us to visualize the data in a lower-dimensional space. A common visualization is a *biplot*, a two-dimensional plot of data points with respect to the first two PCA components, overlaid with a projection of the variables on the components. We use `biplot()` to generate this:

```
> biplot(my.pca)
```

The result is Fig. 8.4, where every data point is plotted (and labeled by row number) according to its values on the first two components. Such plots are especially helpful when there are a smaller number of points (as we will see below for brands) or when there are clusters (as we see in Chap. 11).

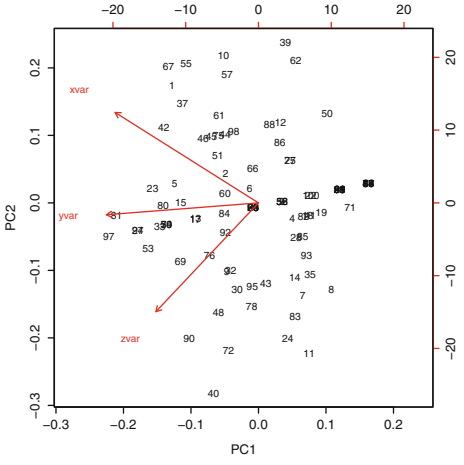

Fig. 8.4. A `biplot()` of a PCA solution for the simple, constructed example, showing data points plotted on the first two components.

In Fig. 8.4, there are arrows that show the best fit of each of the variables on the principal components—a projection of the variables onto the two-dimensional space of the first two PCA components, which explain a large part of the variation in the data. These are useful to inspect because the *direction* and *angle* of the arrows reflect

the relationship of the variables; a closer angle indicates higher positive association, while the relative direction indicates positive or negative association of the variables.

In the present case, we see in the variable projections (arrows) that `yvar` is closely aligned with the first component (X axis). In the relationships among the variables themselves, we see that `xvar` and `zvar` are more associated with `yvar`, relative to the principal components, than either is with the other. Thus, this visually matches our interpretation of the correlation matrix and loadings above.

By plotting against principal components, a biplot benefits from the fact that components are uncorrelated; this helps to disperse data on the chart because the x- and y-axes are independent. When there are several components that account for substantial variance, it is also useful to plot components beyond the first and second. This can be done with the `choices` argument to `biplot()`.

8.2.3 PCA for Brand Ratings

Let's look at the principal components for the brand rating data (refer to Sect. 8.1 above if you need to load the data). We find the components with `prcomp()`, selecting just the rating columns 1–9:

```
> brand.pc <- prcomp(brand.sc[, 1:9])
> summary(brand.pc)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7
Standard deviation     1.726 1.4479 1.0389 0.8528 0.79846 0.73133 0.62458 ...
Proportion of Variance 0.331 0.2329 0.1199 0.0808 0.07084 0.05943 0.04334 ...
Cumulative Proportion  0.331 0.5640 0.6839 0.7647 0.83554 0.89497 0.93831 ...
```

The default `plot()` for a PCA is a *scree plot*, which shows the successive proportion of additional variance that each component adds. We plot this as a line chart using `type="l"` (lower case "L" for *line*):

```
> plot(brand.pc, type="l")
```

The result is Fig. 8.5. A scree plot is often interpreted as indicating where additional components are not worth the complexity; this occurs where the line has an *elbow*, a kink in the angle of bending, a somewhat subjective determination. In Fig. 8.5, the elbow occurs at either component three or four, depending on interpretation; and this suggests that the first two or three components explain most of the variation in the observed brand ratings.

A `biplot()` of the first two principal components—which `biplot()` selects by default for a PCA object—reveals how the rating adjectives are associated:

```
> biplot(brand.pc)
```

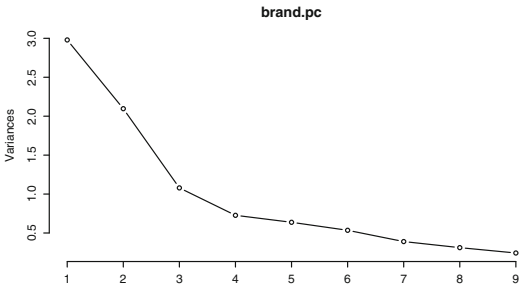

Fig. 8.5. A scree `plot()` of a PCA solution shows the successive variance accounted by each component. For the brand rating data, the proportion largely levels out after the third component.

We see the result in Fig. 8.6, where adjectives map in four regions: category leadership ("serious," "leader," and "perform" in the upper right), value ("rebuy," "value," and "bargain"), trendiness ("trendy" and "latest"), and finally "fun" on its own.

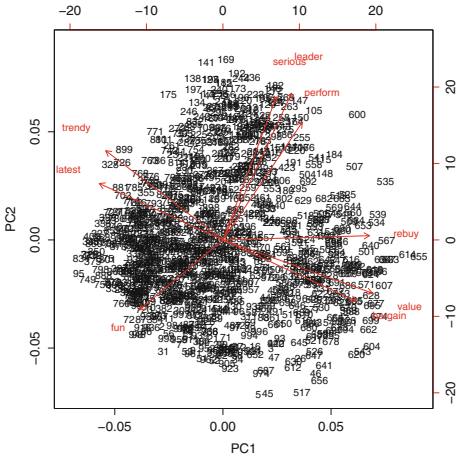

Fig. 8.6. A biplot of an initial attempt at PCA for consumer brand ratings. Although we see adjective groupings on the variable loading *arrows* in *red*, and gain some insight into the areas where ratings cluster (as dense areas of observation points), the chart would be more useful if the data were first aggregated by brand.

