# Multiple Linear Regression

A multiple linear regression (MLR) model that describes a dependent variable y by independent variables $x_1$, $x_2$, ..., $x_p$ ($p > 1$) is expressed by the equation as follows, where the numbers α and $\beta_k$ (k = 1, 2, ..., p) are the parameters, and ϵ is the error term.

$$y = \alpha + \sum_{\kappa} \beta_{\kappa} x_{\kappa} + \epsilon$$

For example, in the built-in data set stackloss from observations of a chemical plant operation, if we assign stackloss as the dependent variable, and assign Air.Flow (cooling air flow), Water.Temp (inlet water temperature) and Acid.Conc. (acid concentration) as independent variables, the multiple linear regression model is:

$$Stack.Loss = \alpha + \beta_1 * Air.Flow + \beta_2 * Water.Temp + \beta_3 * Acid.Conc. + \epsilon$$

Further detail of the stackloss data set can be found in the R documentation.

```
> help(stackloss)
```

## Estimated Multiple Regression Equation

If we choose the parameters α and $\beta_k$ (k = 1, 2, ..., p) in the multiple linear regression model so as to minimize the sum of squares of the error term ϵ, we will have the so called estimated multiple regression equation.

It allows us to compute fitted values of y based on a set of values of $x_k$ (k = 1, 2, ..., p) .

$$\hat{y} = a + \sum_{\kappa} b_{\kappa} x_{\kappa}$$

**Problem**

Apply the multiple linear regression model for the data set stackloss, and predict the stack loss if the air flow is 72, water temperature is 20 and acid concentration is 85.

**Solution**

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc. And we save the linear regression model in a new variable stackloss.lm.

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.,
+     data=stackloss)
```

We also wrap the parameters inside a new data frame named newdata.

```
> newdata = data.frame(Air.Flow=72,  # wrap the parameters
+     Water.Temp=20,
+     Acid.Conc.=85)
```

Lastly, we apply the predict function to stackloss.lm and newdata.

```
> predict(stackloss.lm, newdata)
     1
24.582
```

**Answer**

Based on the multiple linear regression model and the given parameters, the predicted stack loss is 24.582.

# Multiple Coefficient of Determination

The coefficient of determination of a multiple linear regression model is the quotient of the variances of the fitted values and observed values of the dependent variable.

If we denote $y_i$ as the observed values of the dependent variable, $\bar{y}$ as its mean, and $\hat{y}_i$ as the fitted value, then the coefficient of determination is:

$$R^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

**Problem**

Find the coefficient of determination for the multiple linear regression model of the data set stackloss.

**Solution**

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc. And we save the linear regression model in a new variable stackloss.lm.

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.,
+     data=stackloss)
```

Then we extract the coefficient of determination from the r.squared attribute of its summary.

```
> summary(stackloss.lm)$r.squared
[1] 0.91358
```

**Answer**

The coefficient of determination of the multiple linear regression model for the data set stackloss is 0.91358.

**Note**

Further detail of the r.squared attribute can be found in the R documentation.

```
> help(summary.lm)
```

# Adjusted Coefficient of Determination

The adjusted coefficient of determination of a multiple linear regression model is defined in terms of the coefficient of determination as follows, where n is the number of observations in the data set, and p is the number of independent variables.

$$R_{adj}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

**Problem**

Find the adjusted coefficient of determination for the multiple linear regression model of the data set stackloss.

**Solution**

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc. And we save the linear regression model in a new variable stackloss.lm.

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.,
+     data=stackloss)
```

Then we extract the coefficient of determination from the adj.r.squared attribute of its summary.

```
> summary(stackloss.lm)$adj.r.squared
[1]  0.89833
```

**Answer**

The adjusted coefficient of determination of the multiple linear regression model for the data set stackloss is 0.89833.

**Note**

Further detail of the adj.r.squared attribute can be found in the R documentation.

