

Chapter 6

R Software

The methods described in this book are useful in any regression model that involves a linear combination of regression parameters. The software that is described below is useful in the same situations. Functions in R⁵²⁰ allow interaction spline functions as well as a wide variety of predictor parameterizations for any regression function, and facilitate model validation by resampling.

R is the most comprehensive tool for general regression models for the following reasons.

1. It is very easy to write R functions for new models, so R has implemented a wide variety of modern regression models.
2. Designs can be generated for any model. There is no need to find out whether the particular modeling function handles what SAS calls “class” variables—dummy variables are generated automatically when an R **category**, **factor**, **ordered**, or **character** variable is analyzed.
3. A single R object can contain all information needed to test hypotheses and to obtain predicted values for new data.
4. R has superior graphics.
5. *Classes* in R make possible the use of generic function names (e.g., **predict**, **summary**, **anova**) to examine fits from a large set of specific model-fitting functions.

R^{44,601,635} is a high-level object-oriented language for statistical analysis with over six thousand packages and tens of thousands of functions available. The R system^{318,520} is the basis for R software used in this text, centered around the Regression Modeling Strategies (**rms**) package²⁶¹. See the Appendix and the Web site for more information about software implementations.

1

6.1 The R Modeling Language

R has a battery of functions that make up a statistical modeling language.⁹⁶ At the heart of the modeling functions is an R *formula* of the form

```
response ~ terms
```

The `terms` represent additive components of a general linear model. Although variables and functions of variables make up the `terms`, the formula refers to additive combinations; for example, when `terms` is `age + blood.pressure`, it refers to $\beta_1 \times \text{age} + \beta_2 \times \text{blood.pressure}$. Some examples of formulas are below.

```
y ~ age + sex           # age + sex main effects
y ~ age + sex + age:sex # add second-order interaction
y ~ age*sex            # second-order interaction +
                        # all main effects
y ~ (age + sex + pressure)^2
                        # age+sex+pressure+age:sex+age:pressure...
y ~ (age + sex + pressure)^2 - sex:pressure
                        # all main effects and all 2nd order
                        # interactions except sex:pressure
y ~ (age + race)*sex   # age+race+sex+age:sex+race:sex
y ~ treatment*(age*race + age*sex)
                        # no interact. with race,sex
sqrt(y) ~ sex*sqrt(age) + race
# functions, with dummy variables generated if
# race is an R factor (classification) variable
y ~ sex + poly(age,2) # poly makes orthogonal polynomials
race.sex ← interaction(race,sex)
y ~ age + race.sex     # if desire dummy variables for all
                        # combinations of the factors
```

The formula for a regression model is given to a modeling function; for example,

```
lrm(y ~ rcs(x,4))
```

is read “use a logistic regression model to model `y` as a function of `x`, representing `x` by a restricted cubic spline with four default knots.”^a You can use the R function `update` to refit a model with changes to the model terms or the data used to fit it:

```
f ← lrm(y ~ rcs(x,4) + x2 + x3)
f2 ← update(f, subset=sex=="male")
f3 ← update(f, .~.-x2)           # remove x2 from model
f4 ← update(f, .~. + rcs(x5,5)) # add rcs(x5,5) to model
f5 ← update(f, y2 ~ .)         # same terms, new response var.
```

^a `lrm` and `rcs` are in the `rms` package.

6.2 User-Contributed Functions

In addition to the many functions that are packaged with R, a wide variety of user-contributed functions is available on the Internet (see the Appendix or Web site for addresses). Two packages of functions used extensively in this text are `Hmisc`²⁰ and `rms` written by the author. The `Hmisc` package contains miscellaneous functions such as `varclus`, `spearman2`, `transcan`, `hoeffd`, `rcspline.eval`, `impute`, `cut2`, `describe`, `sas.get`, `latex`, and several power and sample size calculation functions. The `varclus` function uses the R `hclust` hierarchical clustering function to do variable clustering, and the R `plclust` function to draw dendrograms depicting the clusters. `varclus` offers a choice of three similarity measures (Pearson r^2 , Spearman ρ^2 , and Hoeffding D) and uses pairwise deletion of missing values. `varclus` automatically generates a series of dummy variables for categorical factors. The `Hmisc` `hoeffd` function computes a matrix of Hoeffding D s for a series of variables. The `spearman2` function will do Wilcoxon, Spearman, and Kruskal–Wallis tests and generalizes Spearman’s ρ to detect non-monotonic relationships.