But there is a problem: the plot of individual respondents' ratings is too dense and it does not tell us about the brand positions! A better solution is to perform PCA using *aggregated* ratings by brand. First we remind ourselves of the data that compiled the mean rating of each adjective by brand as we found above using `aggregate()` (see Sect. 8.1). Then we extract the principal components:

```
> brand.mean
      perform     leader     latest        fun     serious     bargain
a -0.88591874 -0.5279035  0.4109732  0.6566458 -0.91894067  0.21409609
b  0.93087022  1.0707584  0.7261069 -0.9722147  1.18314061  0.04161938
...
> brand.mu.pc <- prcomp(brand.mean, scale=TRUE)
> summary(brand.mu.pc)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7
Standard deviation     2.1345 1.7349 0.7690 0.61498 0.50983 0.36662 0.21506
```

```
Proportion of Variance 0.5062 0.3345 0.0657 0.04202 0.02888 0.01493 0.00514
Cumulative Proportion  0.5062 0.8407 0.9064 0.94842 0.97730 0.99223 0.99737
...
```

In the call to `prcomp()`, we added `scale=TRUE` in order to rescale the data; even though the raw data was already rescaled, the aggregated means have a somewhat different scale than the standardized data itself. The results show that the first two components account for 84 % of the explainable variance in the mean ratings, so we focus on interpreting results with regard to them.

8.2.4 Perceptual Map of the Brands

A biplot of the PCA solution for the mean ratings gives an interpretable *perceptual map*, showing where the brands are placed with respect to the first two principal components. We use `biplot()` on the PCA solution for the mean rating by brand:

```
> biplot(brand.mu.pc, main="Brand positioning", cex=c(1.5, 1))
```

We plot the brand labels with a 50 % larger font using the *c*haracter *ex*pansion argument `cex=c(1.5, 1)`. The result is Fig. 8.7.

Before interpreting the new map, we first check that using mean data did not greatly alter the structure. Figure 8.7 shows a different spatial rotation of the adjectives, compared to Fig. 8.6, but the spatial position is arbitrary and the new map has the same overall grouping of adjectives and relational structure (for instance, seeing as in Fig. 8.6 that "serious" and "leader" are closely related while "fun" is rather distant from other adjectives). Thus the variable positions on the components are consistent with PCA on the full set of observations, and we go ahead to interpret the graphic.

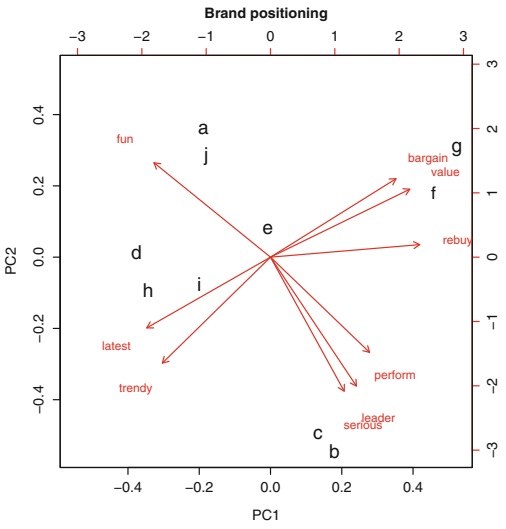

Fig. 8.7. A perceptual map of consumer brands with `biplot()` for aggregate mean rating by brand. This shows components almost identical to those in Fig. 8.6 (although spatially rotated) but the mean brand positions are clear.

What does the map tell us? First we interpret the adjective clusters and relationships and see four areas with well differentiated sets of adjectives and brands that are positioned in proximity. Brands *f* and *g* are high on "value," for instance, while *a* and *j* are relatively high on "fun," which is opposite in direction from leadership adjectives ("leader" and "serious").

With such a map, one might form questions and then refer to the underlying data to answer them. For instance, suppose that you are the brand manager for brand *e*. What does the map tell you? For one thing, your brand is in the center and thus appears not to be well-differentiated on any of the dimensions. That could be good or bad, depending on your strategic goals. If your goal is to be a safe brand that appeals to many consumers, then a relatively undifferentiated position like *e* could be desirable. On the other hand, if you wish your brand to have a strong, differentiated perception, this finding would be unwanted (but important to know).

What should you do about the position of your brand *e*? Again, it depends on the strategic goals. If you wish to increase differentiation, one possibility would be to take action to shift your brand in some direction on the map. Suppose you wanted to move in the direction of brand *c*. You could look at the specific differences from *c* in the data:

```
> brand.mean["c", ] - brand.mean["e", ]
   perform    leader     latest        fun  serious   bargain      value ...
c 1.214314 0.9699315 -0.5587936 -1.140567 1.180621 -1.158594 -0.8588416 ...
```

This shows you that *e* is relatively stronger than *c* on "value" and "fun", which suggests dialing down messaging or other attributes that reinforce those (assuming, of course, that you truly want to move in the direction of *c*). Similarly, *c* is stronger on "perform" and "serious," so those could be aspects of the product or message for *e* to strengthen.

Another option would be *not* to follow another brand but to aim for differentiated space where no brand is positioned. In Fig. 8.7, there is a large gap between the group *b* and *c* on the bottom of the chart, versus *f* and *g* on the upper right. This area might be described as the "value leader" area or similar.