```
> help(summary.lm)
```

# Significance Test for MLR

Assume that the error term ϵ in the multiple linear regression (MLR) model is independent of $x_k$ (k = 1, 2, ..., p), and is normally distributed, with zero mean and constant variance.

We can decide whether there is any significant relationship between the dependent variable y and any of the independent variables $x_k$ (k = 1, 2, ..., p).

### Problem

Decide which of the independent variables in the multiple linear regression model of the data set stackloss are statistically significant at .05 significance level.

### Solution

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc. And we save the linear regression model in a new variable stackloss.lm.

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.,
+     data=stackloss)
```

The t values of the independent variables can be found with the summary function.

```
> summary(stackloss.lm)

Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
    data = stackloss)


Residuals:
   Min     1Q Median     3Q    Max
-7.238 -1.712 -0.455  2.361  5.698
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -39.920     11.896   -3.36   0.0038 **
Air.Flow       0.716      0.135    5.31  5.8e-05 ***
Water.Temp     1.295      0.368    3.52   0.0026 **
Acid.Conc.    -0.152      0.156   -0.97   0.3440
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 3.24 on 17 degrees of freedom
Multiple R-squared: 0.914,      Adjusted R-squared: 0.898
F-statistic: 59.9 on 3 and 17 DF,  p-value: 3.02e-09
```

**Answer**

As the p-values of Air.Flow and Water.Temp are less than 0.05, they are both statistically significant in the multiple linear regression model of stackloss.

**Note**

Further detail of the summary function for linear regression model can be found in the R documentation.

```
> help(summary.lm)
```

# Confidence Interval for MLR

Assume that the error term ϵ in the multiple linear regression (MLR) model is independent of $x_k$ (k = 1, 2, ..., p), and is normally distributed, with zero mean and constant variance.

For a given set of values of $x_k$ (k = 1, 2, ..., p), the interval estimate for the mean of the dependent variable, $\bar{y}$ , is called the confidence interval.

**Problem**

In data set stackloss, develop a 95% confidence interval of the stack loss if the air flow is 72, water temperature is 20 and acid concentration is 85.

**Solution**

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc.

And we save the linear regression model in a new variable stackloss.lm.

```
> attach(stackloss)    # attach the data frame
```

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.)
```

Then we wrap the parameters inside a new data frame variable newdata.

```
> newdata = data.frame(Air.Flow=72,
+     Water.Temp=20,
+     Acid.Conc.=85)
```

We now apply the predict function and set the predictor variable in the newdata argument.

We also set the interval type as "confidence", and use the default 0.95 confidence level.

```
> predict(stackloss.lm, newdata, interval="confidence")
     fit    lwr    upr
1 24.582 20.218 28.945

> detach(stackloss)    # clean up
```

**Answer**

The 95% confidence interval of the stack loss with the given parameters is between 20.218 and 28.945.

**Note**

Further detail of the predict function for linear regression model can be found in the R documentation.

```
> help(predict.lm)
```

# Prediction Interval for MLR

Assume that the error term $\epsilon$ in the multiple linear regression (MLR) model is independent of $x_k$ ($k = 1, 2, ..., p$), and is normally distributed, with zero mean and constant variance.

For a given set of values of $x_k$ ($k = 1, 2, ..., p$), the interval estimate of the dependent variable y is called the prediction interval.

**Problem**

In data set stackloss, develop a 95% prediction interval of the stack loss if the air flow is 72, water temperature is 20 and acid concentration is 85.

**Solution**

We apply the lm function to a formula that describes the variable stack.loss by the variables Air.Flow, Water.Temp and Acid.Conc.

And we save the linear regression model in a new variable stackloss.lm.

```
> attach(stackloss)    # attach the data frame
```

```
> stackloss.lm = lm(stack.loss ~
+     Air.Flow + Water.Temp + Acid.Conc.)
```

Then we wrap the parameters inside a new data frame variable newdata.

```
> newdata = data.frame(Air.Flow=72,
+     Water.Temp=20,
+     Acid.Conc.=85)
```

We now apply the predict function and set the predictor variable in the newdata argument.

We also set the interval type as "predict", and use the default 0.95 confidence level.