`Hmisc`’s `transcan` function (see Section 4.7) performs a similar function to `PROC PRINQUAL` in SAS—it uses restricted splines, dummy variables, and canonical variates to transform each of a series of variables while imputing missing values. An option to shrink regression coefficients for the imputation models avoids overfitting for small samples or a large number of predictors. `transcan` can also do multiple imputation and adjust variance–covariance matrices for imputation. See Chapter 8 for an example of using these functions for data reduction.

See the Web site for a list of R functions for correspondence analysis, principal component analysis, and missing data imputation available from other users. Venables and Ripley [635, Chapter 11] provide a nice description of the multivariate methods that are available in R, and they provide several new multivariate analysis functions.

A basic function in `Hmisc` is the `rcspline.eval` function, which creates a design matrix for a restricted (natural) cubic spline using the truncated power basis. Knot locations are optionally estimated using methods described in Section 2.4.6, and two types of normalizations to reduce numerical problems are supported. You can optionally obtain the design matrix for the anti-derivative of the spline function. The `rcspline.restate` function computes the coefficients (after un-normalizing if needed) that translate the restricted cubic spline function to unrestricted form (Equation 2.27). `rcspline.restate` also outputs \LaTeX and R representations of spline functions in simplified form.

6.3 The rms Package

A package of R functions called `rms` contains several functions that extend R to make the analyses described in this book easy to do. A central function in `rms` is `datadist`, which computes statistical summaries of predictors to automate estimation and plotting of effects. `datadist` exists as a separate function so that the candidate predictors may be summarized once, thus saving time when fitting several models using subsets or different transformations of predictors. If `datadist` is called before model fitting, the distributional summaries are stored with the fit so that the fit is self-contained with respect to later estimation. Alternatively, `datadist` may be called after the fit to create temporary summaries to use as plot ranges and effect intervals, or these ranges may be specified explicitly to `Predict` and `summary` (see below), without ever calling `datadist`. The input to `datadist` may be a data frame, a list of individual predictors, or a combination of the two.

The characteristics saved by `datadist` include the overall range and certain quantiles for continuous variables, and the distinct values for discrete variables (i.e., R factor variables or variables with 10 or fewer unique values). The quantiles and set of distinct values facilitate estimation and plotting, as described later. When a function of a predictor is used (e.g., `po1(pmin(x,50),2)`), the limits saved apply to the innermost variable (here, `x`). When a plot is requested for how `x` relates to the response, the plot will have `x` on the x -axis, not `pmin(x,50)`. The way that defaults are computed can be controlled by the `q.effect` and `q.display` parameters to `datadist`. By default, continuous variables are plotted with ranges determined by the tenth smallest and tenth largest values occurring in the data (if $n < 200$, the 0.05 and 0.95 quantiles are used). The default range for estimating effects such as odds and hazard ratios is the lower and upper quartiles. When a predictor is adjusted to a constant so that the effects of changes in other predictors can be studied, the default constant used is the median for continuous predictors and the most frequent category for factor variables. The R system option `datadist` is used to point to the result returned by the `datadist` function. See the help files for `datadist` for more information.

`rms` fitting functions save detailed information for later prediction, plotting, and testing. `rms` also allows for special restricted interactions and sets the default method of generating contrasts for categorical variables to "`contr.-treatment`", the traditional dummy-variable approach.

`rms` has a special operator `%ia%` in the terms of a formula that allows for restricted interactions. For example, one may specify a model that contains `sex` and a five-knot linear spline for `age`, but restrict the `age × sex` interaction to be linear in `age`. To be able to connect this incomplete interaction with the main effects for later hypothesis testing and estimation, the following formula would be given:

```
y ~ sex + lsp(age, c(20, 30, 40, 50, 60)) +
  sex %ia% lsp(age, c(20, 30, 40, 50, 60))
```

Table 6.1 rms Fitting Functions

Function	Purpose	Related R Functions
<code>ols</code>	Ordinary least squares linear model	<code>lm</code>
<code>lrm</code>	Binary and ordinal logistic regression model Has options for penalized MLE	<code>glm</code>
<code>orm</code>	Ordinal semi-parametric regression model with several link functions	<code>polr</code> , <code>lrm</code>
<code>psm</code>	Accelerated failure time parametric survival models	<code>survreg</code>
<code>cph</code>	Cox proportional hazards regression	<code>coxph</code>
<code>bj</code>	Buckley–James censored least squares model	<code>survreg</code> , <code>lm</code>
<code>glm</code>	General linear models	<code>glm</code>
<code>gls</code>	Generalized least squares	<code>gls</code>
<code>Rq</code>	Quantile regression	<code>rq</code>

The following expression would restrict the $\text{age} \times \text{cholesterol}$ interaction to be of the form $AF(B) + BG(A)$ by removing doubly nonlinear terms.

```
y ~ lsp(age,30) + rcs(cholesterol,4) +
  lsp(age,30) %ia% rcs(cholesterol,4)
```

`rms` has special fitting functions that facilitate many of the procedures described in this book, shown in Table 6.1.