How do we find out how to position there? Let's assume that the gap reflects approximately the average of those four brands (see Sect. 8.2.5 for some of the risks with this assumption). We can find that average using `colMeans()` on the brands' rows, and then take the difference of *e* from that average:

```
> colMeans(brand.mean[c("b", "c", "f", "g"), ]) - brand.mean["e", ]
   perform    leader     latest        fun   serious    bargain      value
e 1.174513 0.3910396 -0.9372789 -0.9337707 0.5732131 -0.2502787 0.07921355
...
```

This suggests that brand *e* could target the gap by increasing its emphasis on performance while reducing emphasis on "latest" and "fun."

To summarize, when you wish to compare several brands across many dimensions, it can be helpful to focus on just the first two or three principal components that explain variation in the data. You can select how many components to focus on

using a scree plot, which shows how much variation in the data is explained by each principal component. A perceptual map plots the brands on the first two principal components, revealing how the observations relate to the underlying dimensions (the components).

PCA may be performed using survey ratings of the brands (as we have done here) or with objective data such as price and physical measurements, or with a combination of the two. In any case, when you are confronted with multidimensional data on brands or products, PCA visualization is a useful tool for understanding differences in the market.

8.2.5 Cautions with Perceptual Maps

There are three important caveats in interpreting perceptual maps. First, you must choose the level and type of aggregation carefully. We demonstrated the maps using mean rating by brand, but depending on the data and question at hand, it might be more suitable to use median (for ordinal data) or even modal response (for categorical data). You should check that the dimensions are similar for the full data and aggregated data before interpreting aggregate maps. You can do this by examining the variable positions and relationships in biplots of both aggregated data (such as means) and raw data (or a random subset of it), as we did above.

Second, the relationships are strictly relative to the product category and the brands and adjectives that are tested. In a different product category, or with different brands, adjectives such as "fun" and "leader" could have a very different relationship. Sometimes simply adding or dropping a brand can change the resulting map significantly because the positions are relative. In other words, if a new brand enters the market (or one's analysis), the other positions may change substantially. One must also be confident that all of the key perceptions (adjectives, in this example) have been assessed. One way to assess sensitivity here is to run PCA and biplot on a few different samples from your data, such as 80 % of your observations, and perhaps dropping an adjective each time. If the maps are similar across those samples, you may feel more confident in their stability.

Third, it is frequently misunderstood that the positions of brands in such a map depend on their relative positioning in terms of the principal components, which are constructed composites of all dimensions. This means that *the strength of a brand on a single adjective cannot be read directly from the chart*. For instance, in Fig. 8.7, it might appear that brands *b* and *c* are weaker than *d*, *h*, and *i* on "latest" but are similar to one another. In fact, *b* is the single strongest brand on "latest" while *c* is weak on that adjective. Overall, *b* and *c* are quite similar to one another in terms of their scores on the two components that aggregate all of the variables (adjectives), but they are not necessarily similar on any single variable. Another way to look at this is that when we use PCA to focus on the first one or two dimensions in the data, we are looking at the largest-magnitude similarities, which may obscure smaller differences that do not show up strongly in the first one or two dimensions.

This last point is a common area of confusion with analysts and stakeholders who want to read adjective positions directly from a biplot. We recommend to explain that positions are not absolute but are *relative*. We often explain positions with language such as, "compared to its position on other attributes, brand X is *relatively* differentiated by perceptions of strength (or weakness) on such-and-such attribute."

Despite these caveats, perceptual maps can be a valuable tool. We use them primarily to form hypotheses and to provide material to inform strategic analyses of brand and product positioning. If they are used in that way—rather than as absolute assessments of position—they can contribute to engaging discussions about position and potential strategy.

Although we illustrated PCA with brand position, the same kind of analysis could be performed for product ratings, position of consumer segments, ratings of political candidates, evaluations of advertisements, or any other area where you have metric data on multiple dimensions that is aggregated for a modest number of discrete entities of interest.

In Chap. 9 we will see the usefulness of PCA for highly correlated data. By extracting components, one can derive a reduced set of variables that captures as much of the variance as desired, yet where each of the measures is independent of the others.

8.3 Exploratory Factor Analysis

EFA is a family of techniques to assess the relationship of *constructs* (concepts) in surveys and psychological assessments. Factors are regarded as *latent variables* that cannot be observed directly, but are imperfectly assessed through their relationship to other variables.

In psychometrics, canonical examples of factors occur in psychological and educational testing. For example, "intelligence," "knowledge of mathematics," and "anxiety" are all abstract concepts (constructs) that are not directly observable in themselves. Instead, they are observed empirically through multiple behaviors, each one of which is an imperfect indicator of the underlying latent variable. These observed values are known as *manifest variables* and include indicators such as test scores, survey responses, and other empirical behaviors. EFA attempts to find the degree to which latent, composite *factors* account for the observed variance of those manifest variables.

In marketing, we often observe a large number of variables that we believe should be related to a smaller set of underlying constructs. For example, we cannot directly observe *customer satisfaction* but we might observe responses on a survey that asks about different aspects of a customer's experience, jointly representing different

facets of the underlying construct *satisfaction*. Similarly, we cannot directly observe *purchase intent*, *price sensitivity*, or *category involvement* but we can observe multiple behaviors that are related to them.

In this section, we use EFA to examine respondents' attitudes about brands, using the brand rating data from above (Sect. 8.1) to uncover the latent dimensions in the data. Then we assess the brands in terms of those estimated latent factors.

8.3.1 Basic EFA Concepts

The result of EFA is similar to PCA: a matrix of factors (similar to PCA components) and their relationship to the original variables (*loadings* of the factors on the variables). Unlike PCA, EFA attempts to find solutions that are maximally *interpretable* in terms of the manifest variables. In general, it attempts to find solutions in which a small number of loadings for each factor are very high, while other loadings for that factor are low. When this is possible, that factor can be interpreted in terms of that small set of variables.

