Chapter 6

Linear regression and ANOVA

Regression and analysis of variance (ANOVA) form the basis of many investigations. Here we describe how to undertake many common tasks in linear regression (broadly defined), while Chapter 7 discusses many generalizations, including other types of outcome variables, longitudinal and clustered analysis, and survival methods.

Many R commands can perform linear regression, as it constitutes a special case of which many models are generalizations. We present detailed descriptions for the `lm()` command, as it offers the most flexibility and best output options tailored to linear regression in particular. While ANOVA can be viewed as a special case of linear regression, separate routines are available (`aov()`) to perform it.

R supports a flexible modeling language implemented using formulas (see `help(formula)` and 6.1.1) for regression that shares functionality with the lattice graphics functions (as well as other packages). Many of the routines available within R return or operate on `lm` class objects, which include objects such as coefficients, residuals, fitted values, weights, contrasts, model matrices, and similar quantities (see `help(lm)`).

The CRAN statistics for the social sciences task view provides an excellent overview of methods described here and in Chapter 7.

6.1 Model fitting

6.1.1 Linear regression

Example: 6.6.2

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
summary(mod1)
summary.aov(mod1)
```

or

```
form = as.formula(y ~ x1 + ... + xk)
mod1 = lm(form, data=ds)
summary(mod1)
coef(mod1)
```

Note: The first argument of the `lm()` function is a formula object, with the outcome specified followed by the $\sim$ operator then the predictors. It returns a linear model object. More information about the linear model `summary()` command can be found using `help(summary.lm)`. The `coef()` function extracts coefficients from a model (see also the `coefplot` package). The `biglm()` function in the `biglm` package can support model fitting to very large datasets (see 6.1.7). By default, stars are used to annotate the output of

the `summary()` functions regarding significance levels: these can be turned off using the command `options(show.signif.stars=FALSE)`.

6.1.2 Linear regression with categorical covariates

Example: 6.6.2

See 6.1.4 (parameterization of categorical covariates).

```
ds = transform(ds, x1f = as.factor(x1))
mod1 = lm(y ~ x1f + x2 + ... + xk, data=ds)
```

Note: The `as.factor()` command creates a categorical variable from a variable. By default, the lowest value (either numerically or lexicographically) is the reference value. The `levels` option for the `factor()` function can be used to select a particular reference value (see 2.2.19). Ordered factors can be constructed using the `ordered()` function.

6.1.3 Changing the reference category

```
library(dplyr)
ds = mutate(ds, neworder = factor(classvar,
   levels=c("level", "otherlev1", "otherlev2")))
mod1 = lm(y ~ neworder, data=ds)
```

Note: The first level of a factor (by default, that which appears first lexicographically) is the reference group. This can be modified through use of the `factor()` function.

6.1.4 Parameterization of categorical covariates

Example: 6.6.6

The `as.factor()` function can be applied within any model-fitting command. Parameterization of the covariate can be controlled as below.

```
ds = transform(ds, x1f = as.factor(x1))
mod1 = lm(y ~ x1f, contrasts=list(x1f="contr.SAS"), data=ds)
```

Note: The `as.factor()` function creates a factor object. The `contrasts` option for the `lm()` function specifies how the levels of that factor object should be used within the function. The `levels` option to the `factor()` function allows specification of the ordering of levels (the default is lexicographic). An example can be found in Section 6.6.

The specification of the design matrix for analysis of variance and regression models can be controlled using the `contrasts` option. Examples of options (for a factor with four equally spaced levels) are given below.

```
> contr.treatment(4)
  2 3 4
1 0 0 0
2 1 0 0
3 0 1 0
4 0 0 1
> contr.SAS(4)
  1 2 3
1 1 0 0
2 0 1 0
```

```
> contr.poly(4)
              .L   .Q     .C
[1,] -0.671  0.5 -0.224
[2,] -0.224 -0.5  0.671
[3,]  0.224 -0.5 -0.671
[4,]  0.671  0.5  0.224
> contr.sum(4)
  [,1] [,2] [,3]
1    1    0    0
2    0    1    0
```

```
3 0 0 1                                  3    0    0    1
4 0 0 0                                  4   -1   -1   -1
> contr.helmert(4)
  [,1] [,2] [,3]
1   -1   -1   -1
2    1   -1   -1
3    0    2   -1
4    0    0    3
```

See `options("contrasts")` for defaults, and `contrasts()` or `C()` to apply a contrast function to a factor variable. Support for reordering factors is available within the `factor()` function.

6.1.5 Linear regression with no intercept

```
mod1 = lm(y ~ 0 + x1 + ... + xk, data=ds)
```

or

```
mod1 = lm(y ~ x1 + ... + xk - 1, data=ds)
```

6.1.6 Linear regression with interactions

Example: 6.6.2

```
mod1 = lm(y ~ x1 + x2 + x1:x2 + x3 + ... + xk, data=ds)
```

or

```
lm(y ~ x1*x2 + x3 + ... + xk, data=ds)
```

Note: The `*` operator includes all lower-order terms (in this case main effects), while the `:` operator includes only the specified interaction. So, for example, the commands `y ~ x1*x2*x3` and `y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3` are equivalent. The syntax also works with any covariates designated as categorical using the `as.factor()` command (see 6.1.2).

6.1.7 Linear regression with big data

```
library(biglm)
myformula = as.formula(y ~ x1)
res = biglm(myformula, chunk1)
res = update(res, chunk2)
coef(res)
```

Note: The `biglm()` and `update()` functions in the `biglm` package can fit linear (or generalized linear) models with dataframes larger than memory. It allows a single large model to be estimated in more manageable chunks, with results updated iteratively as each chunk is processed. The chunk size will depend on the application. The data argument may be a function, dataframe, `SQLiteConnection`, or `RODBC` connection object.

6.1.8 One-way analysis of variance

Example: 6.6.6

```
ds = transform(ds, xf=as.factor(x))
mod1 = aov(y ~ xf, data=ds)
summary(mod1)
anova(mod1)
```

Note: The `summary()` command can be used to provide details of the model fit. More information can be found using `help(summary.aov)`. Note that `summary.lm(mod1)` will display the regression parameters underlying the ANOVA model.

6.1.9 Analysis of variance with two or more factors

Example: 6.6.6

Interactions can be specified using the syntax introduced in 6.1.6 (see interaction plots, 8.5.2).

```
aov(y ~ as.factor(x1) + as.factor(x2), data=ds)
```

6.2 Tests, contrasts, and linear functions of parameters

6.2.1 Joint null hypotheses: several parameters equal 0

As an example, consider testing the null hypothesis $H_0 : \beta_1 = \beta_2 = 0$.

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
mod2 = lm(y ~ x3 + ... + xk, data=ds)
anova(mod2, mod1)
```

6.2.2 Joint null hypotheses: sum of parameters

As an example, consider testing the null hypothesis $H_0 : \beta_1 + \beta_2 = 1$.

