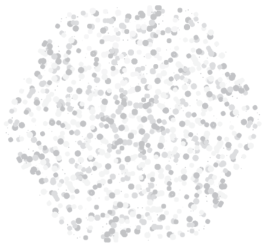
8



LINEAR PROJECTIONS

Our weakness forbids our considering the entire universe and makes us cut it up into slices.

—Poincaré (1913, p 1386)

This chapter covers models that make sense of data with more dimensions than we humans can visualize. The first approach, taken in Section 8.1 and known as principal component analysis (PCA), is to find a two- or three-dimensional subspace that best describes the fifty-dimensional data, and flatten the data down to those few dimensions.

The second approach, in Section 8.2, provides still more structure. The model labels one variable as the dependent variable, and claims that it is a linear combination of the other, independent, variables. This is the ordinary least squares (OLS) model, which has endless variants. The remainder of the chapter looks at how OLS can be applied by itself and in combination with the distributions in the prior chapter.

One way to characterize the two projection approaches is that both aim to project N-dimensional data onto the best subspace of significantly fewer than N dimensions, but they have different definitions of *best*. The standard OLS regression consists of finding the one-dimensional line that minimizes the sum of squared distances between the data and that line, while PCA consists of finding the few dimensions where the variance of the projection of the data onto those dimensions is maximized.

8.1 ※ PRINCIPAL COMPONENT ANALYSIS

PCA is closely related to factor analysis, and in some fields is known as spectral decomposition. The first phase (calculating the eigenvalues) is sometimes called the *singular value decomposition*. It is a purely descriptive method. The idea is that we want a few dimensions that will capture the most variance possible—usually two, because we can plot two dimensions on paper.

After plotting the data, perhaps with markers for certain observations, we may find intuitive descriptions for the dimensions on which we had just plotted the data. For example, Poole & Rosenthal (1985) projected the votes cast by all Congressmen in all US Congresses, and found that 90% of the variance in vote patterns could be explained by two dimensions.[1] One of these dimensions could be broadly described as 'fiscal issues' and the other as 'social issues.' This method stands out because Poole & Rosenthal did not have to look at bills and place them on either scale—the data placed itself, and they just had to name the scales.

Shepard & Cooper (1992) asked their sighted subjects questions regarding color words (red, orange, . . .), and did a principal component analysis on the data to place the words in two dimensions, where they formed a familiar color wheel. They did the same with blind subjects, and found that the projection collapsed to a single dimension, with violet, purple, and blue on one end of the scale, and yellow and gold on the other. Thus, the data indicates that the blind subjects think of colors on a univariate scale ranging from *dark colors* to *bright colors*.

It can be shown that the best n axes, in the sense above, are the n eigenvectors of the data's covariance matrix with the n largest associated eigenvalues.

The programs discussed below query three variables from the US Census data: the population, median age, and males per 100 females for each US state and commonwealth, the District of Columbia and Puerto Rico. They do a factor analysis and then project the original data onto the space that maximizes variance, producing the plot in Figure 8.1.

The programs also display the eigenvectors on the screen. They find that the first eigenvector is approximately $(0.06, -1, 0.03)$. among the three variables given, the second term—population—by itself describes the most variance in the data. The X-axis in the plot follows population[2]

The eigenvalues for the Y-axis are $(0.96, 0.05, -0.29)$, and are thus a less one-sided combination of the first variable (males per female) and the last (median age). That said, how can we interpret the Y axis? Those familiar with US geography will observe that the states primarily covered by cities (at the extreme, DC) are high on

[1]They actually did the analysis using an intriguing maximum likelihood method, rather than the eigenvector method here. Nonetheless, the end result and its interpretation are the same.

[2]As of the 2000 census, California≈ 33.8 million, Texas≈ 20.8, New York≈ 19.0, Florida≈ 16.0, et cetera.

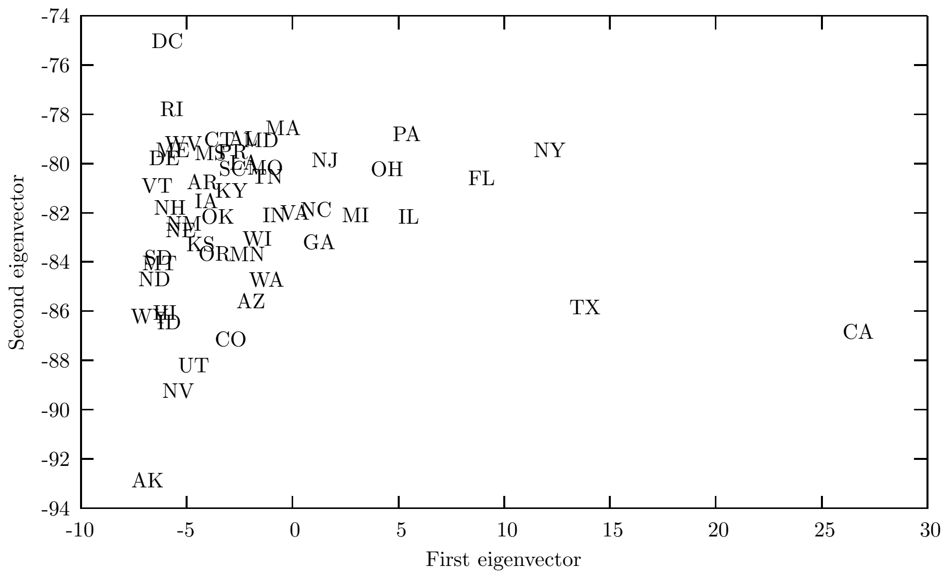


Figure 8.1 States decomposed into population on the X axis, and a combination of median age and gender balance (urban-ness?) on the Y axis.

the Y-axis, while states that are far more rural than urban (at the extreme, Alaska) are toward the bottom. Thus, the principal component analysis indicates that we could plausibly interpret the variation in median age and male/female balance by a single variable representing a state's urban–rural balance. Because interpreting the meaning of an artificial axis is a subjective exercise, other stories are also possible; for example, one could also argue that this second axis instead represents a state's East-to-West location.

※ **Coding it** As with many algorithms, the process of coding is straightforward, but involves a number of details. This section will show you the computation of a principal component analysis on two levels. The first goes through the steps of calculating eigenvectors yourself; the second is a single function call.

- The input and output functions are identical for both programs, so the redundancy-minimizing method of implementing these functions is via a separate file, Listing 8.2, and a corresponding header, which is too simple to be repeated here but is included in the online code supplement as `eigenbox.h`.
- The `query_data` function is self-explanatory. With the clause `select geo_names as row_names`, Apophenia will use the state names as row names rather than plain data.
- These programs are intended to be run via pipe to Gnuplot, like `eigeneasy | gnuplot -persist`. So if we want additional information that Gnuplot will not

```
 1  #include "eigenbox.h"
 2
 3  apop_data *query_data(){
 4      apop_db_open("data-census.db");
 5      return apop_query_to_data(" select postcode as row_names, \n\
 6                              m_per_100_f, population/1e6 as population, median_age \n\
 7                              from geography, income,demos,postcodes \n\
 8                              where income.sumlevel= '040' \n\
 9                              and geography.geo_id = demos.geo_id \n\
10                              and income.geo_name = postcodes.state \n\
11                              and geography.geo_id = income.geo_id ");
12  }
13
14  void show_projection(gsl_matrix *pc_space, apop_data *data){
15      apop_opts.output_type = 'p';
16      apop_opts.output_pipe = stderr;
17      fprintf(stderr,"The eigenvectors:\n");
18      apop_matrix_print(pc_space, NULL);
19      apop_data *projected = apop_dot(data, apop_matrix_to_data(pc_space), 0, 0);
20      printf("plot '-' using 2:3:1 with labels\n");
21      apop_data_show(projected);
22  }
```

Listing 8.2 The tools used below, including the query and a method for displaying labeled points. Online source: `eigenbox.c`.

understand, we need to send it to a different location. Thus, lines 15–18 send output to `stderr`, so the eigenvectors are printed to screen instead of sent down the `stdout` pipeline to Gnuplot.

- The `plot` command on line 20 is `with labels`, meaning that instead of points, we get the two-letter postal codes, as seen in Figure 8.1.
- As for the actual math, Listing 8.3 shows every step in the process.[3] The only hard part is finding the eigenvalues of $\underline{\mathbf{X}}'\underline{\mathbf{X}}$; the GSL saw us coming, and gives us the `gsl_eigen_symm` functions to calculate the eigenvectors of a symmetric matrix. The `find_eigens` function shows how one would use that function. The GSL is too polite to allocate large vectors behind our backs, so it asks that we pass in pre-allocated workspaces when it needs such things. Thus, the `findeigens` function allocates the workspace, calls the eigenfunction, then frees the workspace. It frees the matrix whose eigenvalues are being calculated at the end because the matrix is destroyed in the calculations, and should not be referred to again. To make sure future tests in the way of `if (!subject)...` work, the last line sets the pointer to `NULL`.