To accomplish this, EFA uses *rotations* that start with an uncorrelated (*orthogonal*) mathematical solution and then mathematically alter the solution to explain identical variance but with different loadings on the original variables. There are many such rotations available, and they typically share the goals of maximizing the loadings on a few variables while making factors as distinct as possible from one another.

Instead of reviewing that mathematically (see [119]), let's consider a loose analogy. One might think about EFA in terms of a pizza topped with large items such as tomato slices and mushrooms that will be cut into a certain number of slices. The pizza could be rotated and cut in an infinite number of ways that are all mathematically equivalent insofar as they divide up the same underlying structure.

However, some rotations are more useful than others because they fall in-between the large items rather than dividing them. When this occurs, one might have a "tomato slice," a "mushroom slice," a "half-and-half tomato and mushroom slice," and so forth. By rotating and cutting differently, one makes the underlying substance more interpretable relative to one's goals (such as having differentiated pizza slices). No rotation is inherently better or worse, but some are more useful than others. Similarly, the manifest variables in EFA can be sliced in many ways according to one's goals for interpreting the latent factors. We will see how this works in Sect. 8.3.3.

Because EFA produces results that are interpretable in terms of the original variables, an analyst may be able to interpret and act on the results in ways that would be difficult with PCA. For instance, EFA can be used to refine a survey by keeping items with high loading on factors of interest while cutting items that do not load highly. EFA is also useful to investigate whether a survey's items actually go together in a way that is consistent with expectations.

For example, if we have a 10-item survey that is supposed to assess the single construct *customer satisfaction*, it is important to know whether those items in fact go together in a way that can be interpreted as a single factor, or whether they instead reflect multiple dimensions that we might not have considered. Before interpreting multiple items as assessing a single concept, one might wish to test that it is appropriate to do so. In this chapter, we use EFA to investigate such structure. In Chap. 10, we will see how to test whether one's data are in fact consistent with an asserted structure.

EFA serves as a data reduction technique in three broad senses:

1. In the technical sense of dimensional reduction, we can use *factor scores* instead of a larger set of items. For instance, if we are assessing satisfaction, we could use a single satisfaction score instead of several separate items. (In Sect. 9.1.2 we review how this is also useful when observations are correlated.)
2. We can reduce uncertainty. If we believe *satisfaction* is imperfectly manifest in several measures, the combination of those will have less noise than the set of individual items.
3. We might also reduce data collection by focusing on items that are known to have high contribution to factors of interest. If we discover that some items are not important for a factor of interest, we can discard them from data collection efforts.

In this chapter we use the brand rating data to ask the following questions: How many latent factors are there? How do the survey items map to the factors? How are the brands positioned on the factors? What are the respondents' factor scores?

8.3.2 Finding an EFA Solution

The first step in EFA is to determine the number of factors to estimate. There are various ways to do this, and two traditional methods are to use a scree plot (Sect. 8.2.3), and to retain factors where the *eigenvalue* (a metric for proportion of variance explained) is greater than 1.0. An eigenvalue of 1.0 corresponds to the amount of variance that might be attributed to a single independent variable; a factor that captures less variance than such an item may be considered relatively uninteresting.

As we saw in Sect. 8.2.3, a scree plot of the brand rating data suggests two or three components. The `nFactors` package [130] (install if necessary) formalizes this analysis with `nScree()`:

```
> library(nFactors)
> nScree(brand.sc[, 1:9])
  noc naf nparallel nkaiser
1   3   2         3       3
```

`nScree()` applies several methods to estimate the number of factors from scree tests, and in the present case three of the four methods suggest that the data set has 3 factors. We can examine the eigenvalues using `eigen()` on a correlation matrix:

```
> eigen(cor(brand.sc[, 1:9]))
$values
[1] 2.9792956 2.0965517 1.0792549 0.7272110 0.6375459 0.5348432 0.3901044
...
```

The first three eigenvalues are greater than 1.0, although barely so for the third value. This again suggests 3—or possibly 2—factors.

The final choice of a model depends on whether it is useful. For EFA, a best practice is to check a few factor solutions, including the ones suggested by the scree and eigenvalue results. Thus, we test a 3-factor solution and a 2-factor solution to see which one is more useful.

An EFA model is estimated with `factanal(x, factors=K)`, where K is the number of factors to fit. For a 2-factor solution, we write:

```
> factanal(brand.sc[, 1:9], factors=2)
...
Loadings:
        Factor1 Factor2
perform          0.600
leader           0.818
latest  -0.451
fun     -0.137  -0.382
serious          0.686
bargain  0.803
value    0.873   0.117
trendy  -0.534
rebuy    0.569   0.303
...
```

We have removed all of the information except for the loadings because those are the most important to interpret (see “Learning More” in this chapter for material that explains much more about EFA and the output of such procedures). Some of the factor loadings are near zero, and are not shown; this makes EFA potentially easier to interpret than PCA.

In the 2-factor solution, factor 1 loads strongly on “bargain” and “value,” and therefore might be interpreted as a “value” factor while factor 2 loads on “leader” and “serious” and thus might be regarded as a “category leader” factor.