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
covb = vcov(mod1)
coeff.mod1 = coef(mod1)
t = (coeff.mod1[2] + coeff.mod1[3] - 1)/
   sqrt(covb[2,2] + covb[3,3] + 2*covb[2,3])
pvalue = 2*(1-pt(abs(t), df=mod1$df))
```

6.2.3 Tests of equality of parameters

Example: 6.6.8

As an example, consider testing the null hypothesis $H_0 : \beta_1 = \beta_2$.

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
mod2 = lm(y ~ I(x1+x2) + ... + xk, data=ds)
anova(mod2, mod1)
```

or

```
library(gmodels)
estimable(mod1, c(0, 1, -1, 0, ..., 0))
```

or

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
covb = vcov(mod1)
coeff.mod1 = coef(mod1)
t = (coeff.mod1[2]-coeff.mod1[3])/sqrt(covb[2,2]+covb[3,3]-2*covb[2,3])
pvalue = 2*(1-pt(abs(t), mod1$df))
```

Note: The `I()` function inhibits the interpretation of operators, to allow them to be used as arithmetic operators. The `estimable()` function calculates a linear combination of the parameters. The more general code below utilizes the same approach introduced in 6.2.1 for the specific test of $\beta_1 = \beta_2$ (different coding would be needed for other comparisons).

6.2.4 Multiple comparisons

Example: 6.6.7

```
mod1 = aov(y ~ x, data=ds)
TukeyHSD(mod1, "x")
```

Note: The `TukeyHSD()` function takes an `aov` object as an argument and evaluates pairwise comparisons between all of the combinations of the factor levels of the variable `x`. (See the `p.adjust()` function, as well as the `multcomp` and `factorplot` packages for other multiple comparison methods, including Bonferroni, Holm, Hochberg, and false discovery rate adjustments.)

6.2.5 Linear combinations of parameters

Example: 6.6.8

It is often useful to find predicted values for particular covariate values. Here, we calculate the predicted value $E[Y|X_1 = 1, X_2 = 3] = \hat{\beta}_0 + \hat{\beta}_1 + 3\hat{\beta}_2$.

```
mod1 = lm(y ~ x1 + x2, data=ds)
newdf = data.frame(x1=c(1), x2=c(3))
predict(mod1, newdf, se.fit=TRUE, interval="confidence")
```

or

```
library(gmodels)
estimable(mod1, c(1, 1, 3))
```

or

```
library(mosaic)
myfun = makeFun(mod1)
myfun(x1=1, x2=3)
```

Note: The `predict()` command in R can generate estimates at any combination of parameter values, as specified as a dataframe that is passed as an argument. More information on this function can be found using `help(predict.lm)`.

6.3 Model results and diagnostics

There are many functions available to produce predicted values and diagnostics. For additional commands not listed here, see `help(influence.measures)` and the "See also" in `help(lm)`.

6.3.1 Predicted values

Example: 6.6.2

```
mod1 = lm(y ~ x, data=ds)
predicted.varname = predict(mod1)
```

Note: The command `predict()` operates on any `lm` object and by default generates a vector of predicted values.

6.3.2 Residuals

Example: 6.6.2

```
mod1 = lm(y ~ x, data=ds)
residual.varname = residuals(mod1)
```

Note: The command `residuals()` operates on any `lm` object and generates a vector of residuals. Other functions exist for `aov`, `glm`, or `lme` objects (see `help(residuals.glm)`).

6.3.3 Standardized and Studentized residuals

Example: 6.6.2

Standardized residuals are calculated by dividing the ordinary residual (observed minus expected, $y_i - \hat{y}_i$) by an estimate of its standard deviation. Studentized residuals are calculated in a similar manner, where the predicted value and the variance of the residual are estimated from the model fit while excluding that observation.

```
mod1 = lm(y ~ x, data=ds)
standardized.resid.varname = rstandard(mod1)
studentized.resid.varname = rstudent(mod1)
```

Note: The `rstandard()` and `rstudent()` functions operate on any `lm` object, and generate a vector of studentized residuals (the former command includes the observation in the calculation, while the latter does not).

6.3.4 Leverage

Example: 6.6.2

Leverage is defined as the diagonal element of the $(X(X^TX)^{-1}X^T)$ or "hat" matrix.

```
mod1 = lm(y ~ x, data=ds)
leverage.varname = hatvalues(mod1)
```

Note: The command `hatvalues()` operates on any `lm` object and generates a vector of leverage values.

6.3.5 Cook's distance

Example: 6.6.2

Cook's distance (D) is a function of the leverage (see 6.3.4) and the magnitude of the residual. It is used as a measure of the influence of a data point in a regression model.

```
mod1 = lm(y ~ x, data=ds)
cookd.varname = cooks.distance(mod1)
```

Note: The command `cooks.distance()` operates on any `lm` object and generates a vector of Cook's distance values.

6.3.6 DFFITs

Example: 6.6.2

DFFITs are a standardized function of the difference between the predicted value for the observation when it is included in the dataset and when (only) it is excluded from the dataset. They are used as an indicator of the observation's influence.

```
mod1 = lm(y ~ x, data=ds)
dffits.varname = dffits(mod1)
```

Note: The command `dffits()` operates on any `lm` object and generates a vector of DFFITS values.

6.3.7 Diagnostic plots

Example: 6.6.4

```
mod1 = lm(y ~ x, data=ds)
par(mfrow=c(2, 2)) # display 2 x 2 matrix of graphs
plot(mod1)
```

Note: The `plot.lm()` function (which is invoked when `plot()` is given a linear regression model as an argument) can generate six plots: (1) a plot of residuals against fitted values, (2) a Scale-Location plot of $\sqrt{(Y_i - \hat{Y}_i)}$ against fitted values, (3) a normal Q-Q plot of the residuals, (4) a plot of Cook's distances (6.3.5) versus row labels, (5) a plot of residuals against leverages (6.3.4), and (6) a plot of Cook's distances against leverage/(1−leverage). The default is to plot the first three and the fifth. The `which` option can be used to specify a different set (see `help(plot.lm)`).

6.3.8 Heteroscedasticity tests

```
library(lmtest)
bptest(y ~ x1 + ... + xk, data=ds)
```

Note: The `bptest()` function in the `lmtest` package performs the Breusch–Pagan test for heteroscedasticity [18]. Other diagnostic tests are available within the package.

6.4 Model parameters and results

6.4.1 Parameter estimates

Example: 6.6.2

```
mod1 = lm(y ~ x, data=ds)
coeff.mod1 = coef(mod1)
```

Note: The first element of the vector `coeff.mod1` is the intercept (assuming that a model with an intercept was fit).

6.4.2 Standardized regression coefficients

Standardized coefficients from a linear regression model are the parameter estimates obtained when the predictors and outcomes have been standardized to have a variance of 1 prior to model fitting.