[3]There is one cheat: the `apop_sv_decomposition` function would use the `apop_normalize_for_svd(xpx->matrix)` function to ensure that for each row, $\mathbf{x}'\mathbf{x} = 1$. You can erase line 28 of `eigenhard.c`, input this SVD-specific normalization function after the `apop_dot` step, and look for subtle shifts.

```
 1  #include "eigenbox.h"
 2
 3  void find_eigens(gsl_matrix **subject, gsl_vector *eigenvals, gsl_matrix *eigenvecs){
 4      gsl_eigen_symmv_workspace * w = gsl_eigen_symmv_alloc((*subject)->size1);
 5      gsl_eigen_symmv(*subject, eigenvals, eigenvecs, w);
 6      gsl_eigen_symmv_free (w);
 7      gsl_matrix_free(*subject); *subject = NULL;
 8  }
 9
10  gsl_matrix *pull_best_dims(int ds, int dims, gsl_vector *evals, gsl_matrix *evecs){
11    size_t indexes[dims], i;
12    gsl_matrix *pc_space = gsl_matrix_alloc(ds,dims);
13      gsl_sort_vector_largest_index(indexes, dims, evals);
14      for (i=0; i<dims; i++){
15          APOP_MATRIX_COL(evecs, indexes[i], temp_vector);
16          gsl_matrix_set_col(pc_space, i, temp_vector);
17      }
18      return pc_space;
19  }
20
21  int main(){
22    int dims = 2;
23    apop_data *x = query_data();
24    apop_data *cp = apop_data_copy(x);
25    int ds = x->matrix->size2;
26    gsl_vector *eigenvals = gsl_vector_alloc(ds);
27    gsl_matrix *eigenvecs = gsl_matrix_alloc(ds, ds);
28      apop_matrix_normalize(x->matrix, 'c', 'm');
29      apop_data *xpx = apop_dot(x, x, 1, 0);
30      find_eigens(&(xpx->matrix), eigenvals, eigenvecs);
31      gsl_matrix *pc_space = pull_best_dims(ds, dims, eigenvals, eigenvecs);
32      show_projection(pc_space, cp);
33  }
```

Listing 8.3 The detailed version. Online source: `eigenhard.c`.

- If the space of the data has full rank, then there will be three eigenvectors for three-dimensional data. The `pull_best_dimensions` function allocates a new matrix that will have only `dims` dimensions, and the best eigenvectors are included therein. Again, the GSL saw us coming, and provides the `gsl_sort_vector_largest_index` function, which returns an array of the indices of the largest elements of the `evals` vector. Thus, `indexes[0]` is the index of the largest eigenvalue, `indexes[1]` is the point in the vector of values with the second largest eigenvector, et cetera. Given this information, it is an easy `for` loop (lines 14–17) to copy columns from the set of all eigenvectors to the `pc_space` matrix.
- Finally, after the principal component vectors have been calculated, `show_projection` produces a graphical representation of the result. It projects the data onto the

space via the simple dot product `data·pc_space`, and then produces Gnuplottable output as per the tricks discussed above.

- Given all these functions, the main routine is just declarations and function calls to implement the above procedure: pull the data, calculate $\underline{\mathbf{X}}'\underline{\mathbf{X}}$, send the result to the eigencalculator, project the data, and show the results.

```
#include "eigenbox.h"

int main(){
  int dims = 2;
  apop_data *x = query_data();
  apop_data *cp = apop_data_copy(x);
  apop_data *pc_space = apop_matrix_pca(x−>matrix, dims);
    fprintf(stderr, "total explained: %Lf\n", apop_sum(pc_space−>vector));
    show_projection(pc_space−>matrix, cp);
}
```

Listing 8.4 The easy way: just let `apop_sv_decomposition` do the work. Online source: `eigeneasy.c`.

Listing 8.4 presents the same program using the `apop_matrix_pca` function to do the singular value decomposition in one quick function call. That function simply does the normalization and bookkeeping necessary to use the `gsl_linalg_SV_-decomp` function. All that is left for the `main` function in Listing 8.4 is to query the input data to a matrix, call the SVD function, and display the output results.

A matrix is *positive definite* (PD) iff all of its eigenvalues are greater than zero, and *positive semidefinite* (PSD) iff all of its eigenvalues are greater than or equal to zero. Similarly for *negative definite* (ND) and *negative semidefinite* (NSD).

$\mathbb{Q}_{8.1}$ These are often used as multidimensional analogues to positive or negative. For example, just as $x^2 \geq 0, \forall x \in \mathbb{R}$, $\mathbf{X} \cdot \mathbf{X}$ is PSD for any $\mathbf{X}$ with real elements. An extremum of $f(x)$ is a maximum when the second derivative $f''(x)$ is negative, and a minimum when $f''(x)$ is positive; $f(\mathbf{x})$ is a maximum when the matrix of second partial derivatives is NSD, and a minimum when the matrix of second partials is PSD (otherwise it's a saddle point representing a maximum along one direction and a minimum along another, and is thus a false extremum). ⋙

Q8.1 ⋙
Write a function `matrix_is_definite` that takes in a matrix and outputs a single character, say `'P'`, `'p'`, `'N'`, or `'n'`, to indicate whether the matrix is one of the above types (and another character, say `'x'`, if it is none of them). Write a test function that takes in any data matrix $\mathbf{X}$ (your favorite data set or a randomly-generated matrix) and checks that $\mathbf{X} \cdot \mathbf{X}$ is PSD.

$\sum$

- ➤ Principal component analysis projects data of several dimensions onto the dimensions that display the most variance.
- ➤ Given the data matrix $\mathbf{X}$, the process involves finding the eigenvalues of the matrix $\underline{\mathbf{X}}'\underline{\mathbf{X}}$ associated with the largest eigenvalues, and then projecting the data onto the space defined by those eigenvectors.
- ➤ `apop_matrix_pca` runs the entire process for the efficiently lazy user.

8.2 OLS AND FRIENDS

Assume that our variable of interest, $\mathbf{y}$, is described by a linear combination of the explanatory variables, the columns of $\mathbf{X}$, plus maybe a Normally-distributed error term, $\boldsymbol{\epsilon}$. In short, $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where $\boldsymbol{\beta}$ is the vector of parameters to be estimated. This is known as the ordinary least squares (*OLS*) model, for reasons discussed in the introduction to this chapter

To a great extent, the OLS model is the null prior of models: it is the default that researchers use when they have little information about how variables interrelate. Like a good null prior, it is simple, it is easy for the computer to work with, it flexibly adapts to minor digressions from the norm, it handles nonlinear subelements (despite often being called linear regression), it is exceptionally well-studied and understood, and it is sometimes the case that $\mathbf{y}$ really is a linear combination of the columns of $\mathbf{X}$.

OLS is frequently used for solving the snowflake problem from the last chapter: we had a series of very clean univariate models that stated that the outcome of interest is the result of a series of identical draws with equal probability, but in most real-world situations, each draw is slightly different—in the terminology of OLS, we need to control for the other characteristics of each observation. The term *control for* is an analogy to controlled experiments where nuisance variables are fixed, but the metaphor is not quite appropriate, because adding a column to $\mathbf{X}$ representing the control leads to a new projection onto a new space (see below), and may completely change the estimate of the original OLS parameters.[4] Later sections will present other options for surmounting the snowflake problem when using the distributions from the last chapter.

Because linear models are so well-studied and documented, this book will only briefly skim over them.[5] This chapter puts OLS in the context from the first page of this book: a model that claims a relationship between $\mathbf{X}$ and $\mathbf{y}$ and whose parameters can be estimated with data, whose computational tools provide several conveniences. The next chapter briefly covers OLS for hypothesis testing.

Unlike the models to this point, OLS implicitly makes a causal claim: the variables listed in $\mathbf{X}$ cause $\mathbf{y}$ to take the values they do. However, there is no true concept of causality in statistics. The question of when statistical evidence of causality is valid is a tricky one that will be left to the volumes that cover this question in detail.[6] For the purposes here, the reader should merely note the shift in descriptive goal, from fitting distributions to telling a causal story.

A brief derivation Part of OLS's charm is that it is easy (and instructive) to derive $\hat{\boldsymbol{\beta}}$ for the OLS case. We seek the parameter estimate $\hat{\boldsymbol{\beta}}$ that minimizes the squared error, $\boldsymbol{\epsilon}'\boldsymbol{\epsilon}$, where $\mathbf{y} = \mathbf{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$.

This is smallest when the error term $\boldsymbol{\epsilon}$ is orthogonal to the space of $\mathbf{X}$, meaning that $\mathbf{X}'\boldsymbol{\epsilon} = \mathbf{0}$. If $\mathbf{X}$ and $\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$ were not orthogonal, then we could always reduce the size of $\boldsymbol{\epsilon}$ by twiddling $\hat{\boldsymbol{\beta}}$ by an iota to either $\hat{\boldsymbol{\beta}} + \boldsymbol{\iota}$ or $\hat{\boldsymbol{\beta}} - \boldsymbol{\iota}$.

※ Proof: After adding $\boldsymbol{\iota}$ to $\hat{\boldsymbol{\beta}}$, the new equation would be $\mathbf{y} = \mathbf{X}\hat{\boldsymbol{\beta}} + \mathbf{X}\boldsymbol{\iota} + (\boldsymbol{\epsilon} - \mathbf{X}\boldsymbol{\iota})$, meaning that the new error term is now $\boldsymbol{\epsilon}_n \equiv \boldsymbol{\epsilon} - \mathbf{X}\boldsymbol{\iota}$, and

$$\begin{aligned}\boldsymbol{\epsilon}_n'\boldsymbol{\epsilon}_n &= (\boldsymbol{\epsilon} - \mathbf{X}\boldsymbol{\iota})'(\boldsymbol{\epsilon} - \mathbf{X}\boldsymbol{\iota})\\ &= \boldsymbol{\epsilon}'\boldsymbol{\epsilon} - 2\boldsymbol{\iota}'\mathbf{X}'\boldsymbol{\epsilon} + \boldsymbol{\iota}'\mathbf{X}'\mathbf{X}\iota.\end{aligned}$$

The last term, $\iota'\mathbf{X}'\mathbf{X}\iota$, is always a non-negative scalar, just as $x \cdot x$ is non-negative for any real value of x. So the only way that $\boldsymbol{\epsilon}_n'\boldsymbol{\epsilon}_n$ can be smaller than $\boldsymbol{\epsilon}'\boldsymbol{\epsilon}$ is if $2\boldsymbol{\iota}'\mathbf{X}'\boldsymbol{\epsilon} > 0$. But if $\mathbf{X}'\boldsymbol{\epsilon} = \mathbf{0}$, then this is impossible.