This is not a bad interpretation, but let's compare it to a 3-factor solution:

```
> factanal(brand.sc[, 1:9], factors=3)
...
Loadings:
        Factor1 Factor2 Factor3
perform          0.607
leader           0.810   0.106
latest  -0.163           0.981
fun             -0.398   0.205
serious          0.682
bargain  0.826          -0.122
value    0.867          -0.198
trendy  -0.356           0.586
rebuy    0.499   0.296  -0.298
```

The 3-factor solution retains the "value" and "leader" factors and adds a clear "latest" factor that loads strongly on "latest" and "trendy." This adds a clearly interpretable concept to our understanding of the data. It also aligns with the bulk of suggestions from the scree and eigen tests, and fits well with the perceptual maps we saw in Sect. 8.2.4, where those adjectives were in a differentiated space. So we regard the 3-factor model as superior to the 2-factor model because the factors are more interpretable.

8.3.3 EFA Rotations

As we described earlier, a factor analysis solution can be rotated to have new loadings that account for the same proportion of variance. Although a full consideration of rotations is out of scope for this book, there is one issue worth considering in any EFA: do you wish to allow the factors to be *correlated* with one another or not?

You might think that one should let the data decide. However, the question of whether to allow correlated factors is less a question about the *data* than it is about your *concept* of the underlying latent factors. Do you think the factors should be conceptually independent, or does it make more sense to consider them to be related? An EFA rotation can be obtained under either assumption.

The default in `factanal()` is to find factors that have zero correlation (using a *varimax* rotation). In case you're wondering how this differs from PCA, it differs mathematically because EFA finds latent variables that may be observed with error (see [119]) whereas PCA simply recomputes transformations of the observed data. In other words, EFA focuses on the underlying latent dimensions, whereas PCA focuses on transforming the dimensionality of the data.

Returning to our present data, we might judge that *value* and *leader* are reasonably expected to be related; in many categories, the leader can command a price

premium, and thus we might expect those two latent constructs to be negatively correlated rather than independent of one another. This suggests that we could allow correlated factors in our solution. This is known as an *oblique* rotation ("oblique" because the dimensional axes are not perpendicular but are skewed by the correlation between factors).

A common oblique rotation is the "oblimin" rotation from the `GPArotation` package [11] (install if necessary). We add that to our 3-factor model with `rotation="oblimin"`:

```
> library(GPArotation)
> (brand.fa.ob <- factanal(brand.sc[, 1:9], factors=3, rotation="oblimin"))
...
Loadings:
        Factor1 Factor2 Factor3
perform          0.601
leader           0.816
latest                   1.009
fun             -0.381   0.229
serious          0.689
bargain  0.859
value    0.880
trendy  -0.267   0.128   0.538
rebuy    0.448   0.255  -0.226
...
Factor Correlations:
        Factor1 Factor2 Factor3
Factor1  1.0000  -0.388  0.0368
Factor2 -0.3884   1.000 -0.1091
Factor3  0.0368  -0.109  1.0000
...
```

When we compare this oblimin result to the default varimax rotation above, there are two main differences. First, the loadings are slightly different for the relationships of the factors to the adjectives. However, the loadings are similar enough in this case that there is no substantial change in how we would interpret the factors. There are still factors for "value," "leader," and "latest."

Second, the result includes a factor correlation matrix showing the relationships between the estimated latent factors. Factor 1 (value) is negatively correlated with Factor 2 (leader), $r = -0.39$, and is essentially uncorrelated with Factor 3 (latest), $r = 0.037$.

The negative correlation between factors 1 and 2 is consistent with our theory that brands that are leaders are less likely to be value brands, and thus we think this is a more interpretable result. However, in other cases a correlated rotation may or may not be a better solution than an orthogonal one; that is largely an issue to be decided on the basis of domain knowledge and interpretive utility rather than statistics.

In the output above, the item-to-factor loadings are displayed. In the returned model object, those are present as the `$loadings` element. We can the visualize item-factor relationships with a heatmap of `$loadings`:

```
> library(gplots)
> library(RColorBrewer)
> heatmap.2(brand.fa.ob$loadings,
+           col=brewer.pal(9, "Greens"), trace="none", key=FALSE, dend="none",
+           Colv=FALSE, cexCol = 1.2,
+           main="\n\n\n\n\nFactor loadings for brand adjectives")
```

The result is Fig. 8.8, which shows a distinct separation of items into 3 factors, which are roughly interpretable as *value*, *leader*, and *latest*. Note that the item `rebuy`, which reflects stated intention to repurchase, loads on both Factor1 (*value*) and Factor2 (*leader*). This suggests that in our simulated data, consumers say they would rebuy a brand for either reason, because it is a good value or because it is a leader.

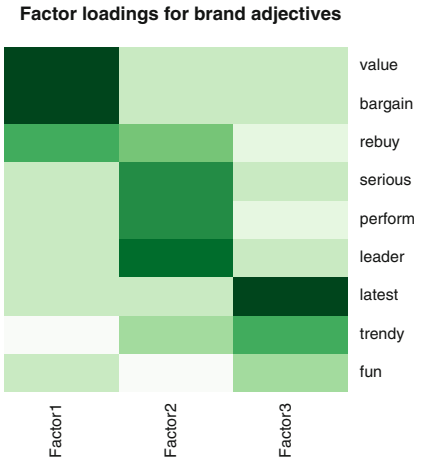

Fig. 8.8. A heatmap of item-factor loadings.

Another useful graphic for factor analysis models is a *path diagram*, which shows latent variables and the individual items that load on them.