```
library(QuantPsyc)
mod1 = lm(y ~ x)
lm.beta(mod1)
```

6.4.3 Coefficient plot

Example: 6.6.3

An alternative way to display regression results (coefficients and associated confidence intervals) is with a figure rather than a table [51].

```
library(mosaic)
mplot(mod, which=7)
```

Note: The specific coefficients to be displayed can be specified (or excluded, using negative values) via the `rows` option.

6.4.4 Standard errors of parameter estimates

See 6.4.10 (covariance matrix).

```
mod1 = lm(y ~ x, data=ds)
sqrt(diag(vcov(mod1)))
```

or

```
coef(summary(mod1))[,2]
```

Note: The standard errors are the second column of the results from `coef()`.

6.4.5 Confidence interval for parameter estimates

Example: 6.6.2

```
mod1 = lm(y ~ x, data=ds)
confint(mod1)
```

6.4.6 Confidence limits for the mean

These are the lower (and upper) confidence limits for the mean of observations with the given covariate values, as opposed to the prediction limits for individual observations with those values (see prediction limits, 6.4.7).

```
mod1 = lm(y ~ x, data=ds)
pred = predict(mod1, interval="confidence")
lcl.varname = pred[,2]
```

Note: The lower confidence limits are the second column of the results from `predict()`. To generate the upper confidence limits, the user would access the third column of the `predict()` object. The command `predict()` operates on any `lm()` object, and with these options generates confidence limit values. By default, the function uses the estimation dataset, but a separate dataset of values to be used to predict can be specified. The `panel=panel.lmbands` option from the `mosaic` package can be added to an `xyplot()` call to augment the scatterplot with confidence interval and prediction bands.

6.4.7 Prediction limits

These are the lower (and upper) prediction limits for "new" observations with the covariate values of subjects observed in the dataset, as opposed to confidence limits for the population mean (see confidence limits, 6.4.6).

```
mod1 = lm(y ~ ..., data=ds)
pred.w.lowlim = predict(mod1, interval="prediction")[,2]
```

Note: This code saves the second column of the results from the `predict()` function into a vector. To generate the upper confidence limits, the user would access the third column of the `predict()` object in R. The command `predict()` operates on any `lm()` object, and with these options generates prediction limit values. By default, the function uses the estimation dataset, but a separate dataset of values to be used to predict can be specified.

6.4.8 R-squared

```
mod1 = lm(y ~ ..., data=ds)
summary(mod1)$r.squared
```

or

```
library(mosaic)
rsquared(mod1)
```

6.4.9 Design and information matrix

See 3.3 (matrices).

```
mod1 = lm(y ~ x1 + ... + xk, data=ds)
XpX = t(model.matrix(mod1)) %*% model.matrix(mod1)
```

or

```
X = cbind(rep(1, length(x1)), x1, x2, ..., xk)
XpX = t(X) %*% X
rm(X)
```

Note: The `model.matrix()` function creates the design matrix from a linear model object. Alternatively, this quantity can be built up using the `cbind()` function to glue together the design matrix `X`. Finally, matrix multiplication (3.3.6) and the transpose function are used to create the information ($X'X$) matrix.

6.4.10 Covariance matrix of parameter estimates

Example: 6.6.2

See 3.3 (matrices) and 6.4.4 (standard errors).

```
mod1 = lm(y ~ x, data=ds)
vcov(mod1)
```

or

```
sumvals = summary(mod1)
covb = sumvals$cov.unscaled*sumvals$sigma^2
```

Note: Running `help(summary.lm)` provides details on return values.

6.4.11 Correlation matrix of parameter estimates

See 3.3 (matrices) and 6.4.4 (standard errors).

```
mod1 = lm(y ~ x, data=ds)
mod1.cov = vcov(mod1)
mod1.cor = cov2cor(mod1.cov)
```

Note: The `cov2cor()` function is a convenient way to convert a covariance matrix into a correlation matrix.

6.5 Further resources

An accessible guide to linear regression in R can be found in [36]. Cook [28] reviews regression diagnostics. Frank Harrell's `rms` (regression modeling strategies) package [61] features extensive support for regression modeling. The CRAN statistics for the social sciences task view provides an excellent overview of methods described here and in Chapter 7.

6.6 Examples

To help illustrate the tools presented in this chapter, we apply many of the entries to the HELP data. The code can be downloaded from `http://www.amherst.edu/~nhorton/r2/examples`.

We begin by reading in the dataset and keeping only the female subjects. To prepare for future analyses, we create a version of `substance` as a factor variable (see 6.1.4) as well as dataframes containing subsets of our data.

```
> options(digits=3)
> # read in Stata format
> library(foreign)
> ds = read.dta("http://www.amherst.edu/~nhorton/r2/datasets/help.dta",
    convert.underscore=FALSE)
> library(dplyr)
> ds = mutate(ds, sub=factor(substance,
    levels=c("heroin", "alcohol", "cocaine")))
> newds = filter(ds, female==1)
> alcohol = filter(newds, substance=="alcohol")
> cocaine = filter(newds, substance=="cocaine")
> heroin = filter(newds, substance=="heroin")
```

6.6.1 Scatterplot with smooth fit

As a first step to help guide estimation of a linear regression, we create a scatterplot (8.3.1) displaying the relationship between age and the number of alcoholic drinks consumed in the period before entering detox (variable name: `i1`), as well as primary substance of abuse (alcohol, cocaine, or heroin).

Figure 6.1 displays a scatterplot of observed values for `i1` (along with separate smooth fits by primary substance). To improve legibility, the plotting region is restricted to those with number of drinks between 0 and 40 (see plotting limits, 9.2.9).