The last step of the proof is to show that if $\boldsymbol{\iota}'\mathbf{X}'\boldsymbol{\epsilon}$ is not zero, then there is always a way to pick $\boldsymbol{\iota}$ such that $\boldsymbol{\epsilon}_n'\boldsymbol{\epsilon}_n < \boldsymbol{\epsilon}'\boldsymbol{\epsilon}$. $\mathbb{Q}$: Prove this. (*Hint*: if $\boldsymbol{\iota}'\mathbf{X}'\boldsymbol{\epsilon} \neq 0$, then the

[4]That is, regression results can be unstable, so never trust a single regression.

[5]Economists are especially dependent on linear regression, because it is difficult to do controlled studies on the open economy. Thus, econometrics texts such as Maddala (1977), Kmenta (1986), or Greene (1990) are a good place to start when exploring linear models.

[6]See Perl (2000) or any book on structural equation modeling. The typical full causal model is a directed acyclic graph representing the causal relationships among nodes, and the data can reject or fail to reject it like any other model.

same is true for $\boldsymbol{\iota}_d = 2\boldsymbol{\iota}$ and $\boldsymbol{\iota}_h = \boldsymbol{\iota}/2$.) ♦

Projection So we seek $\hat{\boldsymbol{\beta}}$ that will lead to an error such that $\mathbf{X}'\boldsymbol{\epsilon} = 0$. Expanding $\boldsymbol{\epsilon}$ to $\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$, we can solve for $\hat{\boldsymbol{\beta}}$:

$$\begin{aligned}\mathbf{X}'[\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}] &= 0\\ \mathbf{X}'\mathbf{y} &= \mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}}\\ (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} &= \hat{\boldsymbol{\beta}}\end{aligned}$$

This is the familiar form from page 3.

Now consider the *projection matrix*, which is defined as

$$\mathbf{X}^P \equiv \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'.[7]$$

It is so named because, as will be demonstrated below, $\mathbf{X}^P\mathbf{v}$ projects the vector $\mathbf{v}$ onto the space of $\mathbf{X}$.

Start with $\mathbf{X}^P\mathbf{X}$. Writing this out, it reduces instantly: $\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X} = \mathbf{X}$. So the projection matrix projects $\mathbf{X}$ onto itself.

The expression $\mathbf{X}^P\boldsymbol{\epsilon}$ also simplifies nicely:

$$\begin{aligned}\mathbf{X}^P\boldsymbol{\epsilon} &= \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'[\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}]\\ &= \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'[\mathbf{y} - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}]\\ &= \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}\\ &= \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}\\ &= \mathbf{0}.\end{aligned}$$

The projection matrix projects $\boldsymbol{\epsilon}$ onto the space of $\mathbf{X}$, but $\boldsymbol{\epsilon}$ and $\mathbf{X}$ are orthogonal, so the projected vector is just $\mathbf{0}$.

What does the projection matrix have to do with OLS? Since $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, $\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = \mathbf{X}^P\mathbf{y}$. Thus, the OLS estimate of $\mathbf{y}$, $\hat{\mathbf{y}} \equiv \mathbf{X}\hat{\boldsymbol{\beta}}$, is the projection of $\mathbf{y}$ onto the space of $\mathbf{X}$: $\hat{\mathbf{y}} = \mathbf{X}^P\mathbf{y}$.

And that, in a nutshell, is what OLS is about: the model claims that $\mathbf{y}$ can be projected onto the space of $\mathbf{X}$, and finds the parameters that achieve that with least squared error.

[7] This is sometimes called the *hat matrix*, because (as will be shown), it links $\mathbf{y}$ and $\hat{\mathbf{y}}$.

A sample projection At this point in the narrative, most statistics textbooks would include a picture of a cloud of points and a plane onto which they are projected. But if you have a Gnuplot setup that lets you move plots, you can take the data and projected points in your hands, and get a feel for how they move.

```
 1  #include "eigenbox.h"
 2  gsl_vector *do_OLS(apop_data *set);
 3
 4  gsl_vector *project(apop_data *d, apop_model *m){
 5      apop_data *d2 = apop_data_copy(d);
 6      APOP_COL(d2, 0, ones);
 7      gsl_vector_set_all(ones, 1);
 8      return apop_dot(d2, m->parameters, 0, 'v')->vector;
 9  }
10
11  int main(){
12      apop_data *d = query_data();
13      apop_model *m = apop_estimate(d, apop_ols);
14      d->vector = project(d, m);
15      //d->vector = do_OLS(d);
16      d->names->rowct = 0;
17      d->names->colct = 0;
18      apop_data_print(d, "projected");
19      FILE *cmd = fopen("command.gnuplot", "w");
20      fprintf(cmd, "set view 20, 90\n\
21              splot 'projected' using 1:3:4 with points, 'projected' using 2:3:4\n");
22  }
```

Listing 8.5 Code to project data onto a plane via OLS. Compile with `eigenbox.c` from earlier. Online source: `projection.c`.

Listing 8.5 queries the same data as was plotted in Figure 8.1, does an OLS projection, and produces a Gnuplot command file to plot both the original and projected data.

- Ignore lines 2 and 15 for now; they will allow us to do the projection manually later on.
- Line 13 does the OLS regression. The `apop_estimate` function estimates the parameters of a model from data. It takes in a data set and a model, and returns an `apop_model` structure with the parameters set.
- The `project` function makes a copy of the data set and replaces the dependent variable with a column of ones, thus producing the sort of data on the right-hand side of an OLS regression. Line seven calculates $\mathbf{X}\boldsymbol{\beta}$.
- When the data set is printed on line 18, the first column will be the $\mathbf{X}\boldsymbol{\beta}$ vector just calculated, the second will be the original $\mathbf{y}$ data, and the third and fourth columns

will be the non-constant variables in $\mathbf{X}$.

- Gnuplot has some awkwardness with regards to replotting data (see page 170). Lines 16–18 write the data to a file (with the row and column names masked out), then lines 19–20 write a two-line command file. You can run it from the command line via `gnuplot command.gnuplot -`, and should then have on your screen a plot that you can move about.

The view set on line 20 is basically a head-on view of the plane onto which the data has been projected, and you can see how the data (probably in green) shifts to a projected value (red). It is a somewhat different picture from the PCA in Listing 170. [$\mathbb{Q}$: Add the `with label` commands to put labels on the plot.] Spinning the graph a bit, to `set view 83, 2`, shows that all of the red points are indeed on a single plane, while in another position, at `set view 90, 90`, you can verify that the points do indeed match on two axes.[8]

THE CATALOG Because OLS is so heavily associated with hypothesis testing, this section will plan ahead and present both the estimates of $\boldsymbol{\beta}$ produced by each model, its expected value, and its variance. This gives us all that we need to test hypotheses regarding elements of our estimate of $\boldsymbol{\beta}$, $\hat{\boldsymbol{\beta}}$.

OLS The model:

- $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$
- n = the number of observations, so $\mathbf{y}$ and $\boldsymbol{\epsilon}$ are $n \times 1$ matrices.
- k = the number of parameters to be estimated. $\mathbf{X}$ is $n \times k$; $\boldsymbol{\beta}$ is k by 1.
- Many results that will appear later assume that the first column of $\boldsymbol{\beta}$ is a column of ones. If it isn't, then you need to replace every non-constant column $\mathbf{x}_i$ below with $\mathbf{x}_i - \overline{\mathbf{x}}_i$, the equivalence of which is left as an exercise for the reader.[9] See below for the actual format to use when constructing your `apop_data` set.

Assumptions:

- $E(\boldsymbol{\epsilon}) = 0$.
- $\text{var}(\epsilon_i) = \sigma^2$, a constant, $\forall\, i$.

[8]If your Gnuplot setup won't let you spin the plot with a pointing device, then you can modify the Gnuplot command file to print three separate static graphs in three files via the `set view` commands here, or you can try a command like `set view 83, 2; replot` at the Gnuplot command prompt.

[9]If you would like to take the route of normalizing each column to have mean zero, try `apop_matrix_normalize(dataset, 'm')`. This will normalize a **1** column to **0**, so after calling the normalize function, you may need to do something like `APOP_COL(your_data, 0, onecol); gsl_vector_set_all(onecol, 1)`.

- $\text{cov}(\epsilon_i, \epsilon_j) = 0, \forall\ i \neq j$. Along with the above assumption, this means that the $n \times n$ covariance matrix for the observations' errors is $\Sigma \equiv \sigma^2\mathbf{I}$.
- The columns of $\mathbf{X}$ are not collinear (i.e., $\det(\mathbf{X}'\mathbf{X}) \neq 0$, so $(\mathbf{X}'\mathbf{X})^{-1}$ exists).[10]
- $n > k$.

Notice that we do not assume that $\boldsymbol{\epsilon}$ has a Normal distribution, but that assumption will be imposed in the next chapter, when we apply t tests to $\hat{\boldsymbol{\beta}}$. When all of that holds, then

$$\boxed{\begin{aligned} \hat{\boldsymbol{\beta}}_{\text{OLS}} &= (\mathbf{X}'\mathbf{X})^{-1}(\mathbf{X}'\mathbf{y}) \\ E(\hat{\boldsymbol{\beta}}_{\text{OLS}}) &= \boldsymbol{\beta} \\ \text{var}(\hat{\boldsymbol{\beta}}_{\text{OLS}}) &= \sigma^2(\mathbf{X}'\mathbf{X})^{-1} \end{aligned}}$$

Almost anything can have a variance, which often creates confusion. A column of data has a variance, the column of errors $\boldsymbol{\epsilon}$ has a variance, the estimate $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ has a covariance matrix, and (if so inclined) you could even estimate the variance of the variance estimate $\hat{\sigma}^2$. The variance listed above is the $K \times K$ covariance matrix for the estimate $\hat{\boldsymbol{\beta}}_{\text{OLS}}$, and will be used for testing hypotheses regarding $\boldsymbol{\beta}$ in later chapters. It is a combination of the other variances: the variance of the error term $\boldsymbol{\epsilon}$ is σ^2, and the various columns of the data set have covariance $\mathbf{X}'\mathbf{X}$, so the variance of $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ is the first times the inverse of the second.