The `semPlot` package (install if needed) will draw a visual representation of a factor analysis model. We use the procedure `semPaths()` to draw the paths. It is a complex command and we add several arguments as explained below:

```
> library(semPlot)
> semPaths(brand.fa.ob, what="est", residuals=FALSE,
+          cut=0.3, posCol=c("white", "darkgreen"), negCol=c("white", "red"),
+          edge.label.cex=0.75, nCharNodes=7)
```

First we will explain the `semPaths()` call. We plotted the `brand.fa.ob` model as fit above. To draw the loading estimates, we requested `what="est"`. We omit the residual estimates for manifest variables (an advanced topic we don't cover in this book) using `residuals=FALSE`. Then we cut loadings with absolute magnitude < 0.3 by adding `cut=0.3` and the options `posCol=c("white", "darkgreen")` and `negCol=c("white", "red")`. The `posCol` argument

says that positive *loadings* < 0.3 should be colored `white` (and thus not appear in the output), while *loadings* > 0.3 should be `darkgreen`. The `negCol` argument similarly excludes or colors `red` the *loadings* < 0. We adjust the loadings' text size with `edge.label.cex`, and create room to spell out full variable names with `nCharNodes`.

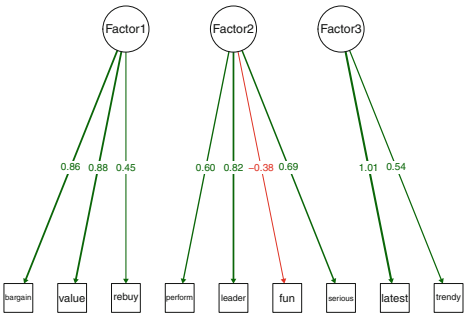

Fig. 8.9. A path diagram for the factor analysis solution, which clearly displays the three factors and their item loadings ($|loadings| < 0.3$ are excluded). The graphic is generated with `semPaths()` from the `semPlot` package.

The result is shown in Fig. 8.9. Luckily, interpreting the path diagram is easier than the code to create it! Latent variables are shown at the top and are traditionally drawn with circles; these correspond to the three factors. Manifest variables appear in squares at the bottom; these are the observed variables that load on the factors. The strength of loading is shown on the path from each factor to its manifest variables, with positive loading in green and negative loading in red (and with a negative sign).

We will see many more examples of path diagrams when we explore confirmatory factor analysis (CFA) and structural equation models in Chap. 10.

Overall, the result of the EFA for this data set is that instead of using 9 distinct variables, we might instead represent the data with 3 underlying latent factors. We have seen that each factor maps to 2–4 of the manifest variables. However, this only tells us about the relationships of the rating variables among themselves in our data; in the next section, we use the estimated factor scores to learn about the *brands*.

8.3.4 Using Factor Scores for Brands

In addition to estimating the factor structure, EFA will also estimate latent factor *scores* for each observation. In the present case, this gives us the best estimates of each respondent's latent ratings for the "value," "leader," and "latest" factors. We can then use the factor scores to determine brands' positions on the factors. Interpreting factors eliminates the separate dimensions associated with the manifest variables, allowing us to concentrate on a smaller, more reliable set of dimensions that map to theoretical constructs instead of individual items.

Factor scores are requested from `factanal()` by adding the `scores=...` argument. We request *Bartlett* scores (see `?factanal`), and extract them from the `factanal()` object using `$scores`, storing them as a separate data frame:

```
> brand.fa.ob <- factanal(brand.sc[, 1:9], factors=3, rotation="oblimin",
+                         scores="Bartlett")
> brand.scores <- data.frame(brand.fa.ob$scores)    # get the factor scores
> brand.scores$brand <- brand.sc$brand              # get the matching brands
> head(brand.scores)
     Factor1    Factor2    Factor3 brand
1  1.6521364 -0.6886749  0.5256104     a
2 -1.4005333 -1.6681901 -0.6764121     a
...
```

The result is an estimated score for each respondent on each factor and brand. If we wish to investigate individual-level correlates of the factors, such as their relationship to demographics or purchase behavior, we could use these estimates of factor scores. This can be very helpful in analyses such as regression and segmentation because it reduces the model complexity (number of dimensions) and uses more reliable estimates (factor scores that reflect several manifest variables). Instead of nine items, we have three factors.

To find the overall position for a brand, we `aggregate()` the individual scores by brand as usual:

```
> brand.fa.mean <- aggregate(. ~ brand, data=brand.scores, mean)
```

We clean this up by assigning names for the rows (brands) and columns (factors):

```
> rownames(brand.fa.mean) <- brand.fa.mean[, 1]           # brand names
> brand.fa.mean <- brand.fa.mean[, -1]
> names(brand.fa.mean) <- c("Leader", "Value", "Latest")  # factor names
> brand.fa.mean
       Leader       Value      Latest
a  0.23158792 -1.06993703  0.39326652
b  0.09686823  1.51913070  0.72391174
...
```

Finally, a heatmap graphs the scores by brand:

```
> heatmap.2(as.matrix(brand.fa.mean),
+           col=brewer.pal(9, "GnBu"), trace="none", key=FALSE, dend="none",
+           cexCol=1.2, main="\n\n\n\n\n\nMean factor score by brand")
```

The result is Fig. 8.10. When we compare this to the chart of brand by adjective in Fig. 8.2, we see that the chart of factor scores is significantly simpler than the full adjective matrix. The brand similarities are evident again in the factor scores, for instance that *f* and *g* are similar, as are *b* and *c*, and so forth.

We conclude that EFA is a valuable way to examine the underlying structure and relationship of variables. When items are related to underlying constructs, EFA reduces data complexity by aggregating variables to create simpler, more interpretable latent variables.