`glm` is a slight modification of the built-in R `glm` function so that `rms` methods can be run on the resulting fit object. `glm` fits general linear models under a wide variety of distributions of Y . `gls` is a modification of the `gls` function from the `nlme` package of Pinheiro and Bates⁵⁰⁹, for repeated measures (longitudinal) and spatially correlated data. The `Rq` function is a modification of the `quantreg` package's `rq` function^{356,357}. Functions related to survival analysis make heavy use of Therneau's `survival` package⁴⁸².

You may want to specify to the fitting functions an option for how missing values (`NA`s) are handled. The method for handling missing data in R is to specify an `na.action` function. Some possible `na.actions` are given in Table 6.2. The default `na.action` is `na.delete` when you use `rms`'s fitting functions. An easy way to specify a new default `na.action` is, for example,

```
options(na.action="na.omit") # don't report frequency of NAs
```

before using a fitting function. If you use `na.delete` you can also use the system option `na.detail.response` that makes model fits store information about Y stratified by whether each X is missing. The default descriptive statistics for Y are the sample size and mean. For a survival time response object the sample size and proportion of events are used. Other summary functions can be specified using the `na.fun.response` option.

Table 6.2 Some `na.actions` Used in `rms`

Function Name	Method Used
<code>na.fail</code>	Stop with error message if any missing values present
<code>na.omit</code>	Function to remove observations with any predictors or responses missing
<code>na.delete</code>	Modified version of <code>na.omit</code> to also report on frequency of NAs for each variable

```
options(na.action="na.delete", na.detail.response=TRUE,
        na.fun.response="mystats")
# Just use na.fun.response="quantile" if don't care about n
mystats <- function(y) {
  z <- quantile(y, na.rm=T)
  n <- sum(!is.na(y))
  c(N=n, z)      # elements named N, 0%, 25%, etc.
}
```

When R deletes missing values during the model-fitting procedure, residuals, fitted values, and other quantities stored with the fit will not correspond row-for-row with observations in the original data frame (which retained NAs). This is problematic when, for example, `age` in the dataset is plotted against the residual from the fitted model. Fortunately, for many `na.actions` including `na.delete` and a modified version of `na.omit`, a class of R functions called `naresid` written by Therneau works behind the scenes to put NAs back into residuals, predicted values, and other quantities when the `predict` or `residuals` functions (see below) are used. Thus for some of the `na.actions`, predicted values and residuals will automatically be arranged to match the original data.

Any R function can be used in the terms for formulas given to the fitting function, but if the function represents a transformation that has data-dependent parameters (such as the standard R functions `poly` or `ns`), R will not in general be able to compute predicted values correctly for new observations. For example, the function `ns` that automatically selects knots for a B-spline fit will not be conducive to obtaining predicted values if the knots are kept “secret.” For this reason, a set of functions that keep track of transformation parameters, exists in `rms` for use with the functions highlighted in this book. These are shown in Table 6.3. Of these functions, `asis`, `catg`, `scored`, and `matrx` are almost always called implicitly and are not mentioned by the user. `catg` is usually called explicitly when the variable is a numeric variable to be used as a polytomous factor, and it has not been converted to an R categorical variable using the `factor` function.

Table 6.3 rms Transformation Functions

Function	Purpose	Related R Functions
<code>asis</code>	No post-transformation (seldom used explicitly)	<code>I</code>
<code>rcs</code>	Restricted cubic spline	<code>ns</code>
<code>pol</code>	Polynomial using standard notation	<code>poly</code>
<code>lsp</code>	Linear spline	
<code>catg</code>	Categorical predictor (seldom)	<code>factor</code>
<code>scored</code>	Ordinal categorical variables	<code>ordered</code>
<code>matrx</code>	Keep variables as group for <code>anova</code> and <code>fastbw</code>	<code>matrix</code>
<code>strat</code>	Nonmodeled stratification factors (used for <code>cph</code> only)	<code>strata</code>

These functions can be used with any function of a predictor. For example, to obtain a four-knot cubic spline expansion of the cube root of x , specify `rcs(x^(1/3),4)`.

When the transformation functions are called, they are usually given one or two arguments, such as `rcs(x,5)`. The first argument is the predictor variable or some function of it. The second argument is an optional vector of parameters describing a transformation, for example location or number of knots. Other arguments may be provided.