```
> predict(stackloss.lm, newdata, interval="predict")
     fit    lwr    upr
1 24.582 16.466 32.697
```

```
> detach(stackloss)    # clean up
```

**Answer**

The 95% confidence interval of the stack loss with the given parameters is between 16.466 and 32.697.

**Note**

Further detail of the predict function for linear regression model can be found in the R documentation.

```
> help(predict.lm)
```

**Exercise**

Free-format data are text files containing numbers or character strings separated by spaces. Optionally the file may have a header containing variable names. Here's an example of a data file containing information on three variables for 20 countries in Latin America:

| | **setting** | **effort** | **change** |
|---|---|---|---|
| Bolivia | 46 | 0 | 1 |
| Brazil | 74 | 0 | 10 |
| Chile | 89 | 16 | 29 |
| Colombia | 77 | 16 | 25 |
| CostaRica | 84 | 21 | 29 |
| Cuba | 89 | 15 | 40 |
| DominicanRep | 68 | 14 | 21 |
| Ecuador | 70 | 6 | 0 |
| ElSalvador | 60 | 13 | 13 |
| Guatemala | 55 | 9 | 4 |
| Haiti | 35 | 3 | 0 |
| Honduras | 51 | 7 | 7 |
| Jamaica | 87 | 23 | 21 |
| Mexico | 83 | 4 | 9 |
| Nicaragua | 68 | 0 | 7 |
| Panama | 84 | 19 | 22 |
| Paraguay | 74 | 3 | 6 |
| Peru | 73 | 0 | 2 |
| TrinidadTobago | 84 | 15 | 29 |
| Venezuela | 91 | 7 | 11 |

This small dataset includes an index of social setting, an index of family planning effort, and the percent decline in the crude birth rate between 1965 and 1975. The data are available on the web at http://data.princeton.edu/wws509/datasets/ in a file called `effort.dat` which includes a header with the variable names.

R can read the data directly from the web:

```
> fpe <- read.table("http://data.princeton.edu/wws509/datasets/effort.dat",
header=T)

> attach(fpe)
```

## Fitting a Model

To fit an ordinary linear model with fertility change as the response and setting and effort as predictors, try

```
> lmfit = lm( change ~ setting + effort )
```

Note first that `lm` is a function, and we assign the result to an object that we choose to call `lmfit` (for linear model fit). This stores the results of the fit for later examination.

The argument to `lm` is a *model formula*, which has the response on the left of the tilde ~ (read "is modeled as") and a Wilkinson-Rogers model specification formula on the right. R uses

| | |
|---|---|
| **+** | to combine elementary terms, as in A+B |
| **:** | for interactions, as in A:B; |
| ***** | for both main effects and interactions, so A*B = A+B+A:B |

A nice feature of R is that it lets you create interactions between categorical variables, between categorical and continuous variables, and even between numeric variables (it just creates the cross-product).

## Examining a Fit

Let us look at the results of the fit. One thing you can do with `lmfit`, as you can with any R object, is print it.

```
> lmfit

Call:
lm(formula = change ~ setting + effort)

Coefficients:
(Intercept)      setting       effort
   -14.4511       0.2706       0.9677
```

The output includes the model formula and the coefficients. You can get a bit more detail by requesting a summary:

```
> summary(lmfit)

Call:
lm(formula = change ~ setting + effort)

Residuals:
     Min       1Q   Median       3Q      Max
-10.3475  -3.6426   0.6384   3.2250  15.8530

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -14.4511     7.0938  -2.037 0.057516 .
setting       0.2706     0.1079   2.507 0.022629 *
effort        0.9677     0.2250   4.301 0.000484 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.389 on 17 degrees of freedom
Multiple R-Squared: 0.7381,     Adjusted R-squared: 0.7073
F-statistic: 23.96 on 2 and 17 DF,  p-value: 1.132e-05
```

The output includes a more conventional table with parameter estimates and standard errors, as well the residual standard error and multiple R-squared. (By default S-Plus includes the matrix of correlations among parameter estimates, which is often bulky, while R sensibly omits it. If you really need it, add the option `correlation=TRUE` to the call to `summary`.)

To get a hierarchical analysis of variance table corresponding to introducing each of the terms in the model one at a time, in the same order as in the model formula, try the `anova` function:

```
> anova(lmfit)
Analysis of Variance Table

Response: change
          Df  Sum Sq Mean Sq F value    Pr(>F)
setting    1 1201.08 1201.08  29.421 4.557e-05 ***
effort     1  755.12  755.12  18.497 0.0004841 ***
Residuals 17  694.01   40.82
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1
```

Alternatively, you can plot the results using

```
> plot(lmfit)
```

This will produce a set of four plots: residuals versus fitted values, a Q-Q plot of standardized residuals, a scale-location plot (square roots of standardized residuals versus fitted values, and a plot of residuals versus leverage that adds bands corresponding to Cook's distances of 0.5 and 1.

R will prompt you to click on the graph window or press Enter before showing each plot, but we can do better. Type `par(mfrow=c(2,2))` to set your graphics window to show four plots at once, in a layout with 2 rows and 2 columns. Then redo the graph using `plot(lmfit)`. To go back to a single graph per window use `par(mfrow=c(1,1))`. There are many other ways to customize your graphs by setting high-level parameters, type `?par` to learn more.

*Technical Note:* You may have noticed that we have used the function `plot` with all kinds of arguments: one or two variables, a data frame, and now a linear model fit. In R jargon plot is a generic function. It checks for the kind of object that you are plotting and then calls the appropriate (more specialized) function to do the work. There are actually many plot functions in R, including `plot.data.frame` and `plot.lm`. For most purposes the generic function will do the right thing and you don't need to be concerned about its inner workings.

## Extracting Results

There are some specialized functions that allow you to extract elements from a linear model fit. For example

```
> fitted(lmfit)
        1         2         3         4         5         6         7         8
-2.004026  5.572452 25.114699 21.867637 28.600325 24.146986 17.496913 10.296380
        ... output edited ...
```

extracts the fitted values. In this case it will also print them, because we did not asign them to anything. (The longer form `fitted.values` is an alias.)

To extract the coefficients use the `coef` function (or the longer form `coefficients`)

```
> coef(lmfit)
(Intercept)     setting      effort
-14.4510978   0.2705885   0.9677137
```

To get the residuals, use the `residuals` function (or the abbreviation `resid`):

```
> residuals(lmfit)
         1           2           3           4           5           6
  3.0040262   4.4275478   3.8853007   3.1323628   0.3996747  15.8530144
        ... output edited ...
```

If you are curious to see exactly what a linear model fit produces, try the function

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

which lists the named components of a linear fit. All of these objects may be extracted using the `$` operator. However, whenever there is a special extractor function you are encouraged to use it.

## Factors and Covariates

So far our predictors have been continuous variables or *covariates*. We can also use categorical variables or *factors*. Let us group family planning effort into three categories:

```
> effortg = cut(effort, breaks = c(-1, 4, 14, 100),
+    label=c("weak","moderate","strong"))
```

The function `cut` creates a factor or categorical variable. The first argument is an input vector, the second is a vector of breakpoints, and the third is a vector of category labels. Note that there is one more breakpoint than there are categories. All values greater than the *i*-th breakpoint and less than or equal to the *(i+1)*-st breakpoint go into the *i*-th category. Any values below the first breakpoint or above the last one are coded `NA` (a special R code for missing values). If the labels are omitted, R generates a suitable default of the form "a thru b".

Try fitting the analysis of covariance model:

```
> covfit = lm( change ~ setting + effortg )
> covfit

Call:
lm(formula = change ~ setting + effortg)

Coefficients:
    (Intercept)          setting   effortgmoderate    effortgstrong
        -5.9540           0.1693            4.1439          19.4476
```

As you can see, family planning effort has been treated automatically as a factor, and R has generated the necessary dummy variables for moderate and strong programs treating weak as the reference cell.

*Choice of Contrasts:* R codes unordered factors using the reference cell or "treatment contrast" method. The reference cell is always the first category which, depending on how the factor was created, is usually the first in alphabetical order. If you don't like this choice, R provides a special function to re-order levels, check out `help(relevel)`.

S codes unordered factors using the *Helmert* contrasts by default, a choice that is useful in designed experiments because it produces orthogonal comparisons, but has baffled many a new user. Both R and S-Plus code ordered factors using polynomials. To change to the reference cell method for unordered factors use the following call