INSTRUMENTAL VARIABLES The proofs above gave us a guarantee that we will calculate a value of $\hat{\boldsymbol{\beta}}$ such that $\mathbf{X}$ will be uncorrelated to $\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \mathbf{x}\hat{\boldsymbol{\beta}}$.

Our hope is to estimate a 'true' model, claiming that $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \epsilon$, where $\mathbf{X}$ is uncorrelated to $\boldsymbol{\epsilon}$, and so on. But if it is the case that a column of $\mathbf{X}$ really is correlated to the error in this model, then there is no way that $\hat{\boldsymbol{\beta}}$ (which guarantees that the estimated error and the columns of $\mathbf{X}$ are not correlated) could match $\boldsymbol{\beta}$ (which is part of a model where error and a column of $\mathbf{X}$ are correlated). This creates major problems.

For example, say that one column of the data, $\mathbf{x}_i$, is measured with error, so we are actually observing $\mathbf{x}_i + \boldsymbol{\epsilon}_i$. This means that the error term in our equation is now

[10]If this assumption is barely met, so $\det(\mathbf{X}'\mathbf{X})$ is not zero but is very small, then the resulting estimates will be unstable, meaning that very small changes in $\mathbf{X}$ would lead to large changes in $(\mathbf{X}'\mathbf{X})^{-1}$ and thus in the parameter estimates. This is known as the problem of *multicollinearity*. The easiest solution is to simply exclude some of the collinear variables from the regression, or do a principal component analysis to find a smaller set of variables and then regress on those. See, e.g., Maddala (1977, pp 190–194) for further suggestions.

the true error joined with an offset to the measurement error, $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_i$. If the true $\mathbf{x}_i$ and the true $\boldsymbol{\epsilon}$ have no correlation, then $\mathbf{x}_i + \boldsymbol{\epsilon}_i$ and $\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_i$ almost certainly do.

As above, the OLS estimate of $\boldsymbol{\beta}$ is $\hat{\boldsymbol{\beta}}_{\text{OLS}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, or taking a step back in the derivation, $(\mathbf{X}'\mathbf{X})\hat{\boldsymbol{\beta}}_{\text{OLS}} = \mathbf{X}'\mathbf{y}$. Also, the true $\boldsymbol{\beta}$ is defined to satisfy $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. With a few simple manipulations, we can find the distance between $\boldsymbol{\beta}$ and $\hat{\boldsymbol{\beta}}$:

$$\begin{aligned}
\mathbf{y} &= \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \\
\mathbf{X}'\mathbf{y} &= \mathbf{X}'\mathbf{X}\boldsymbol{\beta} + \mathbf{X}'\boldsymbol{\epsilon} \\
(\mathbf{X}'\mathbf{X})\hat{\boldsymbol{\beta}}_{\text{OLS}} &= \mathbf{X}'\mathbf{X}\boldsymbol{\beta} + \mathbf{X}'\boldsymbol{\epsilon} \\
(\mathbf{X}'\mathbf{X})(\hat{\boldsymbol{\beta}}_{\text{OLS}} - \boldsymbol{\beta}) &= \mathbf{X}'\boldsymbol{\epsilon} \\
\hat{\boldsymbol{\beta}}_{\text{OLS}} - \boldsymbol{\beta} &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\epsilon} \qquad (8.2.1)
\end{aligned}$$

If $\mathbf{X}'\boldsymbol{\epsilon} = 0$, then the distance between $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ and $\boldsymbol{\beta}$ is zero—$\hat{\boldsymbol{\beta}}_{\text{OLS}}$ is a consistent estimator of $\boldsymbol{\beta}$. But if $\mathbf{X}$ and $\boldsymbol{\epsilon}$ are correlated, then $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ does not correctly estimate $\boldsymbol{\beta}$. Further, unless we have a precise measure of the right-hand side of Equation 8.2.1, then we don't know if our estimate is off by a positive, negative, large or small amount. Still further, every column of $\mathbf{X}$ affects every element of $(\mathbf{X}'\mathbf{X})^{-1}$, so mismeasurement of one column can throw off the estimate of the OLS coefficient for every other column as well.

The solution is to replace the erroneously-measured column of data with an *instrument*, a variable that is not correlated to $\boldsymbol{\epsilon}$ but is correlated to the column of data that it will replace. Let $\mathbf{x}_i$ be the column that is measured with error (or is otherwise correlated to $\boldsymbol{\epsilon}$), let $\mathbf{z}$ be a column of alternate data (the instrument), and let $\mathbf{Z}$ be the original data set $\mathbf{X}$ with the column $\mathbf{x}_i$ replaced by $\mathbf{z}$. If $\text{cov}(\mathbf{z}, \boldsymbol{\epsilon}) = 0$, then the following holds:

$$\boxed{\begin{aligned}
\hat{\boldsymbol{\beta}}_{\text{IV}} &= (\mathbf{Z}'\mathbf{X})^{-1}(\mathbf{Z}'\mathbf{y}) \\
E(\hat{\boldsymbol{\beta}}_{\text{IV}}) &= \boldsymbol{\beta} \\
\text{var}(\hat{\boldsymbol{\beta}}_{\text{IV}}) &= \sigma^2(\mathbf{Z}'\mathbf{X})^{-1}\mathbf{Z}'\mathbf{Z}(\mathbf{X}'\mathbf{Z})^{-1}
\end{aligned}}$$

Whe $\det(\mathbf{Z}'\mathbf{X})$ is small, $(\mathbf{Z}'\mathbf{X})^{-1}$—and thus the variance—is large; this happens when $\text{cov}(\mathbf{z}, \mathbf{x}_i) \to 0$. We want the variance of our estimator to be as small as possible, which brings us to the usual rule for searching for an instrument: find a variable that can be measured without significant error, but which is as well-correlated as possible to the original variable of interest.

GLS Generalized Least Squares generalizes OLS by allowing $\boldsymbol{\epsilon}'\boldsymbol{\epsilon}$ to be a known matrix Σ, with no additional restrictions. Note how neatly plugging $\sigma^2\mathbf{I}$ in to the estimator of $\boldsymbol{\beta}$ and its variance here reduces the equations to the OLS versions above.

$$\boxed{\begin{aligned}\hat{\boldsymbol{\beta}}_{\text{GLS}} &= (\mathbf{X}'\Sigma^{-1}\mathbf{X})^{-1}(\mathbf{X}'\Sigma^{-1}\mathbf{y})\\ E(\hat{\boldsymbol{\beta}}_{\text{GLS}}) &= \boldsymbol{\beta}\\ \text{var}(\hat{\boldsymbol{\beta}}_{\text{GLS}}) &= (\mathbf{X}'\Sigma^{-1}\mathbf{X})^{-1}\end{aligned}}$$

But there is a computational problem with GLS as written here. For a data set of a million elements, Σ is $10^6 \times 10^6 = 10^{12}$ (a trillion) elements, and it is often the case that all but a million of them will be zero. A typical computer can only hold tens of millions of `doubles` in memory at once, so simply writing down such a matrix is difficult—let alone inverting it. Thus, although this form is wonderfully general, we can use it only when there is a special form of Σ that can be exploited to make computation tractable.

WEIGHTED LEAST SQUARES For example, let Σ be a diagonal matrix. That is, errors among different observations are uncorrelated, but the error for each observation itself is different. This is *heteroskedastic* data. The classic example in Econometrics is due to differences in income: we expect that our errors in measurement regarding the consumption habits of somebody earning \$10,000/year will be about a thousandth as large as our measurement errors regarding consumption by somebody earning \$10 million/year.

It can be shown (e.g., Kmenta (1986)) that the optimum for this situation, where σ_i is known for each observation i, is to use the GLS equations above, with Σ set to zero everywhere but the diagonal, where the ith element is $\frac{1}{\sigma_i^2}$.

The GLS equations about $\boldsymbol{\beta}$ now apply directly to produce Weighted Least Squares estimates—and there is a trick that lets us do computation using just a vector of diagonal elements of Σ, instead of the full matrix. Let $\sqrt{\boldsymbol{\sigma}}$ be a vector where the ith element is the square root of the ith diagonal element of Σ. For WLS, the ith element of $\sqrt{\boldsymbol{\sigma}}$ is thus $\frac{1}{\sigma_i}$. Now let $\mathbf{y}_\Sigma$ be a vector whose ith element is the ith element of $\mathbf{y}$ times the ith element of $\sqrt{\boldsymbol{\sigma}}$, and $\mathbf{X}_\Sigma$ be the column-wise product of $\mathbf{X}$ and $\sqrt{\boldsymbol{\sigma}}$. That is,

```
void columnwise_product(gsl_matrix *data, gsl_vector *sqrt_sigma){
    for (size_t i=0; i< data−>size2; i++){
        Apop_matrix_col(data, i, v);
        gsl_vector_mul(v, sqrt_sigma);
    }
}
```

Then you can verify that $\mathbf{X}'_\Sigma\mathbf{X}_\Sigma = \mathbf{X}'\Sigma\mathbf{X}$, $\mathbf{X}'_\Sigma\mathbf{y}_\Sigma = \mathbf{X}'\Sigma\mathbf{y}$, and so

$$\boxed{\begin{aligned}\hat{\boldsymbol\beta}_{\text{WLS}} &= \left(\mathbf{X}'_\Sigma\mathbf{X}_\Sigma\right)^{-1}\left(\mathbf{X}'_\Sigma\mathbf{y}_\Sigma\right)\\ E(\hat{\boldsymbol\beta}_{\text{WLS}}) &= \boldsymbol\beta\\ \text{var}(\hat{\boldsymbol\beta}_{\text{WLS}}) &= \sigma^2\left(\mathbf{X}'_\Sigma\mathbf{X}_\Sigma\right)^{-1}\end{aligned}}$$

Thus, we can solve for the Weighted Least Squares elements without ever writing down the full Σ matrix in all its excess. This is the method used by `apop_wls` (which is what you would use in practice, rather than calculating $\mathbf{X}_\Sigma$ yourself).