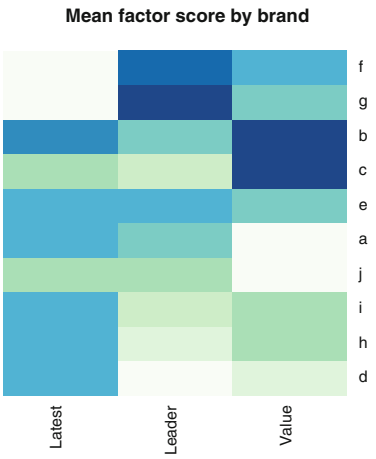

Fig. 8.10. A heatmap of the latent factor scores for consumer brand ratings, by brand.

In this exposition, we have only explored a small number of the possibilities for factor analysis; to learn more, see Sect. 8.5. You will also want to review Chap. 10, which considers the closely related topic of CFA. CFA does not attempt to *find* a factor structure as EFA does, but rather *assesses* how well a proposed structure fits one's data.

8.4 Multidimensional Scaling

MDS is a family of procedures that can also be used to find lower-dimensional representations of data. Instead of extracting underlying components or latent factors, MDS works instead with *distances* (also known as *similarities*). MDS attempts to find a lower-dimensional map that best preserves all the observed similarities between items.

If you have similarity data already, such as ratings of whether one product is like another, you can apply MDS directly to the data. If you have other kinds of data, such as the brand rating data we've considered in this chapter, then you must compute the distances between points before applying MDS. If you have *metric* data—where you consider the units of measurement to have interval or ratio properties—then you might simply calculate euclidian distances with the default `dist()` command, as we do for the mean ratings computed above:

```
> brand.dist <- dist(brand.mean)
```

A procedure to find an MDS solution for a distance matrix from metric data is `cmdscale()`:

```
> (brand.mds <- cmdscale(brand.dist))
           [,1]        [,2]
```

```
a -7.570113e-01  1.4619032
b  5.586301e-01 -2.1698618
...
```

The result of `cmdscale()` is a list of X and Y dimensions indicating two-dimensional estimated plot coordinates for entities (in this case, brands). We see the plot locations for brands *a* and *b* in the output above. Given those coordinates, we can simply `plot()` the values and label them:

```
> plot(brand.mds, type="n")
> text(brand.mds, rownames(brand.mds), cex=2)
```

In this code, `plot(..., type="n")` tells R not to plot symbols. Instead, we add the brand labels to the plot with `text(x, labels)`. The result is Fig. 8.11. The brand positions are grouped nearly identically to what we saw in the perceptual map in Fig. 8.7.

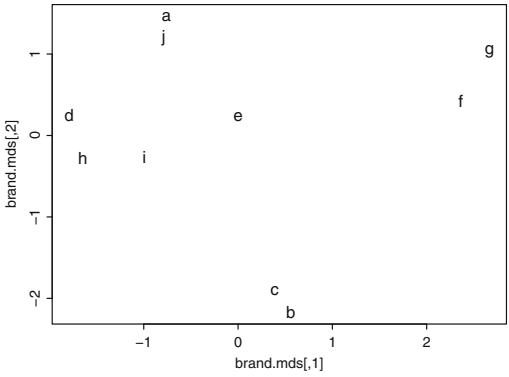

Fig. 8.11. A metric MDS chart for mean brand rating, using `cmdscale()`. The brand positions are quite similar to those seen in the `biplot()` in Fig. 8.7.

8.4.1 Non-metric MDS

For *non-metric* data such as rankings or categorical variables, you would use a different method to compute distance and an MDS algorithm that does not assume metric distances.

For purposes of illustration, let's convert the mean ratings to rankings instead of raw values; this will be non-metric, ordinal data. We apply `rank()` to the columns using `lapply()` and code each resulting column as an ordinal factor variable using `ordered()`:

```
> brand.rank <- data.frame(lapply(brand.mean, function(x) ordered(rank(x))))
> str(brand.rank)
'data.frame': 10 obs. of  9 variables:
```

```
 $ perform: Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 10 8 2 4 5 9 6 7 3
 $ leader : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 3 9 10 2 7 8 6 5 4 1
 ...
```

To find distances between the ranks, we use an alternative to `dist()`, `daisy()` from the `cluster` package (see Sect. 11.3.2), which can handle non-metric data such as rank ordering. In `daisy()`, we compute distance with the `gower` metric, which handles mixed numeric, ordinal, and nominal data:

```
> library(cluster)
> brand.dist.r <- daisy(brand.rank, metric="gower")
```

Now that we have a distance matrix we apply the non-metric MDS function `isoMDS()` to scale the data. Then we plot the result:

```
> brand.mds.r <- isoMDS(brand.dist.r)
initial  value 9.063777
...
converged
> plot(brand.mds.r$points, type="n")
> text(brand.mds.r$points, levels(brand.sc$brand), cex=2)
```

The `plot()` and `text()` commands are slightly different than those we saw above for `cmdscale()`, because `isoMDS()` returns coordinates in the `$points` matrix within its object.

The resulting chart is shown in Fig. 8.12. Compared to Fig. 8.11, we see that brand positions in the non-metric solution are more diffuse. The X axis is arbitrarily reversed, which is not important. Still, the nearest neighbors of brands are largely consistent with the exception of brands *h* and *i*, which are separated quite a bit more than in the metric solution. (This occurs because the rank-order procedure loses some of the information that is present in the original metric data solution, resulting in a slightly different map.)