```
> with(newds, plot(age, i1, ylim=c(0,40), type="n", cex.lab=1.2,
    cex.axis=1.2))
> with(alcohol, points(age, i1, pch="a"))
> with(alcohol, lines(lowess(age, i1), lty=1, lwd=2))
> with(cocaine, points(age, i1, pch="c"))
> with(cocaine, lines(lowess(age, i1), lty=2, lwd=2))
> with(heroin, points(age, i1, pch="h"))
> with(heroin, lines(lowess(age, i1), lty=3, lwd=2))
> legend(44, 38, legend=c("alcohol", "cocaine", "heroin"), lty=1:3,
    cex=1.4, lwd=2, pch=c("a", "c", "h"))
```

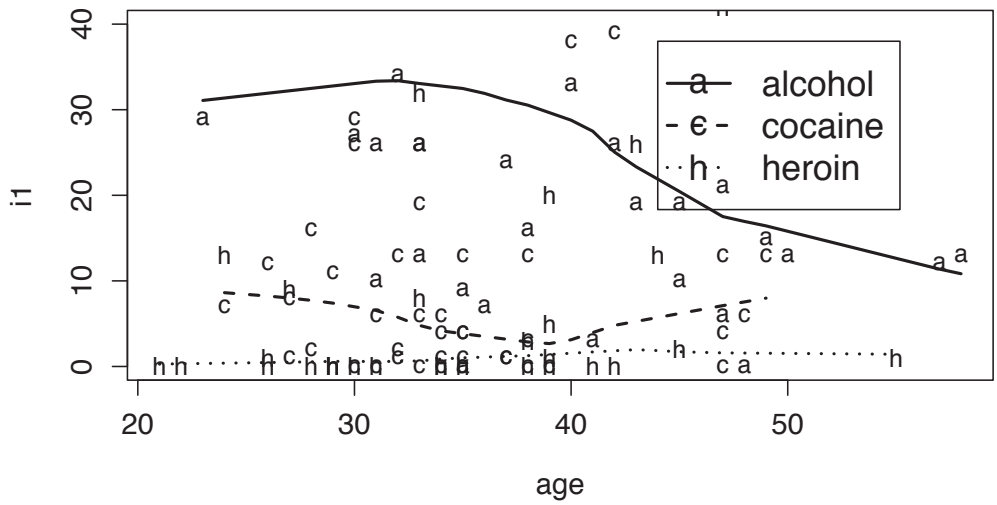

Figure 6.1: Scatterplot of observed values for age and I1 (plus smoothers by substance) using base graphics

The `pch` option to the `legend()` command can be used to insert plot symbols in R legends (Figure 6.1 displays the different line styles). A similar plot can be generated using the `lattice` package (see Figure 6.2). Finally, a third figure can be generated using the `ggplot2` package (see Figure 6.3). Not surprisingly, the plots suggest a dramatic effect of primary substance, with alcohol users drinking more than others. There is some indication of an interaction with age.

6.6.2 Linear regression with interaction

Next we fit a linear regression model (6.1.1) for the number of drinks as a function of age, substance, and their interaction (6.1.6). We also fit the model with no interaction and use the `anova()` function to compare the models (the `drop1()` function could also be used).

```
> options(show.signif.stars=FALSE)
> lm1 = lm(i1 ~ sub * age, data=newds)
> lm2 = lm(i1 ~ sub + age, data=newds)
```

```
> xyplot(i1 ~ age, groups=substance, type=c("p", "smooth"),
    auto.key=list(columns=3, lines=TRUE, points=FALSE),
    ylim=c(0, 40), data=newds)
```

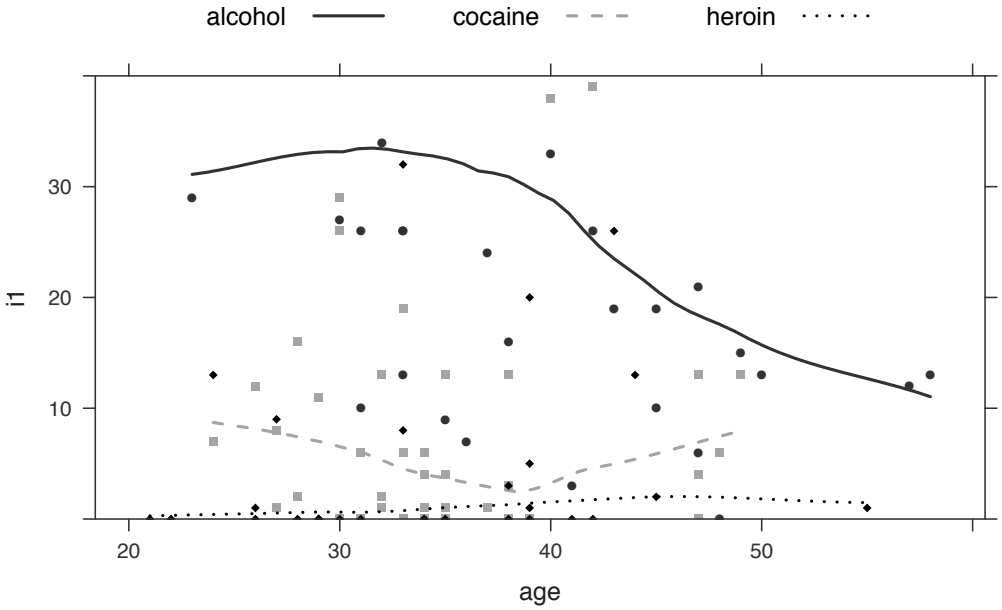

Figure 6.2: Scatterplot of observed values for age and I1 (plus smoothers by substance) using the `lattice` package

```
> anova(lm2, lm1)
Analysis of Variance Table

Model 1: i1 ~ sub + age
Model 2: i1 ~ sub * age
  Res.Df   RSS Df Sum of Sq    F Pr(>F)
1    103 26196
2    101 24815  2      1381 2.81  0.065
```

```
> summary.aov(lm1)
             Df Sum Sq Mean Sq F value  Pr(>F)
sub           2  10810    5405   22.00 1.2e-08
age           1     84      84    0.34   0.559
sub:age       2   1381     690    2.81   0.065
Residuals   101  24815     246
```

We observe a borderline significant interaction between age and substance group ($p = 0.065$). Additional information about the model can be displayed using the `summary()` and `confint()` functions.

```
> library(ggplot2)
> ggplot(data=newds, aes(x=age, y=i1)) + geom_point(aes(shape=substance)) +
    stat_smooth(method=loess, level=0.50, colour="black") +
    aes(linetype=substance) +
    coord_cartesian(ylim = c(0, 40)) +
    theme(legend.position="top") + labs(title="")
```

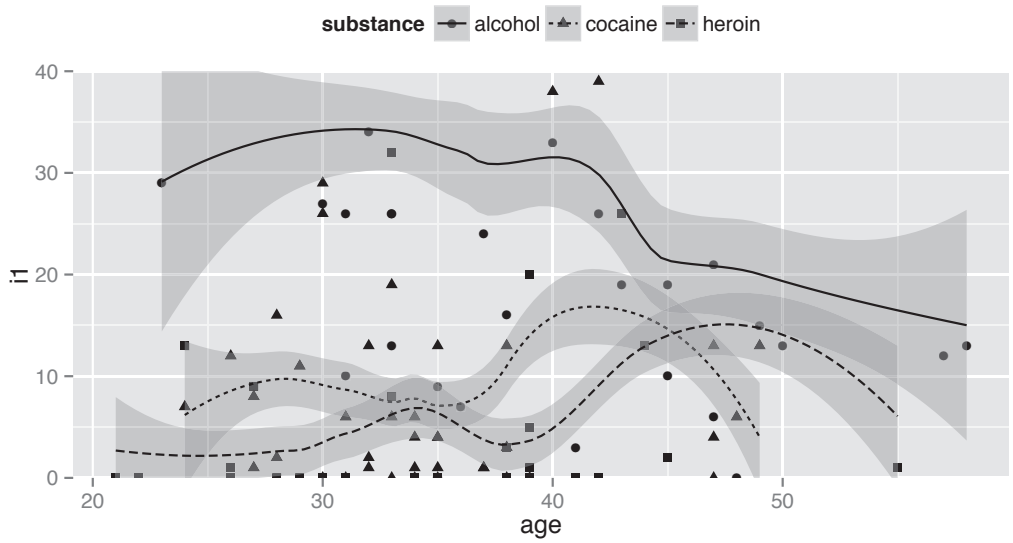

Figure 6.3: Scatterplot of observed values for age and I1 (plus smoothers by substance) using the `ggplot2` package