The `Hmisc` package's `cut2` function is sometimes used to create a categorical variable from a continuous variable x . You can specify the actual interval endpoints (`cuts`), the number of observations to have in each interval on the average (`m`), or the number of quantile groups (`g`). Use, for example, `cuts=c(0,1,2)` to cut into the intervals $[0, 1)$, $[1, 2]$.

A key concept in fitting models in R is that the fitting function returns an object that is an R list. This object contains basic information about the fit (e.g., regression coefficient estimates and covariance matrix, model χ^2) as well as information about how each parameter of the model relates to each factor in the model. Components of the fit object are addressed by, for example, `fit$coef`, `fit$var`, `fit$loglik`. `rms` causes the following information to also be retained in the fit object: the limits for plotting and estimating effects for each factor (if `options(datadist="name")` was in effect), the label for each factor, and a vector of values indicating which parameters associated with a factor are nonlinear (if any). Thus the “fit object” contains all the information needed to get predicted values, plots, odds or hazard ratios, and hypothesis tests, and to do “smart” variable selection that keeps parameters together when they are all associated with the same predictor.

R uses the notion of the *class* of an object. The object-oriented class idea allows one to write a few generic functions that decide which specific functions to call based on the class of the object passed to the generic function. An example is the function for printing the main results of a logistic model.

The `lrm` function returns a fit object of class "lrm". If you specify the R command `print(fit)` (or just `fit` if using R interactively—this invokes `print`), the `print` function invokes the `print.lrm` function to do the actual printing specific to logistic models. To find out which particular methods are implemented for a given generic function, type `methods(generic.name)`.

Generic functions that are used in this book include those in Table 6.4.

Table 6.4 rms Package and R Generic Functions

Function	Purpose	Related Functions
<code>print</code>	Print parameters and statistics of fit	
<code>coef</code>	Fitted regression coefficients	
<code>formula</code>	Formula used in the fit	
<code>specs</code>	Detailed specifications of fit	
<code>vcov</code>	Fetch covariance matrix	
<code>logLik</code>	Fetch maximized log-likelihood	
<code>AIC</code>	Fetch AIC	
<code>lrtest</code>	Likelihood ratio test for two nested models	
<code>univarLR</code>	Compute all univariable LR χ^2	
<code>robcov</code>	Robust covariance matrix estimates	
<code>bootcov</code>	Bootstrap covariance matrix estimates and bootstrap distributions of estimates	
<code>pentrace</code>	Find optimum penalty factors by tracing effective AIC for a grid of penalties	
<code>effective.df</code>	Print effective d.f. for each type of variable in model, for penalized fit or <code>pentrace</code> result	
<code>summary</code>	Summary of effects of predictors	
<code>plot.summary</code>	Plot continuously shaded confidence bars for results of <code>summary</code>	
<code>anova</code>	Wald tests of most meaningful hypotheses	
<code>plot.anova</code>	Graphical depiction of <code>anova</code>	
<code>contrast</code>	General contrasts, C.L., tests	
<code>Predict</code>	Predicted values and confidence limits easily varying a subset of predictors and leaving the rest set at default values	
<code>plot.Predict</code>	Plot the result of <code>Predict</code> using <code>lattice</code>	
<code>ggplot</code>	Plot the result of <code>Predict</code> using <code>ggplot2</code>	
<code>bplot</code>	3-dimensional plot when <code>Predict</code> varied two continuous predictors over a fine grid	
<code>gendata</code>	Easily generate predictor combinations	
<code>predict</code>	Obtain predicted values or design matrix	
<code>fastbw</code>	Fast backward step-down variable selection	<code>step</code>
<code>residuals</code>	(or <code>resid</code>) Residuals, influence stats from fit	
<code>sensuc</code>	Sensitivity analysis for unmeasured confounder	
<code>which.influence</code>	Which observations are overly influential	<code>residuals</code>
<code>latex</code>	L ^A T _E X representation of fitted model	<code>Function</code>

continued on next page

<i>continued from previous page</i>		
Function	Purpose	Related Functions
Function	R function analytic representation of $X\hat{\beta}$ from a fitted regression model	latex
Hazard	R function analytic representation of a fitted hazard function (for psm)	
Survival	R function analytic representation of fitted survival function (for psm , cph)	
ExProb	R function analytic representation of exceedance probabilities for orm	
Quantile	R function analytic representation of fitted function for quantiles of survival time (for psm , cph)	
Mean	R function analytic representation of fitted function for mean survival time or for ordinal logistic	
nomogram	Draws a nomogram for the fitted model	latex , plot
survest	Estimate survival probabilities (psm , cph)	survfit
survplot	Plot survival curves (psm , cph)	plot.survfit
validate	Validate indexes of model fit using resampling	
calibrate	Estimate calibration curve using resampling	val.prob
vif	Variance inflation factors for fitted model	
naresid	Bring elements corresponding to missing data back into predictions and residuals	
naprint	Print summary of missing values	
impute	Impute missing values	transcan