FITTING IT If you already have a data matrix in `apop_data *set`, then you can estimate an OLS model in one line:

```
apop_estimate_show(apop_estimate(set, apop_ols));
```

If your data set has a non-`NULL` `weights` vector, then you could replace `apop_ols` in the above with `apop_wls`.

If you would like more control over the details of the estimation routine, see page 145 on the format for changing estimation settings, and the online references for the detailed list of options. The `apop_IV` model requires the settings setting treatment, because that model requires that the settings' `instruments` element is set.

$\mathbb{Q}_{8.2}$ In the exercise on page 232, you found a relationship between males per female (the dependent variable) and both density and male–female wage ratios (the independent variables). Produce an appropriate data set and send it to `apop_estimate(`*`your_data`*`, apop_ols)` to check the coefficients when both independent variables are included.

If the assumptions of OLS do not fit, then you will need to be ready to modify the innards of the OLS estimation to suit, so the remainder of this section goes back to the linear algebra layer of abstraction, to go over the steps one would take to estimate $\boldsymbol{\beta}$.

Listing 8.6 extends the example in Listing 8.5 (page 273) by doing every step in the linear algebra of OLS. Uncomment Line 15 in that code, link it with this (by adding `projectiontwo.o` to the `OBJECTS` line in the makefile), and re-run to produce the new projection, which should be identical to the old.

```
 1  #include <apop.h>
 2
 3  gsl_vector *do_OLS(apop_data *set){
 4      apop_data *d2 = apop_data_copy(set);
 5      APOP_COL(d2, 0, firstcol);
 6      apop_data *y_copy = apop_vector_to_data(apop_vector_copy(firstcol));
 7      gsl_vector_set_all(firstcol, 1);
 8
 9      apop_data *xpx = apop_dot(d2,d2,'t',0);
10      gsl_matrix *xpxinv = apop_matrix_inverse(xpx->matrix); //$(\mathbf{X}'\mathbf{X})^{-1}$
11      apop_data *second_part = apop_dot(apop_matrix_to_data(xpxinv), d2,0,'t');
12
13      apop_data *beta = apop_dot(second_part, y_copy, 0, 0); //$(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$
14      strcpy(beta->names->title, "The OLS parameters");
15      apop_data_show(beta);
16
17      apop_data *projection_matrix = apop_dot(d2, second_part,0,0); //$\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$
18      return apop_dot(projection_matrix, y_copy, 0,0)->vector;
19  }
```

Listing 8.6 The OLS procedure and its use for projection, spelled out in great detail. Online source: `projectiontwo.c`.

- Typically, the $\mathbf{y}$ values are the first column of the data matrix, so the first step is to extract the data into a separate vector. Lines 4–7 do this, and end up with one `apop_data` set named `y_copy` which copies off the first column of the input matrix, and a second set named `d2`, which is a copy of the input matrix with the same first column set to $\mathbf{1}$.
- Lines 9–11 find $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$. If you like, you can use the debugger to check the value of any of the intermediate elements all along the calculation.
- Now that we have $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$, Lines 13–15 do the single additional dot product to find $\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, and display the result.
- Line 17 produces the projection matrix $\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$, and given the projection matrix, another dot product projects $\mathbf{y}$ onto it.

The GSL has a function to solve equations of the type $\mathbf{A}\boldsymbol{\beta} = \mathbf{C}$ using Householder transformations, and this happens to be exactly the form we have here—$(\mathbf{X}'\mathbf{X})\boldsymbol{\beta} = (\mathbf{X}'\mathbf{y})$. Given `apop_data` sets for $\mathbf{X}'\mathbf{X}$, $\mathbf{X}'\mathbf{y}$, and a `gsl_vector` allocated for `beta`, this line would fill `beta` with the solution to the equation:

```
gsl_linalg_HH_solve (xpx−>matrix, xpy−>vector, ∗beta);
```

Q8.3 In practice, it is almost necessary to take the inverse to solve for OLS parameters, because the covariance matrix of $\hat{\beta}$ is $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$. But the Householder method manages to avoid explicitly finding the inverse of $\mathbf{X}'\mathbf{X}$, and may thus be quicker in situations like the Cook's distance code (page 133) where a `for` loop solves for thousands of OLS parameter sets.
Rewrite `projectiontwo.c` to use the Householder transformation to find `beta`. Check that the results using this alternate method match the `beta` found via inversion.

$\sum$

- The Ordinary Least Squares model assumes that the dependent variable is an *affine projection* of the others. Given this and other assumptions, the likelihood-maximizing parameter vector is $\boldsymbol{\beta}_{\text{OLS}} = (\mathbf{X}'\mathbf{X})^{-1}(\mathbf{X}'\mathbf{Y})$.
- If $\Sigma \neq \mathbf{I}$, then $\boldsymbol{\beta}_{\text{GLS}} = (\mathbf{X}'\Sigma\mathbf{X})^{-1}(\mathbf{X}'\Sigma\mathbf{Y})$. Depending on the value of Σ, one can design a number of models.
- In most cases, you can use `apop_estimate`. But if need be, coding these processes is a simple question of stringing together lines of linear algebra operations from Chapter 4.

8.3 DISCRETE VARIABLES

To this point, the regression methods have been assuming that both $\mathbf{y}$ and the elements of $\mathbf{X}$ are all continuous variables $\in \mathbb{R}$. If they are discrete variables $\in \{0, 1, 2, \dots\}$, then we need to make modifications.

There are a number of distinctions to be made. First, the approaches to handling columns of discrete variables in $\mathbf{X}$ are different, and simpler, than approaches to handling discrete values of the outcome variable $\mathbf{y}$. If $\mathbf{y}$ only takes on integer values, or worse, is just zero or one, then there can't possibly be a $\boldsymbol{\beta}$ and a Normally distributed $\boldsymbol{\epsilon}$ that satisfy $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. There are convolutions like stipulating that for observations where $\mathbf{x}\boldsymbol{\beta} < 0$ we force ϵ such that $\mathbf{x}\boldsymbol{\beta} + \epsilon = 0$ and where $\mathbf{x}\boldsymbol{\beta} > 1$ we force $\mathbf{x}\boldsymbol{\beta} + \epsilon = 1$, but then $\boldsymbol{\epsilon}$ is non-Normal, which makes a mess of the hypothesis tests we'll run in later chapters. More generally, such an approach lacks grace, and requires a cascade of little fixes (Greene, 1990, p 637).

How we proceed in either case depends on the type of discrete variable at hand:

- A discrete binary variable takes on exactly two values, such as male/female or case/control, and can be rewritten as simply either zero or one.
- Ordered, discrete data are typically a count, such as the number of children or cars a person owns.
- Most qualitative categorizations lead to multi-valued and unordered data. For example, a Metro station could be on the Red, Blue, Green, Orange, and Yellow line, or a voter could align himself with a Red, Blue, Green, or Other party.

DUMMY VARIABLES For a column of zero–one data in the independent data $\mathbf{X}$, we don't really have to modify OLS at all, but we change the interpretation slightly: taking zero as observations in the baseline and one as the observations in the treatment group, the OLS coefficient on this zero–one *dummy variable* can indicate how effective the treatment is in changing the outcome. Tests for the significance of the dummy variable can be shown to be equivalent to ANOVA tests of a category's significance.

As an extension, if the categories are ordered but discrete, like the number of children, then we again don't have to do anything: if a jump from $x = 0$ to $x = 1$ induces a shift of size β in the outcome, then a jump from $x = 1$ to $x = 2$ would do the same under a linear model, and a jump from $x = 0$ to $x = 2$ would produce a jump of size 2β. If it is natural to presume this, then the model does no violence to reality.[11]

But if the categories are discrete and unordered, then we can't use one variable with the values $\{0, 1, 2, \dots\}$, because the implied linear relationship makes no sense. With 0=Green, 1=Blue, 2=Red, does it make sense that a shift from Green to Blue will have exactly the same effect as a shift from Blue to Red, and that a shift from Green to Red would have exactly twice that effect? In this case, the solution is to have a separate variable for all but one of the choices, and thus to produce $n - 1$ coefficients for n options. The excluded option is the baseline, like the zero option in the binary case. Here, the function `apop_data_to_dummies` saves the day: it takes the ith column from the data set and outputs a table with $n-1$ binary dummy variables; see the example below for usage.

Given multiple categories, you could even produce *interaction terms* to represent membership in multiple categories: e.g., control $\times$ male, control $\times$ female,

[11]If the perfectly linear form is implausible, it may be sensible to transform the input variable, to the square of x or $\sqrt{x}$.

treatment $\times$ male, treatment $\times$ female. The easiest way to do this is to simply create a column with two other columns mashed together: in SQLite, the query `select 'left' || 'right'` produces the string `leftright`; in mySQL, `concat('left', 'right')` does the same. Then break the column down into $n - 1$ dummy variables as above.