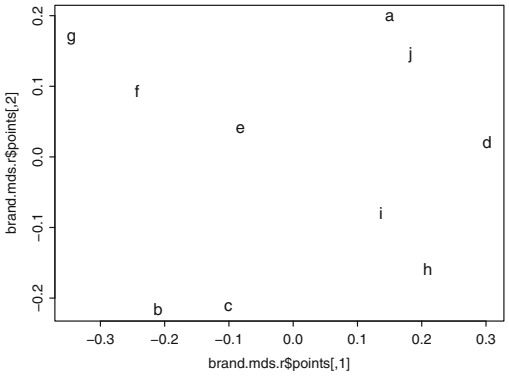

Fig. 8.12. A non-metric MDS chart for mean brand ratings expressed as ordinal ranks, obtained using `daisy()` to find distances and `isoMDS()` for non-metric scaling. The brand groupings are similar to but more diffuse than those in Fig. 8.11.

We generally recommend PCA as a more informative procedure than MDS for typical metric or near-metric (e.g., survey Likert scale) data. However, PCA will not work with non-metric data. In those cases, MDS is a valuable alternative.

MDS may be of particular interest when handling text data such as consumers' feedback, comments, and online product reviews, where text frequencies can be converted to distance scores. For example, if you are interested in similarities between brands in online reviews, you could count how many times various pairs of brands occur together in consumers' postings. The co-occurrence matrix of counts—brand A mentioned with brand B, with brand C, and so forth—could be used as a measure of similarity between the two brands and serve as the distance metric in MDS (see [120]).

8.5 Learning More*

8.5.1 Principal Component Analysis

There is a large literature describing many procedures, options, and applications for each of the analyses in this chapter. With perceptual mapping, a valuable resource is Gower et al. [64] which describes common problems and best practices for perceptual maps. Jolliffe [87] provides a comprehensive text on the mathematics and applications of PCA.

8.5.2 Factor Analysis

The literature on factor analysis is particularly voluminous although it often references statistics packages other than R. A good conceptual overview of EFA with procedural notes (but not R specific) is Fabrigar and Wegener [45], *Exploratory Factor Analysis*. A modestly more technical volume that covers exploratory and confirmatory models together, with a social science (psychology) point of view, is Thompson [151], *Exploratory and Confirmatory Factor Analysis*. For examination of the mathematical bases and procedures of factor analysis, a standard text is Mulaik [119], *Foundations of Factor Analysis*.

The `psych` package [132] presents many additional tools and methods for factor analysis, especially in the context of traditional psychometric instruments such as surveys in general and tests of aptitude or personality. The `fa()` function in `psych` offers an alternative to the standard `factanal()` procedure with more options and more complete assessment of EFA models.

A companion to *exploratory* factor analysis is *confirmatory* factor analysis, which we discuss in Chap. 10. Whereas EFA infers factor structure from a data set, CFA

tests a proposed model to see whether it corresponds well to observed data. A common use of EFA is to select items that load highly on underlying dimensions of interest. CFA allows you to confirm that the relationships between items and factors are maintained in new data sets.

8.5.3 Multidimensional Scaling

There are many uses and options for MDS beyond those considered in this chapter. A readable introduction to the methods and applications is Borg et al. [15], *Applied Multidimensional Scaling*. The statistical foundations and methods are detailed in Borg and Groenen [14], *Modern Multidimensional Scaling*.

8.6 Key Points

Investigation of data complexity has several benefits. It allows inspection of the underlying dimensional relationships among variables, investigation of how observations such as brands or people vary on those dimensions, and estimation of a smaller number of more reliable dimensional scores. The following key points will assist you to investigate the underlying dimensions of your data.

8.6.1 Principal Component Analysis

- PCA finds *linear functions* that explain maximal variance in observed data. A key concept is that such components are *orthogonal* (uncorrelated). The basic R command is `prcomp()` (Sect. 8.2.1).
- A common use for PCA is a *biplot* of aggregate scores for brands or people to visualize relationships. When this is done for attitudinal data such as brand ratings it is called a *perceptual map*. This is created by aggregating the statistic of interest by entity and charting with `biplot()` (Sect. 8.2.2).
- Because PCA components often load on many variables, the results must be inspected cautiously and in terms of relative position. It is particularly difficult to read the status of individual items from a PCA biplot (Sect. 8.2.5).

8.6.2 Exploratory Factor Analysis

- EFA models *latent variables* (factors) that are not observed directly but appear indirectly as observed *manifest variables*. A key procedure is `factanal()` (Sect. 8.3.1).

- A fundamental decision in EFA is the *number of factors* to extract. Common criteria involve inspection of a *scree* plot and extraction of factors such that all *eigenvalues* are greater than 1.0. There are useful tools to determine the number of factors in `nFactors`, but the final determination depends on one's theory and the utility of results (Sect. 8.3.2).
- EFA uses *rotation* to adjust an initial solution to one that is mathematically equivalent but more interpretable according to one's aims. Another key decision in EFA is whether one believes the underlying latent variables should be uncorrelated (calling for an *orthogonal* rotation such as `varimax`) or correlated (calling for an *oblique* rotation such as `oblimin`) (Sect. 8.3.3).
- After performing EFA, you can extract *factor scores* that are the best estimates for each observation (respondent) on each factor. These are present as `$scores` in `factanal()` objects if you request them with the `scores` argument (Sect. 8.3.4).

8.6.3 Multidimensional Scaling

- MDS is similar to PCA but is able to work with both *metric* and *non-metric* data. MDS requires a distance score obtained from `dist()` for metric data or a procedure such as `daisy()` for non-metric data. MDS scaling is then performed by `cmdscale()` for metric data or `isoMDS()` (or other options) for non-metric data (Sect. 8.4).