```
> summary(lm1)

Call:
lm(formula = i1 ~ sub * age, data = newds)

Residuals:
   Min     1Q Median     3Q    Max
-31.92  -8.25  -4.18   3.58  49.88

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      -7.770     12.879   -0.60  0.54763
subalcohol       64.880     18.487    3.51  0.00067
subcocaine       13.027     19.139    0.68  0.49763
age               0.393      0.362    1.09  0.28005
subalcohol:age   -1.113      0.491   -2.27  0.02561
subcocaine:age   -0.278      0.540   -0.51  0.60813
```

```
Residual standard error: 15.7 on 101 degrees of freedom
Multiple R-squared:  0.331,Adjusted R-squared:  0.298
F-statistic: 9.99 on 5 and 101 DF,  p-value: 8.67e-08
```

```
> confint(lm1)
                 2.5 %  97.5 %
(Intercept)    -33.319  17.778
subalcohol      28.207 101.554
subcocaine     -24.938  50.993
age             -0.325   1.112
subalcohol:age  -2.088  -0.138
subcocaine:age  -1.348   0.793
```

It may also be useful to produce the table in LaTeX format. We can use the `xtable` package to display the regression results in LaTeX as shown in Table 6.1.

```
> library(xtable)
> lmtab = xtable(lm1, digits=c(0,3,3,2,4), label="better",
>    caption="Formatted results using the {\\tt xtable} package")
> print(lmtab)  # output the LaTeX
```

Table 6.1: Formatted results using the `xtable` package

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-7.770	12.879	-0.60	0.5476
subalcohol	64.880	18.487	3.51	0.0007
subcocaine	13.027	19.139	0.68	0.4976
age	0.393	0.362	1.09	0.2801
subalcohol:age	-1.113	0.491	-2.27	0.0256
subcocaine:age	-0.278	0.540	-0.51	0.6081

There are many quantities of interest stored in the linear model object `lm1`, and these can be viewed or extracted for further use.

```
> names(summary(lm1))
 [1] "call"          "terms"         "residuals"     "coefficients"
 [5] "aliased"       "sigma"         "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
> summary(lm1)$sigma
[1] 15.7
```

```
> names(lm1)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

```
> coef(lm1)
   (Intercept)     subalcohol     subcocaine            age subalcohol:age
        -7.770         64.880         13.027          0.393         -1.113
subcocaine:age
        -0.278
```

```
> vcov(lm1)
               (Intercept) subalcohol subcocaine    age subalcohol:age
(Intercept)         165.86    -165.86    -165.86 -4.548          4.548
subalcohol         -165.86     341.78     165.86  4.548         -8.866
subcocaine         -165.86     165.86     366.28  4.548         -4.548
age                  -4.55       4.55       4.55  0.131         -0.131
subalcohol:age        4.55      -8.87      -4.55 -0.131          0.241
subcocaine:age        4.55      -4.55     -10.13 -0.131          0.131
               subcocaine:age
(Intercept)             4.548
subalcohol             -4.548
subcocaine            -10.127
age                    -0.131
subalcohol:age          0.131
subcocaine:age          0.291
```

The entire table of regression coefficients and associated statistics can be saved as an object.

```
> mymodel = coef(summary(lm1))
> mymodel
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      -7.770     12.879  -0.603 0.547629
subalcohol       64.880     18.487   3.509 0.000672
subcocaine       13.027     19.139   0.681 0.497627
age               0.393      0.362   1.086 0.280052
subalcohol:age   -1.113      0.491  -2.266 0.025611
subcocaine:age   -0.278      0.540  -0.514 0.608128
> mymodel[2,3]   # alcohol t-value
[1] 3.51
```

6.6.3 Regression coefficient plot

The `mplot()` function in the `mosaic` package generates a coefficient plot (6.4.3) for the main effects multiple regression model (see Figure 6.4).

6.6.4 Regression diagnostics

Assessing the model is an important part of any analysis. We begin by examining the residuals (6.3.2). First, we calculate the quantiles of their distribution (5.1.5), then display the smallest residual.

```
> library(mosaic)
> mplot(lm2, which=7, rows=-1)
[[1]]
```

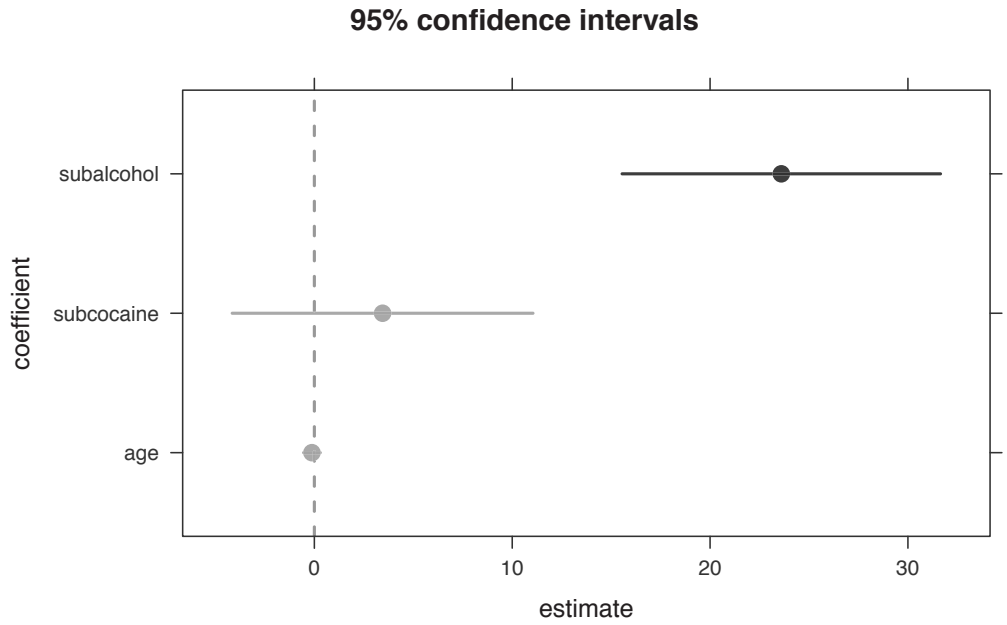

Figure 6.4: Regression coefficient plot

```
> library(dplyr)
> newds = mutate(newds, pred = fitted(lm1), resid = residuals(lm1))
> with(newds, quantile(resid))
    0%    25%    50%    75%   100%
-31.92  -8.25  -4.18   3.58  49.88
```

One way to print the largest value is to select the observation that matches the largest value. We use a series of "pipe" operations (A.5.3) to select a set of variables with the `select()` function, create the standardized residuals and add them to the dataset with the `rstandard()` function nested in the `mutate()` function, and then `filter()` out all rows except the one containing the maximum residual.

```
> library(dplyr)
> newds %>%
    select(id, age, i1, sub, pred, resid) %>%
    mutate(rstand = rstandard(lm1)) %>%
    filter(resid==max(resid))
  id age i1     sub pred resid rstand
1  9  50 71 alcohol 21.1  49.9   3.32
```

Graphical tools are one of the best ways to examine residuals. Figure 6.5 displays the default diagnostic plots (6.3) from the model.