The first argument of the majority of functions is the object returned from the model fitting function. When used with **ols**, **lrm**, **orm**, **psm**, **cph**, **Glm**, **Gls**, **Rq**, **bj**, these functions do the following. **specs** prints the design specifications, for example, number of parameters for each factor, levels of categorical factors, knot locations in splines, and so on. **vcov** returns the variance-covariance matrix for the model. **logLik** retrieves the maximized log-likelihood, whereas **AIC** computes the Akaike Information Criterion for the model on the minus twice log-likelihood scale (with an option to compute it on the χ^2 scale if you specify **type='chisq'**). **lrtest**, when given two fit objects from nested models, computes the likelihood ratio test for the extra variables. **univarLR** computes all univariable likelihood ratio χ^2 statistics, one predictor at a time.

The **robcov** function computes the Huber robust covariance matrix estimate. **bootcov** uses the bootstrap to estimate the covariance matrix of parameter estimates. Both **robcov** and **bootcov** assume that the design matrix and response variable were stored with the fit. They have options to adjust for cluster sampling. Both replace the original variance-covariance matrix with robust estimates and return a new fit object that can be passed to any of the other functions. In that way, robust Wald tests, variable selection, confidence limits, and many other quantities may be computed automatically. The functions do save the old covariance estimates in component **orig.var** of the new fit object. **bootcov** also optionally returns the matrix of parameter estimates over the bootstrap simulations. These estimates can be used to derive bootstrap confidence intervals that don't assume normality or symmetry. Associated with **bootcov** are plotting functions for drawing histogram

and smooth density estimates for bootstrap distributions. `bootcov` also has a feature for deriving approximate nonparametric simultaneous confidence sets. For example, the function can get a simultaneous 0.90 confidence region for the regression effect of age over its entire range.

The `pentrace` function assists in selection of penalty factors for fitting regression models using penalized maximum likelihood estimation (see Section 9.10). Different types of model terms can be penalized by different amounts. For example, one can penalize interaction terms more than main effects. The `effective.df` function prints details about the effective degrees of freedom devoted to each type of model term in a penalized fit.

`summary` prints a summary of the effects of each factor. When `summary` is used to estimate effects (e.g., odds or hazard ratios) for continuous variables, it allows the levels of interacting factors to be easily set, as well as allowing the user to choose the interval for the effect. This method of estimating effects allows for nonlinearity in the predictor. By default, interquartile range effects (differences in $X\hat{\beta}$, odds ratios, hazards ratios, etc.) are printed for continuous factors, and all comparisons with the reference level are made for categorical factors. See the example at the end of the `summary` documentation for a method of quickly computing pairwise treatment effects and confidence intervals for a large series of values of factors that interact with the treatment variable. Saying `plot(summary(fit))` will depict the effects graphically, with bars for a list of confidence levels.

The `anova` function automatically tests most meaningful hypotheses in a design. For example, suppose that age and cholesterol are predictors, and that a general interaction is modeled using a restricted spline surface. `anova` prints Wald statistics for testing linearity of age, linearity of cholesterol, age effect (age + age \times cholesterol interaction), cholesterol effect (cholesterol + age \times cholesterol interaction), linearity of the age \times cholesterol interaction (i.e., adequacy of the simple age \times cholesterol 1 d.f. product), linearity of the interaction in age alone, and linearity of the interaction in cholesterol alone. Joint tests of all interaction terms in the model and all nonlinear terms in the model are also performed. The `plot.anova` function draws a dot chart showing the relative contribution (χ^2 , χ^2 minus d.f., AIC, partial R^2 , P -value, etc.) of each factor in the model.

The `contrast` function is used to obtain general contrasts and corresponding confidence limits and test statistics. This is most useful for testing effects in the presence of interactions (e.g., type II and type III contrasts). See the help file for `contrast` for several examples of how to obtain joint tests of multiple contrasts (see Section 9.3.2) as well as double differences (interaction contrasts).