```
> options(contrasts=c("contr.treatment","contr.poly"))
```

Back on to our analysis of covariance fit. You can obtain a hierarchical anova table for the analysis of covariance model using the `anova` function:

```
> anova(covfit)
Analysis of Variance Table

Response: change
          Df  Sum Sq Mean Sq F value    Pr(>F)    
setting    1 1201.08 1201.08  36.556 1.698e-05 ***
effortg    2  923.43  461.71  14.053 0.0002999 ***
Residuals 16  525.69   32.86                      
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1
```

Type `?anova` to learn more about this function.

## Regression Splines

The real power of R begins to shine when you consider some of the other functions you can include in a model formula. First, you can include mathematical functions, for example `log(setting)` is a perfectly legal term in a model formula. You don't have to create a variable representing the log of setting and then use it, R will create it 'on the fly', so you can type

```
> lm( change ~ log(setting) + effort)
```

If you wanted to use orthogonal polynomials of degree 3 on setting, you could include a term of the form `poly(setting,3)`

You can also get R to calculate a well-conditioned basis for regression splines. First you must load the splines library (this step is not needed in S-Plus):

```
> library(splines)
```

This makes available the function `bs` to generate B-splines. For example the call

```
> setting.bs <- bs(setting, knots = c(66,74,84)) + effort
```

will generate cubic B-splines with interior knots placed at 66, 74 and 84. This basis will use seven degrees of freedom, four corresponding to the constant, linear, quadratic and cubic terms, plus one for each interior knot. Alternatively, you may specify the number of degrees of freedom you are willing to spend on the fit using the parameter `df`. For cubic splines R will choose df-4 interior knots placed at suitable quantiles. You can also control the degree of the spline using the parameter `degree`, the default being cubic.

If you like *natural* cubic splines, you can obtain a well-conditioned basis using the function `ns`, which has exactly the same arguments as `bs` except for degree, which is always three. To fit a natural spline with five degrees of freedom, use the call

```
> setting.ns <- ns(setting, df=5)
```

Natural cubic splines are better behaved than ordinary splines at the extremes of the range. The restrictions mean that you save four degrees of freedom. You will probably want to use two of them to place additional knots at the extremes, but you can still save the other two.

To fit an additive model to fertility change using natural cubic splines on setting and effort with only one interior knot each, placed exactly at the median of each variable, try the following call:

```
> splinefit = lm( change ~ ns(setting, knot=median(setting)) +
+    ns(effort, knot=median(effort)) )
```

Here we used the parameter `knot` to specify where we wanted the knot placed, and the function `median` to calculate the median of setting and effort.

Do you think the linear model was a good fit? Natural cubic splines with exactly one interior knot require the same number of parameters as an ordinary cubic polynomial, but are much better behaved at the extremes.

## Other Options

The `lm` function has several additional parameters that we have not discussed. These include

| | |
|---|---|
| `data` | to specify a dataset, in case it is not attached |
| `subset` | to restrict the analysis to a subset of the data |
| `weights` | to do weighted least squares |

and many others; see `help(lm)` for further details. The `args` function lists the arguments used by any function, in case you forget them. Try `args(lm)`.

The fact that R has powerful matrix manipulation routines means that one can do many of these calculations from first principles. The next couple of lines create a model matrix to represent the constant, setting and effort, and then calculate the OLS estimate of the coefficients as $(X'X)^{-1}X'y$:

```
> X <- cbind(1,effort,setting)
> solve( t(X) %*% X ) %*% t(X) %*% change

               [,1]
  [1,] -14.4510978
  [2,]   0.9677137
  [3,]   0.2705885
```

Compare these results with `coef(lmfit)`.