Figure 6.6 displays the empirical density of the standardized residuals, along with an

```
> oldpar = par(mfrow=c(2, 2), mar=c(4, 4, 2, 2) + .1)
> plot(lm1); par(oldpar)
```

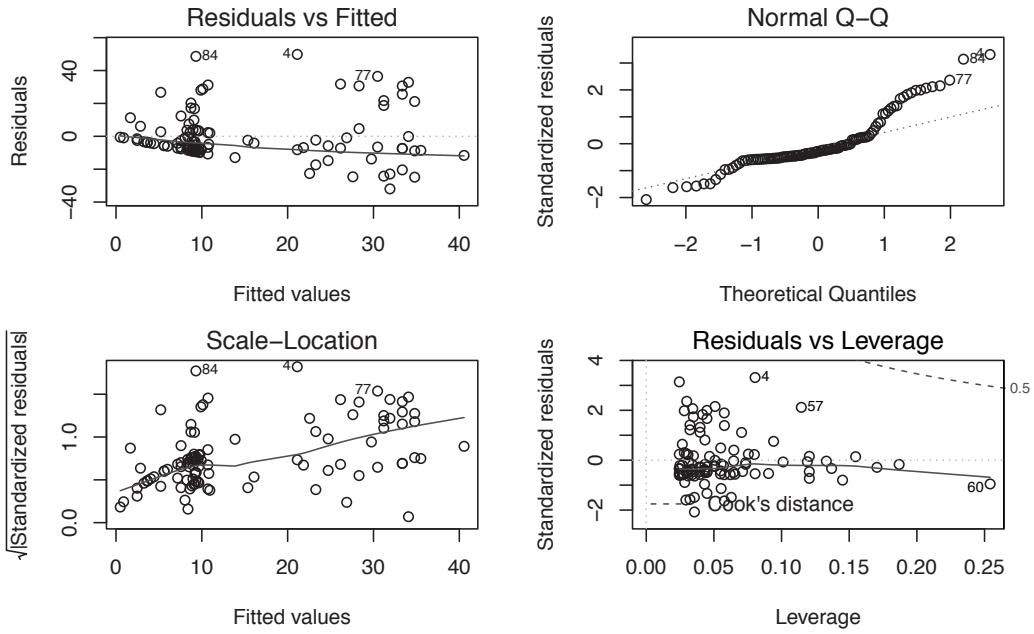

Figure 6.5: Default diagnostics for linear models

overlaid normal density. The assumption that the residuals are approximately Gaussian does not appear to be tenable.

The residual plots also indicate some potentially important departures from model assumptions: further exploration and model assessment should be undertaken.

6.6.5 Fitting a regression model separately for each value of another variable

One common task is to perform identical analyses in several groups. Here, as an example, we consider separate linear regressions for each substance abuse group.

A matrix of the correct size is created, then a `for` loop is run for each unique value of the grouping variable.

```
> uniquevals = unique(newds$substance)
> numunique = length(uniquevals)
> formula = as.formula(i1 ~ age)
> p = length(coef(lm(formula, data=newds)))
> res = matrix(rep(0, numunique*p), p, numunique)
> for (i in 1:length(uniquevals)) {
     res[,i] = coef(lm(formula,
        data=subset(newds, substance==uniquevals[i])))
  }
> rownames(res) = c("intercept","slope")
> colnames(res) = uniquevals
```

```
> library(MASS)
> std.res = rstandard(lm1)
> hist(std.res, breaks=seq(-2.5, 3.5, by=.5), main="",
     xlab="standardized residuals", col="gray80", freq=FALSE)
> lines(density(std.res), lwd=2)
> xvals = seq(from=min(std.res), to=max(std.res), length=100)
> lines(xvals, dnorm(xvals, mean(std.res), sd(std.res)), lty=2)
```

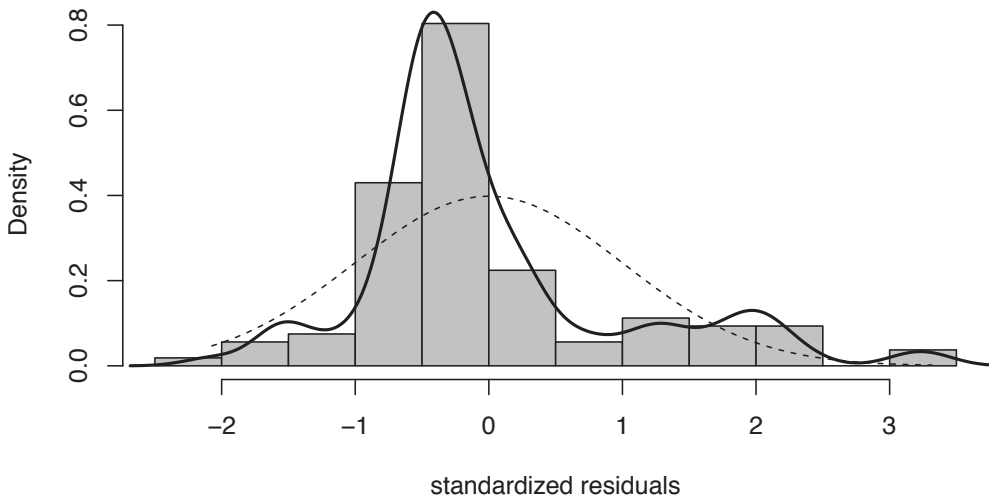

Figure 6.6: Empirical density of residuals, with superimposed normal density

```
> res
          heroin cocaine alcohol
intercept -7.770   5.257   57.11
slope      0.393   0.116   -0.72
```

6.6.6 Two-way ANOVA

Is there a statistically significant association between gender and substance abuse group with depressive symptoms? An interaction plot (8.5.2) may be helpful in making a determination. The `interaction.plot()` function can be used to carry out this task. Figure 6.7 displays an interaction plot for CESD as a function of substance group and gender.

```
> library(dplyr)
> ds = mutate(ds, genf = as.factor(ifelse(female, "F", "M")))
```