The `predict` function is used to obtain a variety of values or predicted values from either the data used to fit the model or a new dataset. The `Predict` function is easier to use for most purposes, and has a special `plot` method. The `gendata` function makes it easy to obtain a data frame containing predictor combinations for obtaining selected predicted values.

The `fastbw` function performs a slightly inefficient but numerically stable version of fast backward elimination on factors, using a method based on Lawless and Singhal.³⁸⁵ This method uses the fitted complete model and computes approximate Wald statistics by computing conditional (restricted) maximum likelihood estimates assuming multivariate normality of estimates. It can be used in simulations since it returns indexes of factors retained and dropped:

```
fit ← ols(y ~ x1*x2*x3)
# run, and print results:
fastbw(fit, optional_arguments)
# typically used in simulations:
z ← fastbw(fit, optional_args)
# least squares fit of reduced model:
lm.fit(X[,z$params.kept], Y)
```

`fastbw` deletes factors, not columns of the design matrix. Factors requiring multiple d.f. will be retained or dropped as a group. The function prints the deletion statistics for each variable in turn, and prints approximate parameter estimates for the model after deleting variables. The approximation is better when the number of factors deleted is not large. For `ols`, the approximation is exact.

The `which.influence` function creates a list with a component for each factor in the model. The names of the components are the factor names. Each component contains the observation identifiers of all observations that are “overly influential” with respect to that factor, meaning that $|\mathbf{dfbetas}| > u$ for at least one β_i associated with that factor, for a given u . The default u is `.2`. You must have specified `x=TRUE`, `y=TRUE` in the fitting function to use `which.influence`. The first argument is the fit object, and the second argument is the cutoff u .

The following R program will print the set of predictor values that were very influential for each factor. It assumes that the data frame containing the data used in the fit is called `df`.

```
f ← lrm(y ~ x1 + x2 + ..., data=df, x=TRUE, y=TRUE)
w ← which.influence(f, .4)
nam ← names(w)
for(i in 1:length(nam)) {
  cat("Influential observations for effect of",
      nam[i], "\n")
  print(df[w[[i]],])
}
```

The `latex` function is a generic function available in the `Hmisc` package. It invokes a specific `latex` function for most of the fit objects created by `rms` to create a \LaTeX algebraic representation of the fitted model for inclusion in a report or viewing on the screen. This representation documents all parameters in the model and the functional form being assumed for Y , and is especially useful for getting a simplified version of restricted cubic spline functions. On

the other hand, the `print` method with optional argument `latex=TRUE` is used to output L^AT_EX code representing the model results in tabular form to the console. This is intended for use with `knitr`⁶⁷⁷ or `Sweave`³⁹⁹.

The `Function` function composes an R function that you can use to evaluate $X\hat{\beta}$ analytically from a fitted regression model. The documentation for `Function` also shows how to use a subsidiary function `sascode` that will (almost) translate such an R function into SAS code for evaluating predicted values in new subjects. Neither `Function` nor `latex` handles third-order interactions.

The `nomogram` function draws a partial nomogram for obtaining predictions from the fitted model manually. It constructs different scales when interactions (up to third-order) are present. The constructed nomogram is not complete, in that point scores are obtained for each predictor and the user must add the point scores manually before reading predicted values on the final axis of the nomogram. The constructed nomogram is useful for interpreting the model fit, especially for non-monotonically transformed predictors (their scales wrap around an axis automatically).

The `vif` function computes variance inflation factors from the covariance matrix of a fitted model, using [147,654].

The `impute` function is another generic function. It does simple imputation by default. It can also work with the `transcan` function to multiply or singly impute missing values using a flexible additive model.