The underlying claim with zero–one dummy variables is $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{k}$, where $\mathbf{k}$ indicates a constant value that gets added to the controls but not the cases, for example. But say that we expect the slopes to differ from cases to controls; for cases the equation is $\mathbf{y} = \mathbf{x}_1\beta_1 + \mathbf{x}_2\beta_2$ and for controls it is $\mathbf{y} = \mathbf{x}_1(\beta_1 + k) + \mathbf{x}_2\beta_2$. The way to get a standard regression to produce k in this case would be to produce a data set which has the appropriate value $\mathbf{x}_1$ for each control, but zero for each case. Then the regression would be equivalent to $\mathbf{y} = \mathbf{x}_1\beta_1 + \mathbf{x}_1 k + \mathbf{x}_2\beta_2$ for controls and $\mathbf{y} = \mathbf{x}_1\beta_1 + \mathbf{x}_2\beta_2$ for cases, as desired.

```
#include <apop.h>

apop_model * dummies(int slope_dummies){
    apop_data *d = apop_query_to_mixed_data("mmt", "select riders, year−1977, line \
            from riders, lines \
            where riders.station=lines.station");
    apop_data *dummified = apop_data_to_dummies(d, 0, 't', 0);
    if (slope_dummies){
        Apop_col(d, 1, yeardata);
        for(int i=0; i < dummified−>matrix−>size2; i ++){
            Apop_col(dummified, i, c);
            gsl_vector_mul(c, yeardata);
        }
    }
    apop_data *regressme = apop_data_stack(d, dummified, 'c');
    apop_model *out = apop_estimate(regressme, apop_ols);
    apop_model_show(out);
    return out;
}

#ifndef TESTING
int main(){
    apop_db_open("data−metro.db");
    printf("With constant dummies:\n"); dummies(0);
    printf("With slope dummies:\n"); dummies(1);
}
#endif
```

Listing 8.7 Two regressions with dummy variables. Online source: `dummies.c`.

Listing 8.7 runs a simple regression of ridership at each station in the Washington Metro on a constant, the year, and a set of dummy variables for Green, Orange, Red, and Yellow lines (meaning the Blue line is the baseline).

- The query on line four pulls two numeric columns (ridership and year) and one text column.
- Line 7 converts text column zero into an `apop_data` set consisting of dummy variables. If you ask the debugger to display the `dummified` data set, you will see that every row has a 1 in at most a single column.
- Line 15 stacks the dummies to the right of the other numeric data, at which point the data set is in the right form to be regressed, as on line 16. Again, it is worth asking the debugger to show you the data set in its final, regressable form.
- We will test the claim that these two tests are equally likely on page 354; the `#ifndef` wrapper on lines 21 and 27 will let us read this file into the testing program. Here, the wrapper is innocuous.
- The `slope_dummies` switch determines whether the dummies will be constant dummies or slope dummies. For constant dummies, we need only zeros or ones, so as above, `apop_data_to_dummies` gives us what we need. For slope dummies, we need to replace every 1 with the current year, which is what the column-by-column vector multiplication achieves over lines 9–13.

PROBIT AND LOGIT Now we move on to the question of discrete outcome variables. For example, a person either buys a house or does not, which makes house purchasing a binary variable. Say that the value of a house is a linear sum of the value of its components: an extra bedroom adds so many dollars, a nearby highly-rated school adds so many more, each square meter is worth a few thousand dollars, et cetera. That is, one could write down a linear model that total value $U = \beta_1$·cost + β_2·bedrooms + β_3·school quality + β_4·square meters $+ \cdots + \epsilon$, where β_1 is typically normalized to -1 and ϵ is the usual error term.[12]

To phrase the model briefly: $U = \mathbf{x}\boldsymbol{\beta} + \epsilon$. But rather than leaving the model in the standard linear form, we add the rule that a person buys iff $U > 0$. Thus, the input is the same set of characteristics $\mathbf{x}$ (of consumers or of houses) and weightings $\boldsymbol{\beta}$, and the output is a binary variable: the person acted or did not; the buyer consumed or did not; the geyser erupted or did not, et cetera.

From here, the details of the decision model depend on whether the outcome is binary, multivalued but unordered, or ordered.

- Logit with two choices: The first alternative, the *logit model* or *logistic model*, assumes that $\epsilon \sim$ a *Gumbel distribution*. The Gumbel is also known as the Type I

[12]Real estate economists call this the *hedonic pricing model*, whose implementation typically differs in some respects from the logit model discussed here. See Cropper *et al.* (1993) for a comparison.

Extreme Value distribution, meaning that it expresses the distribution of the statistic $\max(\mathbf{x})$ under appropriate conditions. McFadden (1973) won a Nobel prize partly by showing that the above assumptions, including the rather eccentric error term, reduce to the following simple form:

$$\begin{aligned} P(0|\mathbf{x}) &= \frac{1}{1+\exp(\mathbf{x}\boldsymbol{\beta}_1)} \\ P(1|\mathbf{x}) &= \frac{\exp(\mathbf{x}\boldsymbol{\beta}_1)}{1+\exp(\mathbf{x}\boldsymbol{\beta}_1)} \end{aligned}$$

There is no $\boldsymbol{\beta}_0$ because is it normalized to **0**, so $\exp(\mathbf{x}\boldsymbol{\beta}_0) = 1$.

- Probit: The *probit model* has a similar setup to the logit, but $\epsilon \sim \mathcal{N}(0,1)$.
  The model can be rephrased slightly. Say that $U(\mathbf{x})$ is deterministic, but the probability of consuming given some utility is random. To describe such a setup, let $U = -\mathbf{x}\boldsymbol{\beta}$ rather than $-\mathbf{x}\boldsymbol{\beta} + \epsilon$ (the introduction of a minus sign will be discussed in detail below), and let the probability that the agent acts be

  $$P(\mathbf{x}|\boldsymbol{\beta}) = \int_{-\infty}^{U(-\mathbf{x}\boldsymbol{\beta})} \mathcal{N}(y|0,1)dy,$$

  where $\mathcal{N}(y|0,1)$ indicates the Normal PDF at y with $\mu = 0$ and $\sigma = 1$, so the integral is the CDF up to $U(-\mathbf{x}\boldsymbol{\beta})$.
  As you can see, this is a much more ornery form, especially since the CDF of the Normal (what engineers call the *error function*, or more affectionately, *erf*) does not simplify further and is not particularly easy to calculate. This partly explains the prevalence of the logit form in existing journal articles, but is not much of an issue now that maximum likelihood estimation is cheap, even when erf is involved.
- With k different options, the logit model generalizes neatly for the case of multiple values, to the *multinomial logit*:

  $$\begin{aligned} P(0|B) &= \frac{1}{1+\sum_{i=1}^{k}\exp(\mathbf{x}\boldsymbol{\beta}_i)} \text{ and} \\ P(k|B) &= \frac{\exp(\mathbf{x}\boldsymbol{\beta}_x)}{1+\sum_{i=1}^{k}\exp(\mathbf{x}\boldsymbol{\beta}_i)}, k \neq 0 \end{aligned} \tag{8.3.1}$$

  where B is the set of choices from which x is chosen, and $\boldsymbol{\beta}_k$ is a different vector of coefficients for each option $k \neq 0$. You can see that $\boldsymbol{\beta}_0$ is once again normalized to **0**, and given only options zero and one, the form here reduces to the binomial form above.[13]
- Ordered multinomial probit: Presume that there is an underlying continuous variable $Y^* = -\mathbf{X}\boldsymbol{\beta}$ such that the true $Y_i = 0$ if $Y_i^* \leq 0$; $Y_i = 1$ if $0 < Y_i^* \leq A_1$; $Y_i = 2$ if $A_1 < Y_i^* \leq A_2$; and so on. That is,

[13]The generalization is so clean that Apophenia doesn't even include separate binary and multinomial logit models: the `apop_logit` model just counts the options and acts accordingly.

$$\begin{aligned}
P(Y_i = 0) &= \int_{-\infty}^{A_1 - \mathbf{x}\boldsymbol{\beta}} \mathcal{N}(y|0,1)dy \\
P(Y_i = 1) &= \int_{A_1 - \mathbf{x}\boldsymbol{\beta}}^{A_2 - \mathbf{x}\boldsymbol{\beta}} \mathcal{N}(y|0,1)dy \\
P(Y_i = 2) &= \int_{A_2 - \mathbf{x}\boldsymbol{\beta}}^{\infty} \mathcal{N}(y|0,1)dy
\end{aligned}$$

Rather than finding a single cutoff between acting and not acting, we estimate several cutoffs, between choosing zero, one, two, . . . items; estimating `apop_multinomial_probit` will therefore return the usual vector of parameters β, plus a vector of $A_1, A_2, \ldots, A_{k-1}$ for k options.

Parameter overload Very reasonable models can produce an overwhelming number of parameters. For the multinomial logit model above, with k options and n elements in each observation $\mathbf{x}$, there are $n \times (k-1)$ parameters to be estimated. Your goals will dictate upon which parameters you should focus, and how you get meaning from them.

If the sole goal is to prove that A causes B, then the univariate test that the coefficient on A is significant is all we will look at from one of the above tests.

If we want to predict how comparable people will act in the future, then we need the model's estimate for the odds that each individual will select any one choice, and the predicted most likely selection. Perhaps you will want the full catalog, or just the average probabilities via `apop_data_summarize(`*`outmodel`*`.expected)`.

Or, you may want to know the dynamic shifts: what happens to B when A rises by 1%? Economists call this the *marginal change*, though virtually every field has some interest in such questions. This is closely related to the question of whether the parameter on A is statistically significant: you will see in the chapter on maximum likelihood estimates that their variance (used to measure confidence for a hypothesis tests) is literally the inverse of the second derivative (used to measure change in outcome variable given change in income variable).

For a linear model, the marginal change is trivial to calculate: we proposed that $y = \mathbf{x}\boldsymbol{\beta}$, meaning that $\partial y/\partial x_i = \beta_i$. That is, a 1% change in x_i leads to a $\beta\%$ change in y.