There are indications of large effects of gender and substance group, but little suggestion of interaction between the two. The same conclusion is reached in Figure 6.8, which displays boxplots by substance group and gender. We begin by creating better labels for the grouping variable, using the `cases()` function from the `memisc` package.

```
> with(ds, interaction.plot(substance, genf, cesd,
    xlab="substance", las=1, lwd=2))
```

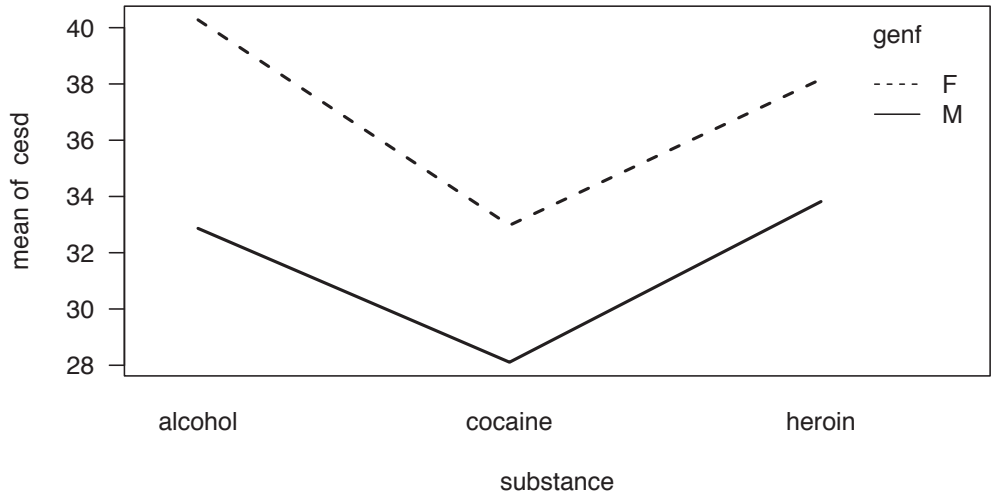

Figure 6.7: Interaction plot of CESD as a function of substance group and gender

```
> library(dplyr)
> library(memisc)
> ds = mutate(ds, subs = cases(
    "Alc" = substance=="alcohol",
    "Coc" = substance=="cocaine",
    "Her" = substance=="heroin"))
```

The width of each box is proportional to the size of the sample, with the notches denoting confidence intervals for the medians and X's marking the observed means. Next, we proceed to formally test whether there is a significant interaction through a two-way analysis of variance (6.1.9). We fit models with and without an interaction, and then compare the results. We also construct the likelihood ratio test manually.

```
> aov1 = aov(cesd ~ sub * genf, data=ds)
> aov2 = aov(cesd ~ sub + genf, data=ds)
> anova(aov2, aov1)
Analysis of Variance Table

Model 1: cesd ~ sub + genf
Model 2: cesd ~ sub * genf
  Res.Df   RSS Df Sum of Sq   F Pr(>F)
1    449 65515
2    447 65369  2       146 0.5   0.61
```

```
> boxout = with(ds,
     boxplot(cesd ~ subs + genf, notch=TRUE, varwidth=TRUE,
        col="gray80"))
> boxmeans = with(ds, tapply(cesd, list(subs, genf), mean))
> points(seq(boxout$n), boxmeans, pch=4, cex=2)
```

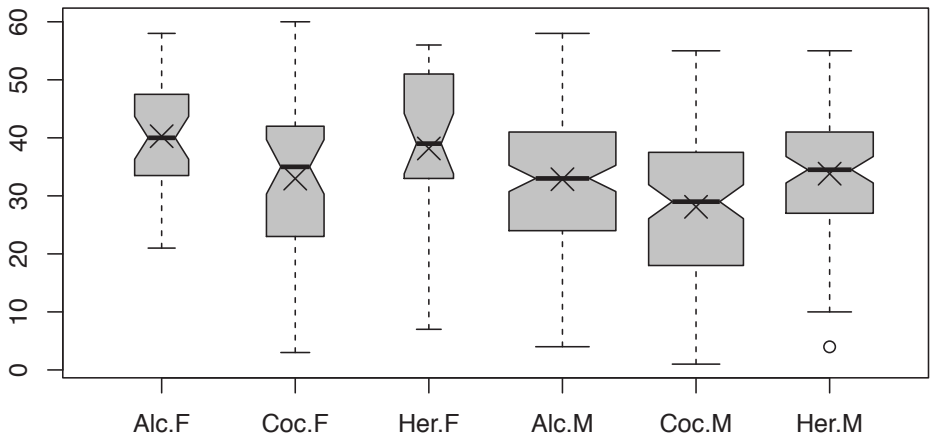

Figure 6.8: Boxplot of CESD as a function of substance group and gender

```
> options(digits=8)
> logLik(aov1)
'log Lik.' -1768.9186 (df=7)
> logLik(aov2)
'log Lik.' -1769.4236 (df=5)
> lldiff = logLik(aov1)[1] - logLik(aov2)[1]
> lldiff
[1] 0.50505522
> 1 - pchisq(2*lldiff, df=2)
[1] 0.60347225
> options(digits=3)
```

There is little evidence ($p > 0.6$) of an interaction, so this term can be dropped.

```
> summary(aov2)
             Df Sum Sq Mean Sq F value  Pr(>F)
sub           2   2704    1352    9.27 0.00011
genf          1   2569    2569   17.61 3.3e-05
Residuals   449  65515     146
```

The AIC (Akaike Information Criterion) statistic (7.8.3) can also be used to compare models.

```
> AIC(aov1)
[1] 3552
> AIC(aov2)
[1] 3549
```

The AIC criterion also suggests that the model without the interaction is most appropriate.

It may be useful to change the default reference level for variables. The default R design matrix (see 6.1.4) can be changed and the model re-fit.

```
> contrasts(ds$sub) = contr.SAS(3)
> aov3 = lm(cesd ~ sub + genf, data=ds)
> summary(aov3)

Call:
lm(formula = cesd ~ sub + genf, data = ds)

Residuals:
   Min     1Q Median     3Q    Max
-32.13  -8.85   1.09   8.48  27.09

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    33.52       1.38   24.22  < 2e-16
sub1            5.61       1.46    3.83  0.00014
sub2            5.32       1.34    3.98  8.1e-05
genfM          -5.62       1.34   -4.20  3.3e-05

Residual standard error: 12.1 on 449 degrees of freedom
Multiple R-squared:  0.0745,Adjusted R-squared:  0.0683
F-statistic:   12 on 3 and 449 DF,  p-value: 1.35e-07
```

6.6.7 Multiple comparisons

We can also carry out multiple comparison (6.2.4) procedures to test each of the pairwise differences between substance abuse groups, using the `TukeyHSD()` function.

```
> mult = TukeyHSD(aov(cesd ~ sub, data=ds), "sub")
> mult
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = cesd ~ sub, data = ds)

$sub
                  diff   lwr   upr p adj
alcohol-heroin  -0.498 -3.89  2.89 0.936
cocaine-heroin  -5.450 -8.95 -1.95 0.001
cocaine-alcohol -4.952 -8.15 -1.75 0.001
```

The alcohol group and heroin group both have significantly higher CESD scores than the cocaine group, but the alcohol and heroin groups do not significantly differ from each other (95% confidence interval (CI) for the difference ranges from -3.9 to 2.9). Figure 6.9 provides a graphical display of the pairwise comparisons.