As an example of using many of the functions, suppose that a categorical variable `treat` has values "a", "b", and "c", an ordinal variable `num.diseases` has values 0,1,2,3,4, and that there are two continuous variables, `age` and `cholesterol`. `age` is fitted with a restricted cubic spline, while `cholesterol` is transformed using the transformation `log(cholesterol+10)`. Cholesterol is missing on three subjects, and we impute these using the overall median cholesterol. We wish to allow for interaction between `treat` and `cholesterol`. The following R program will fit a logistic model, test all effects in the design, estimate effects, and plot estimated transformations. The fit for `num.diseases` really considers the variable to be a five-level categorical variable. The only difference is that a 3 d.f. test of linearity is done to assess whether the variable can be remodeled "asis". Here we also show statements to attach the `rms` package and store predictor characteristics from `datadist`.

```
require(rms) # make new functions available
ddist <- datadist(cholesterol, treat, num.diseases, age)
# Could have used ddist <- datadist(data.frame.name)
options(datadist="ddist") # defines data dist. to rms
cholesterol <- impute(cholesterol)
fit <- lrm(y ~ treat + scored(num.diseases) + rcs(age) +
          log(cholesterol+10) +
          treat:log(cholesterol+10))
describe(y ~ treat + scored(num.diseases) + rcs(age))
# or use describe(formula(fit)) for all variables used in
# fit. describe function (in Hmisc) gets simple statistics
# on variables
# fit <- robcov(fit)# Would make all statistics that follow
```

```

# use a robust covariance matrix
# would need x=TRUE, y=TRUE in lrm()
# Describe the design characteristics
specs(fit)
anova(fit)
anova(fit, treat, cholesterol) # Test these 2 by themselves
plot(anova(fit)) # Summarize anova graphically
summary(fit) # Est. effects; default ranges
plot(summary(fit)) # Graphical display of effects with C.I.
# Specific reference cell and adjustment value:
summary(fit, treat="b", age=60)
# Estimate effect of increasing age: 50->70
summary(fit, age=c(50,70))
# Increase age 50->70, adjust to 60 when estimating
# effects of other factors:
summary(fit, age=c(50,60,70))
# If had not defined datadist, would have to define
# ranges for all variables

# Estimate and test treatment (b-a) effect averaged
# over 3 cholesterol:
contrast(fit, list(treat='b', cholesterol=c(150,200,250)),
         list(treat='a', cholesterol=c(150,200,250)),
         type='average')

p ← Predict(fit, age=seq(20,80,length=100), treat,
           conf.int=FALSE)
plot(p) # Plot relationship between age and
# or ggplot(p) # log odds, separate curve for each
# treat, no C.I.
plot(p, ~ age | treat) # Same but 2 panels
ggplot(p, groups=FALSE)
bplot(Predict(fit, age, cholesterol, np=50))
# 3-dimensional perspective plot for
# age, cholesterol, and log odds
# using default ranges for both
# Plot estimated probabilities instead of log odds:
plot(Predict(fit, num.diseases,
            fun=function(x) 1/(1+exp(-x)),
            conf.int=.9), ylab="Prob")
# Again, if no datadist were defined, would have to tell
# plot all limits
logit ← predict(fit, expand.grid(treat="b", num.dis=1:3,
                                age=c(20,40,60),
                                cholesterol=seq(100,300,length=10)))
# Could obtain list of predictor settings interactively
logit ← predict(fit, gendata(fit, nobs=12))
# An easier approach is
# Predict(fit, treat='b', num.dis=1:3, ...)

# Since age doesn't interact with anything, we can quickly
# and interactively try various transformations of age,
# taking the spline function of age as the gold standard.
# We are seeking a linearizing transformation.

```

```

ag ← 10:80
logit ← predict(fit, expand.grid(treat="a", num.dis=0,
                               age=ag,
                               cholesterol=median(cholesterol)),
               type="terms")[,"age"]
# Note: if age interacted with anything, this would be the
# age `main effect' ignoring interaction terms
# Could also use logit ← Predict(f, age=ag, ...)$yhat,
# which allows evaluation of the shape for any level of
# interacting factors. When age does not interact with
# anything, the result from predict(f, ..., type="terms")
# would equal the result from Predict if all other terms
# were ignored

# Could also specify:
# logit ← predict(fit,
#                 gendata(fit, age=ag, cholesterol=...))
# Unmentioned variables are set to reference values

plot(ag^.5, logit) # try square root vs. spline transform.
plot(ag^1.5, logit) # try 1.5 power

# Pretty printing of table of estimates and
# summary statistics:
print(fit, latex=TRUE) # print  $\LaTeX$  code to console
latex(fit) # invokes latex.lrm, creates fit.tex
# Draw a nomogram for the model fit
plot(nomogram(fit))

# Compose R function to evaluate linear predictors
# analytically
g ← Function(fit)
g(treat='b', cholesterol=260, age=50)
# Letting num.diseases default to reference value

```

To examine interactions in a simpler way, you may want to group age into tertiles:

```

age.tertile ← cut2(age, g=3)
# For auto ranges later, specify age.tertile to datadist
fit ← lrm(y ~ age.tertile * rcs(cholesterol))

```

Example output from these functions is shown in Chapter 10 and later chapters.

Note that `type="terms"` in `predict` scores each factor in a model with its fitted transformation. This may be used to compute, for example, rank correlation between the response and each transformed factor, pretending it has 1 d.f.