Interpreting the marginal changes in the above models can be more difficult. For the probit, the cutoff between draws that induce a choice of option zero and those that lead to option one was $-\mathbf{x}\boldsymbol{\beta}$. As $+\mathbf{x}\boldsymbol{\beta}$ grows (either because of a marginal expansion in an element of $\mathbf{x}$ or of $\boldsymbol{\beta}$), the cutoff falls slightly, and the set of draws that lead to option one grows. This fits our intiution, because we characterized the elements of the linear sum $\mathbf{x}\boldsymbol{\beta}$ as increasing the proclivity to choose option one.

For the ordered probit, there are multiple cutoffs. A marginal change in $\boldsymbol{\beta}$ will perhaps raise the cutoff between options two and three, lowering the odds of choosing three, but at the same time raise the cutoff between options three and four, raising the odds of choosing three. See Greene (1990, p 672) or your estimation program's documentation for more on making productive use of the estimated model.

IIA Return to the multinomial logit formula, Equation 8.3.1, and consider the ratio of the probability of two events, E_1 and E_2.

$$\begin{aligned}\frac{P(E_1|x,B)}{P(E_2|x,B)} &= \frac{\frac{\exp(\mathbf{x}\boldsymbol{\beta}_1)}{\sum_{y\in B}\exp(\mathbf{x}\boldsymbol{\beta}_y)}}{\frac{\exp(\mathbf{x}\boldsymbol{\beta}_2)}{\sum_{y\in B}\exp(\mathbf{x}\boldsymbol{\beta}_y)}}\\ &= \frac{\exp(\mathbf{x}\boldsymbol{\beta}_1)}{\exp(\mathbf{x}\boldsymbol{\beta}_2)}\end{aligned}$$

The key point in this simple result is that this ratio does not depend on what else is in the set of options. The term for this property is *independence of irrelevant alternatives*, or *IIA*. The term derives from the claim that the choice between one option and another in no way depends upon the other options elsewhere.

The standard example is commuters choosing among a red bus, a blue bus, and a train, where the red bus and blue bus are identical except for their paint job. We expect that the ratio of P(red)/P(blue) = 1, and let P(red)/P(train) = P(blue)/P(train) = $\frac{1}{x}$. If the train were out of service, then P(red)/P(blue) would still be 1, because as many former train riders would now pick the blue bus as would pick the red, but if the blue bus were knocked out of commission, then everybody who was riding it would likely just switch to the red bus, so without the blue bus, P(red)/P(train) = $\frac{2}{x}$. In short, the train option is irrelevant to the choice of red or blue bus, but the blue bus option is not irrelevant in the choice between train and red bus.

In the code supplement, you will find `data-election.db`, which includes a small selection of survey data regarding the United States's 2000 election, from National Election Studies (2000). The survey asked respondents their opinion of various candidates (including the two candidates for Vice President) on a scale from 0 to 100; in the database, you will find the person that the respondent most preferred.[14]

[14] *Duverger's law* tells us that a first-past-the-post system such as that of the United States tends to lead to a stable two-party system, due to the various strategic considerations that go into a vote. We see this in the data: question 793 of the survey (not included in the online supplement) asked the respondent for whom he or she expects to vote, and the totals were: (Gore, 704), (Bush, 604), (everybody else, 116). The favorite from the thermometer score avoids the strategic issues involved in vote choice, and produces a much broader range of preferences: (Gore, 711), (Bush, 524), (Nader, 157), (Cheney, 68), et cetera. $\mathbb{Q}$: Write a query to get the complete count for each candidate from the survey data.

The IIA condition dictates that the logit estimates of the relative odds of choosing candidates won't change as other candidates enter and exit, though intuitively, we would expect some sort of shift. The examples also suggest a two-level model, wherein people first choose a category (bus/train, Republican/Democrat/Green), and then having chosen a category, select a single item (such as red bus/blue bus, Buchanan/McCain). Such a model weakens IIA, because a choice's entry or exit may affect the choice of category. These hierarchies of class and item are a type of *multilevel model*; the next section will give an overview of multilevel modeling possibilities, including some further discussion of the nested logit.

The theoretical IIA property is irrefutable—there is not much room for error in the two lines of algebra above. But many theoretical models can be pushed to produce mathematically clean results that are just not relevant in the application of the model to real data. Does IIA have a strong influence on real-world logit estimates?

We would test IIA by running one unconstrained logit, and another logit estimation restricted so that one option is missing; this fits the likelihood ratio (LR) form which will appear on page 351, and a few variants on this test exist in the literature. Cheng & Long (2007) built a number of artificial data sets, some designed around agents who make choices conforming to IIA, and some designed to not have the IIA property. They found that the LR tests were ineffective: the unconstrained and constrained odds ratios sometimes did not differ (demonstrating IIA) and sometimes did (against IIA), but there was no sufficiently reliable relationship between when the underlying data had the IIA property and when the parameter estimates demonstrated IIA. Fry & Harris (1996) used a less broad method that had slightly more optimistic results, but still encountered glitches, such as problems with power and results that were sensitive to which option was removed.

The probit model matches the logit save for the type of bell curve that describes the error term, so one expects the parameters and predicted odds to be similar. In fact, Amemiya (1981) points out that logit parameters typically match probit parameters, but that they are scaled by a factor of about 1.6. That is, for each option i, $\beta_i^L \approx 1.6\beta_i^P$. Yet, the logit parameters have the IIA property, while the probit parameters do not. This is hard to reconcile, unless we accept that the IIA property of the logit is theoretically absolute but empirically weak.

$\sum$

➤ For discrete explanatory variables, you can use the standard family of OLS models, by adding dummy variables.

➤ The standard linear form $\mathbf{X}\boldsymbol{\beta}$ can be plugged in to a parent function, like the Probit's comparison of $\mathbf{X}\boldsymbol{\beta}$ to a Normal distribution or the logit's comparison to a Gumbel distribution, to generate models of discrete choice. ⋙

⋙

$\sum$

- ➤ The Logit model has the property that the ratio of the odds of selecting two choices, e.g., $p(A)/p(B)$, does not depend on what other choices exist.
- ➤ The computational pleasantries that OLS demonstrates are no longer applicable, and we usually need to do a maximum likelihood search to find the best parameters.

8.4 MULTILEVEL MODELING

Retake the snowflake problem from page 236: the models to this point all assumed that each observation is independently and identically distributed relative to the others, but this is frequently not the case.

One way in which this may break is if observations fall into clusters, such as families, classrooms, or geographical region. A regression that simply includes a family/classroom/region dummy variable asserts that each observation is iid relative to the others, but its outcome rises or falls by a fixed amount depending on its group membership. But the distribution of errors for one family may be very different from that of another family, the unobserved characteristics of one classroom of students is likely to be different from the unobserved characteristics of another, and so on.

A better alternative may be to do a model estimation for each group separately. At the level of the subgroup, unmodeled conditions are more likely to be constant for all group members. Then, once each group has been estimated, the parameters can be used to fit a model where the observational unit is the group.

The other type of multilevel model is that where there are no distinct subclusters, but we would like to give more detail about the derivation of the parameters. For example, say that we feel that the propensity to consume a good is Normally distributed, but we know that the likelihood of consumption is also based on a number of factors. We just saw this model—it is the probit, which asserted that the likelihood of consuming is simply the CDF of a Normal distribution up to a parameter, and that parameter has the form $\mathbf{X}\boldsymbol{\beta}$. The logit asserted that the probability of consuming was $\exp(\theta)/(1+\exp(\theta))$, where θ is of the form $\mathbf{X}\boldsymbol{\beta}$.

These are examples of *multilevel models*. There is a parent model, which is typically the primary interest, but instead of estimating it with data, we estimate it using the results from subsidiary models. To this point, we have broadly seen two types of model: simple distributions, like the Normal(μ, σ) or the Poisson(λ), and models asserting a more extensive causal structure, like the OLS family. Either form could be the parent in a multilevel model, and either form could be the child.

Clustering A classroom model may assume a distribution of outcomes for each classroom and estimate μ_i and σ_i for each class, and then assert a linear form that the outcome variable is $\beta_1\mu_i+\beta_2\sigma_i$. Unemployment models are typically modeled as Poisson processes, so one could estimate λ_i for each region i, but then link those estimates together to assert that the λ's have a Normal distribution.

```
#include <apop.h>

void with_means(){
    apop_data *d2 = apop_query_to_data("select avg(riders), year−1977 \
                from riders, lines \
                where riders.station=lines.station\
                group by lines.line, year");
    apop_model_show(apop_estimate(d2, apop_ols));
}

void by_lines(){
  apop_data *lines = apop_query_to_text("select distinct line from lines");
  int linecount = lines−>textsize[0];
  apop_data *parameters = apop_data_alloc(0, linecount, 2);
    for(int i=0; i < linecount; i ++){
        char *color = lines−>text[i][0];
        apop_data *d = apop_query_to_data("select riders, year−1977 from riders, lines\n\
                where riders.station = lines.station and lines.line = '%s'", color);
        apop_model *m = apop_estimate(d, apop_ols);
        APOP_ROW(parameters, i, r);
        gsl_vector_memcpy(r, m−>parameters−>vector);
    }
    apop_data_show(parameters);
    apop_data *s = apop_data_summarize(parameters);
    apop_data_show(s);
}

int main(){
    apop_db_open("data−metro.db");
    printf("Regression parent, Normal child:\n\n"); with_means();
    printf("Normal parent, regression child:\n\n"); by_lines();
}
```

Listing 8.8 Two model estimations: with a regression parent and with a Normal parent. Online source: `ridership.c`.

Listing 8.8 estimates two models of the change in ridersip of the Washington Metro over time. You already saw two models of this data set, in the form of two dummy-variable regressions, on page 282. Here, we take a multilevel approach: first with a regression parent and distribution child, and then the other way around.