The `factorplot()` function in the `factorplot` package provides an alternative plotting scheme. This is demonstrated using a model where the CESD scores are grouped into six categories.

```
> require(mosaic)
> mplot(mult)
```

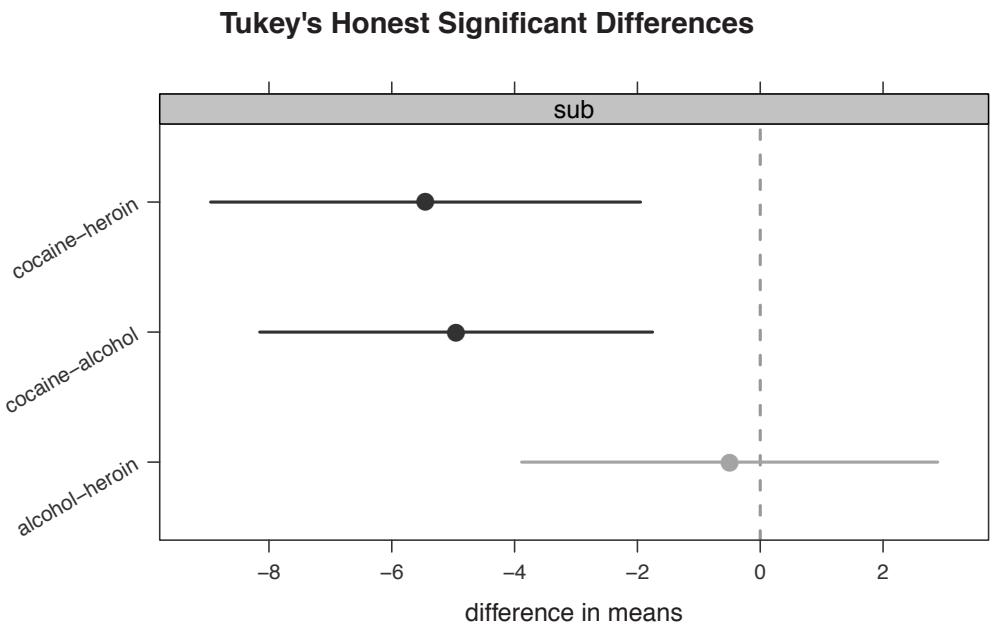

Figure 6.9: Pairwise comparisons (using Tukey HSD procedure)

```
> library(dplyr)
> library(factorplot)
> newds = mutate(newds, cesdgrp = cut(cesd,
    breaks=c(-1, 10, 20, 30, 40, 50, 61),
    labels=c("0-10", "11-20", "21-30", "31-40", "41-50", "51-60")))
> tally(~ cesdgrp, data=newds)

 0-10 11-20 21-30 31-40 41-50 51-60
    4    10    18    31    24    20
> mod = lm(pcs ~ age + cesdgrp, data=newds)
> fp = factorplot(mod, adjust.method="none", factor.variable="cesdgrp",
    pval=0.05, two.sided=TRUE, order="natural")
```

Figure 6.10 provides a graphical display of the fifteen pairwise comparisons, where the pairwise difference is displayed above the standard error of that difference (in italics).

6.6.8 Contrasts

We can also fit contrasts (6.2.3) to test hypotheses involving multiple parameters. In this case, we can compare the CESD scores for the alcohol and heroin groups to the cocaine group.

```
> plot(fp, abbrev.char=100)
```

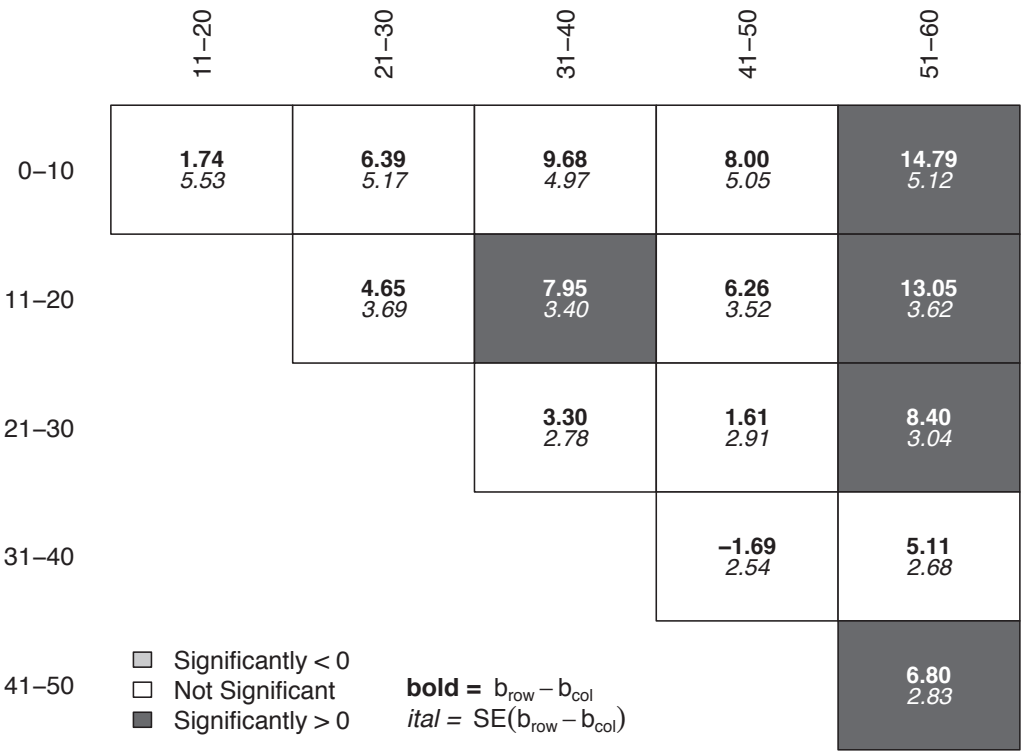

Figure 6.10: Pairwise comparisons (using the factorplot function)

```
> library(gmodels)
> levels(ds$sub)
[1] "heroin"  "alcohol" "cocaine"
> fit.contrast(aov2, "sub", c(1,1,-2), conf.int=0.95 )
                Estimate Std. Error t value Pr(>|t|) lower CI upper CI
sub c=( 1 1 -2 )     10.9       2.42    4.52 8.04e-06     6.17     15.7
```

As expected from the interaction plot (Figure 6.7), there is a statistically significant difference in this 1-degree-of-freedom comparison ($p < 0.0001$).