When regression is done on principal components, one may use an ordinary linear model to decode “internal” regression coefficients for helping to understand the final model. Here is an example.

```

require(rms)
dd ← datadist(my.data)
options(datadist='dd')
pcfit ← princomp(~ pain.symptom1 + pain.symptom2 + sign1 +
                 sign2 + sign3 + smoking)
pc2 ← pcfit$scores[,1:2] # first 2 PCs as matrix
logistic.fit ← lrm(death ~ rcs(age,4) + pc2)
predicted.logit ← predict(logistic.fit)
linear.mod ← ols(predicted.logit ~ rcs(age,4) +
                 pain.symptom1 + pain.symptom2 +
                 sign1 + sign2 + sign3 + smoking)
# This model will have R-squared=1
nom ← nomogram(linear.mod, fun=function(x)1/(1+exp(-x)),
               funlabel="Probability of Death")
# can use fun=plogis
plot(nom)
# 7 Axes showing effects of all predictors, plus a reading
# axis converting to predicted probability scale

```

In addition to many of the add-on functions described above, there are several other R functions that validate models. The first, `predab.resample`, is a general-purpose function that is used by functions for specific models described later. `predab.resample` computes estimates of optimism and bias-corrected estimates of a vector of indexes of predictive accuracy, for a model with a specified design matrix, with or without fast backward step-down of predictors. If `bw=TRUE`, `predab.resample` prints a matrix of asterisks showing which factors were selected at each repetition, along with a frequency distribution of the number of factors retained across resamples. The function has an optional parameter that may be specified to force the bootstrap algorithm to do sampling with replacement from clusters rather than from original records, which is useful when each subject has multiple records in the dataset. It also has a parameter that can be used to validate predictions in a subset of the records even though models are refit using all records.

The generic function `validate` invokes `predab.resample` with model-specific fits and measures of accuracy. The function `calibrate` invokes `predab.resample` to estimate bias-corrected model calibration and to plot the calibration curve. Model calibration is estimated at a sequence of predicted values.

6.4 Other Functions

For principal component analysis, R has the `princomp` and `prcomp` functions. Canonical correlations and canonical variates can be easily computed using the `cancor` function. There are many other R functions for examining associations and for fitting models. The `supsmu` function implements Friedman's "super smoother."²⁰⁷ The `lowess` function implements Cleveland's two-dimensional smoother.¹¹¹ The `glm` function will fit general linear models under

a wide variety of distributions of Y . There are functions to fit Hastie and Tibshirani's²⁷⁵ generalized additive model for a variety of distributions. More is said about parametric and nonparametric additive multiple regression functions in Chapter 16. The `loess` function fits a multidimensional scatterplot smoother (the local regression model of Cleveland et al.⁹⁶). `loess` provides approximate test statistics for normal or symmetrically distributed Y :

```
f ← loess(y ~ age * pressure)
plot(f) # cross-sectional plots
ages ← seq(20,70,length=40)
pressures ← seq(80,200,length=40)
pred ← predict(f,
               expand.grid(age=ages, pressure=pressures))
persp(ages, pressures, pred) # 3-D plot
```

`loess` has a large number of options allowing various restrictions to be placed on the fitted surface.

Atkinson and Therneau's `rpart` recursive partitioning package and related functions implement classification and regression trees⁶⁹ algorithms for binary, continuous, and right-censored response variables (assuming an exponential distribution for the latter). `rpart` deals effectively with missing predictor values using surrogate splits. The `rms` package has a `validate` function for `rpart` objects for obtaining cross-validated mean squared errors and Somers' D_{xy} rank correlations (Brier score and ROC areas for probability models).

For displaying which variables tend to be missing on the same subjects, the `Hmisc` `naclus` function can be used (e.g., `plot(naclus(dataframename))` or `naplot(naclus(dataframename))`). For characterizing what type of subjects have NA's on a given predictor (or response) variable, a tree model whose response variable is `is.na(varname)` can be quite useful.

```
require(rpart)
f ← rpart(is.na(cholesterol) ~ age + sex + trig + smoking)
plot(f) # plots the tree
text(f) # labels the tree
```

The `Hmisc` `rcorr.cens` function can compute Somers' D_{xy} rank correlation coefficient and its standard error, for binary or continuous (and possibly right-censored) responses. A simple transformation of D_{xy} yields the c index (generalized ROC area). The `Hmisc` `improveProb` function is useful for comparing two probability models using the methods of Pencina et al.^{490,492,493} in an external validation setting. See also the `rcorrrp.cens` function in this context.

6.5 Further Reading

- ① Harrell and Goldstein²⁶³ list components of statistical languages or packages and compare several popular packages for survival analysis capabilities.
- ② Imai et al.³¹⁹ have further generalized R as a statistical modeling language.