Having SQL on hand pays off immensely when it comes to clustering, because it is so easy to select data where its group label has a certain value, or calculate aggregates using a `group by` clause. It will not be apparent in this example using just under 2,000 data points, but because SQL tables can be indexed, grouping and aggregation can be much faster than sifting through a table line by line.

The first model is a regression parent, distribution child: we estimate a Normal model for each Metro line for each year, and then regress on the set of $\hat{\mu}$'s thus estimated. Of course, finding $\hat{\mu}$ is trivial—it's the mean, which even standard SQL can calculate, so the output from the query on line 4 is already a set of statistics from a set of Normal distribution estimations. Then, the regression estimates the parameters of a regression on those statistics.

The second model is a distribution parent, regression child: we run a separate regression for every Metro line and then find the mean and variance of the OLS parameters. Line 12 queries a list of the five colors (Blue line, Red line, ...), and then lines 15–22 are a `for` loop that runs a separate regression for each color, and writes the results in the `parameters` matrix. Then, line 24 finds $\hat{\mu}$ and $\hat{\sigma}$ for the set of parameters.

The estimations tell different stories, and produce different estimates for the slope of ridership with time. Notably, the Green line added some relatively unused stations in the 1990s, which means that the slope of the Green line's ridership with time is very different from that of the other lines. This is very clear in the second case where we run a separate regression for every line, but is drowned out when the data set includes every line.

Notice also that the size of the parent model's data set changes with different specifications: in the first model, it was 150; in the second, it was 5. Thus, the first model had greater confidence that the slope was different from zero.[15] Gelman & Hill (2007) point out that we test parameters only on the parent model, which means that if n is very small for some of the clusters, then this should have no effect on the parent—even $n = 1$ for some clusters is not a problem. Clusters with small n should show large natural variation in child parameter estimates, and that would be reflected in the quality of the estimation of the parent parameters. But since we neither make nor test claims about the child parameters, there is no need to concern ourselves directly with the 'quality' of the child parameters.

[15] In the second specification, with five data points, one has a negative slope and four a positive slope. Nonetheless, the mean is still about three times $\hat{\sigma}/\sqrt{n}$, giving us about 99% confidence that the mean is significant.

To get a better handle on what differs among the lines and within the overall regression, draw some graphs:

- total ridership per year
- average ridership per year
- total ridership for each line.
- average ridership for each line.

Q8.4

The difference between the total and average ridership is based on the fact that the number of stations is not constant over time—produce a crosstab of the data to see the progression of new station additions.
The plots for each line are best written via a `for` loop based on the one beginning on line fifteen of Listing 8.8. How do the graphs advise the statistics calculated by the two dummy and two multilevel models?

An example: multi-level logit We can break down the process of choosing a presidential candidate into two steps: first, the voter decides whether to choose a Democrat, Republican, or Green candidate, and then chooses among the offered candidates in the chosen party. The probability of selecting a candidate is thus $P(\text{candidate}) = P(\text{candidate}|\text{party})P(\text{party})$. We would thus need to do a few logit estimations: one for the Democratic candidates, one for the Republican candidates, and one for the party choice.[16] The IIA property would continue to hold within one party, but among parties, the change in candidates could shift $P(\text{party})$, so the proportional odds of choosing a given Democrat versus choosing a given Republican may shift. Listing 8.9 does some of these logit estmations along the way to the multilevel model.

- Once again, SQL saves the day in doing these multilevel estimations. In this case, the explanatory variables won't change from model to model, so they are fixed in the `logit` function, but the outcome variable will change. Thus, the function takes in the varying tail of the query and appends it to the fixed list of explanatory variables to pull out consistent data.
- The tails of the queries, in `main`, are straightforward: get all candidates, get all Democrats, get all Republicans, get only party names.
- Out of the overwhelming amount of data, the `logit` function displays on screen only the mean odds of selecting each option.

[16]There was only one Green candidate, Ralph Nader, so $P(\text{Nader}) = P(\text{Green})$.

```
#include <apop.h>

apop_model * logit(char *querytail){
    apop_data *d = apop_query_to_mixed_data("mmmmmt", "select 1, age, gender, \
            illegal_immigrants, aid_to_blacks, %s", querytail);
    apop_text_to_factors(d, 0, 0);
    apop_model *m = apop_estimate(d, apop_logit);
    apop_data *ev = apop_expected_value(d, m);
    apop_data_show(apop_data_summarize(ev));
    return m;
}

int main(){
    apop_db_open("data-nes.db");
    logit("favorite from choices");
    logit("favorite from choices c, party p where p.person = c.favorite and p.party = 'R'");
    logit("favorite from choices c, party p where p.person = c.favorite and p.party = 'D'");
    logit("p.party from choices c, party p where p.person = c.favorite");
}
```

Listing 8.9 Logit models with and without grouping of candidates by party. Online source: `candidates.c`.

$\mathbb{Q}_{8.5}$ The example is not quite complete: we have the probability of choosing a candidate given a party and the probability of choosing a party, but not the product of the two. Fill in the remainder of the code by finding the predicted odds for each candidate for each observation.

You can count how often a person chose the candidate that the model says the person is most likely to pick. Did the multilevel logit do a better job of picking candidates than the standard one-level logit?

The simple concept of nesting together models is akin to McFadden's nested logit (McFadden, 1978). In fact, if these aren't enough possibilities for you, see Gelman & Hill (2007), who offer several more.

Describing snowflakes Now consider the multilevel model as used to model parameters that are used as inputs to other models. This is the typical form for when we want to use one of the stories from Chapter 7, like the one about the Negative Binomial model or the one about the Poisson, but we want the parameter to vary according to per-observation conditions. As above, the probit model from the last section fits this description, as each agent has a cutoff based on a linear model and that agent's characteristics (i.e., the cutoff for person i is $\mathbf{x}_i\boldsymbol{\beta}$), and that cutoff is then fed into a parent Normal distribution.

Because many of Apophenia's model-handling functions can work with a model that has only a log likelihood specified, the process of writing a log likelihood for such a model is supremely simple:

- Find the linear estimates of the parameters for each observation, probably using `apop_dot` on the input matrix and input parameters.
- Use the stock `log_likelihood` function to find the log likelihood of the outcome data given the parameter estimates from the last step.

Listing 8.10 implements such a process. The first step is building the model, which basically consists of reusing existing building blocks. The log likelihood function is simply the two steps above: find $\mathbf{X}\boldsymbol{\beta}$, then run a `for` loop to estimate a separate Poisson model for each row, based on the ith outcome variable and the ith Poisson parameter. The `main` program declares the model, pulls the data, and then runs the data through both the new model and the standard OLS model.

```
#include <apop.h>

double pplc_ll(apop_data *d, apop_model *child_params){
    apop_data *lambdas = apop_dot(d, child_params->parameters, 0);
    apop_data *smallset = apop_data_alloc(0, 1, 1);
    double ll = 0;
    for(size_t i=0; i < d->vector->size; i ++){
        double lambda = apop_data_get(lambdas, i, -1);
        apop_model *pp = apop_model_set_parameters(apop_poisson, lambda);
        apop_data_set(smallset, 0,0, apop_data_get(d, i, -1));
        ll += pp->log_likelihood(smallset, pp);
    }
    return ll;
}

int main(){
    apop_model pplc = {"Poisson parent, linear child", -1,
                        .log_likelihood= pplc_ll, .prep=apop_probit.prep};
    apop_db_open("data-tattoo.db");
    apop_data *d = apop_query_to_data("select \
                tattoos.'ct tattoos ever had' ct, tattoos.'year of birth' yr, \
                tattoos.'number of days drank alcohol in last 30 days' booze \
                from tattoos \
                where yr+0.0 < 97 and ct+0.0 < 10 and booze notnull");
    apop_data *d2 = apop_data_copy(d);
    apop_model_show(apop_estimate(d, pplc));
    apop_model_show(apop_estimate(d2, apop_ols));
}
```

Listing 8.10 A multilevel model. Online source: `probitlevels.c`.

In this case, you can interpret the parameters in a similar manner to the discussion of the probit and logit cases above. The parent parameter λ is calculated as $\mathbf{X}\boldsymbol{\beta}$, so a 1% shift in x_i leads to a $\beta_i\%$ shift in λ. Thus, after checking whether the parameters are significantly different from zero, you can directly compare the relative magnitudes of the parameters to see the relative effect of a 1% shift in the various inputs.

Statistics is a subjective field At this point, you have seen quite a range of models: you saw the distribution models from Chapter 7, the linear models from earlier in this chapter, and here you can see that you can embed any one model into any other to form an infinite number of richer models. Plus, there are the simulation and agent-based models, standing alone or nested with a different type of model. The Metro data has already been modeled at least four different ways (depending on how you count), and the male-to-female ratio among counties in still more ways. Which model to choose?

A model is a subjective description of a situation, and many situations afford multiple perspectives. This is rare advice from a statistics textbook, but *be creative*. We're not living in the 1970s anymore, and we have the tools to tailor the model to our perceptions of the world, instead of forcing our beliefs to fit a computationally simple model. Try as many models as seem reasonable. Having many different perspectives on the same situation only raises the odds that you will come away with a true and correct understanding of the situation.

We also have some more objective tools at our disposal for selecting a model: Chapter 10 will demonstrate a means of testing which of two disparate models has a better fit to the data. For example, running the code in Listing 8.10 here reveals that the likelihood of the alternative model is higher than the log likelihood of the OLS model. You can use the test on page 353 to test whether the difference in likelihood is truly significant.

$\sum$

- ➤ We can create models where the data used to estimate the parameters in one model is generated by estimation in another model.
- ➤ One common use is clustering: develop a model for individual groups (states, classrooms, families) and then discuss patterns among a set of groups (the country, school, or community).
- ➤ We can also use multiple levels to solve the snowflake problem, specifying a different probability of an event for every observation, but still retaining a simple basic model of events. For example, the probit is based on plugging the output from a standard linear form into a Normal CDF.