6

Regression and correlation

The main object of this chapter is to show how to perform basic regression analyses, including plots for model checking and display of confidence and prediction intervals. Furthermore, we describe the related topic of correlation in both its parametric and nonparametric variants.

6.1 Simple linear regression

We consider situations where you want to describe the relation between two variables using linear regression analysis. You may, for instance, be interested in describing `short.velocity` as a function of `blood.glucose`. This section deals only with the very basics, whereas several more complicated issues are postponed until Chapter 12.

The linear regression model is given by

$$y_i = \alpha + \beta x_i + \epsilon_i$$

in which the ϵ_i are assumed independent and $N(0, \sigma^2)$. The nonrandom part of the equation describes the y_i as lying on a straight line. The slope of the line (the *regression coefficient)* is β, the increase per unit change in x. The line intersects the y-axis at the *intercept* α.

The parameters α, β, and σ^2 can be estimated using the *method of least squares*. Find the values of α and β that minimize the sum of squared

P. Dalgaard, *Introductory Statistics with R*,
DOI: 10.1007/978-0-387-79054-1_6,

residuals

$$SS_{res} = \sum_i (y_i - (\alpha + \beta x_i))^2$$

This is not actually done by trial and error. One can find closed-form expressions for the choice of parameters that gives the smallest value of SS_{res}:

$$\hat{\beta} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$
$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

The residual variance is estimated as $SS_{res}/(n-2)$, and the residual standard deviation is of course the square root of that.

The empirical slope and intercept will deviate somewhat from the true values due to sampling variation. If you were to generate several sets of y_i at the same set of x_i, you would observe a distribution of empirical slopes and intercepts. Just as you could calculate the SEM to describe the variability of the empirical mean, it is also possible from a single sample of (x_i, y_i) to calculate the standard error of the computed estimates, s.e.$(\hat{\alpha})$ and s.e.$(\hat{\beta})$. These standard errors can be used to compute confidence intervals for the parameters and tests for whether a parameter has a specific value.

It is usually of prime interest to test the null hypothesis that $\beta = 0$ since that would imply that the line was horizontal and thus that the ys have a distribution that is the same, whatever the value of x. You can compute a t test for that hypothesis simply by dividing the estimate by its standard error

$$t = \frac{\hat{\beta}}{\text{s.e.}(\hat{\beta})}$$

which follows a t distribution on $n-2$ degrees of freedom if the true β is zero. A similar test can be calculated for whether the intercept is zero, but you should be aware that it is often a meaningless hypothesis either because there is no natural reason to believe that the line should go through the origin or because it would involve an extrapolation far outside the range of data.

For the example in this section, we need the data frame `thuesen`, which we attach with

```
> attach(thuesen)
```

For linear regression analysis, the function `lm` (*l*inear *m*odel) is used:

```
> lm(short.velocity~blood.glucose)

Call:
lm(formula = short.velocity ~ blood.glucose)

Coefficients:
  (Intercept)  blood.glucose
      1.09781        0.02196
```

The argument to `lm` is a *model formula* in which the tilde symbol (~) should be read as "described by". This was seen several times earlier, both in connection with boxplots and stripcharts and with the t and Wilcoxon tests.

The `lm` function handles much more complicated models than simple linear regression. There can be many other things besides a dependent and a descriptive variable in a model formula. A multiple linear regression analysis (which we discuss in Chapter 11) of, for example, `y` on `x1`, `x2`, and `x3` is specified as `y ~ x1 + x2 + x3`.

In its raw form, the output of `lm` is very brief. All you see is the estimated intercept (α) and the estimated slope (β). The best-fitting straight line is seen to be `short.velocity` $= 1.098 + 0.0220 \times$ `blood.glucose`, but for instance no tests of significance are given.

The result of `lm` is a *model object*. This is a distinctive concept of the S language (of which R is a dialect). Whereas other statistical systems focus on generating printed output that can be controlled by setting options, you get instead the result of a model fit encapsulated in an object from which the desired quantities can be obtained using *extractor functions*. An `lm` object does in fact contain much more information than you see when it is printed.

A basic extractor function is `summary`:

```
> summary(lm(short.velocity~blood.glucose))

Call:
lm(formula = short.velocity ~ blood.glucose)

Residuals:
     Min       1Q   Median       3Q      Max
-0.40141 -0.14760 -0.02202  0.03001  0.43490

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.09781    0.11748   9.345 6.26e-09 ***
blood.glucose  0.02196    0.01045   2.101   0.0479 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.2167 on 21 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared: 0.1737,     Adjusted R-squared: 0.1343
F-statistic: 4.414 on 1 and 21 DF,  p-value: 0.0479
```

The format above looks more like what other statistical packages would output. The following is a "dissection" of the output:

```
Call:
lm(formula = short.velocity ~ blood.glucose)
```

As in `t.test`, etc., the output starts with something that is essentially a repeat of the function call. This is not very interesting when one has just given it as a command to R, but it is useful if the result is saved in a variable that is printed later.

```
Residuals:
     Min       1Q   Median       3Q      Max
-0.40141 -0.14760 -0.02202  0.03001  0.43490
```

This gives a superficial view of the distribution of the residuals that may be used as a quick check of the distributional assumptions. The average of the residuals is zero by definition, so the median should not be far from zero, and the minimum and maximum should be roughly equal in absolute value. In the example, it can be noticed that the third quartile is remarkably close to zero, but in view of the small number of observations, this is not really something to worry about.

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.09781    0.11748   9.345 6.26e-09 ***
blood.glucose  0.02196    0.01045   2.101   0.0479 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we see the regression coefficient and the intercept again, but this time with accompanying standard errors, t tests, and p-values. The symbols to the right are graphical indicators of the level of significance. The line below the table shows the definition of these indicators; one star means $0.01 < p < 0.05$.

The graphical indicators have been the target of some controversy. Some people like to have the possibility of seeing at a glance whether there is "anything interesting" in an analysis, whereas others feel that the indicators too often correspond to meaningless tests. For instance, the intercept in the analysis above is hardly a meaningful quantity at all, and the three-star significance of it is certainly irrelevant. If you are bothered by the stars, turn them off with `options(show.signif.stars=FALSE)`.

```
Residual standard error: 0.2167 on 21 degrees of freedom
  (1 observation deleted due to missingness)
```

This is the residual variation, an expression of the variation of the observations around the regression line, estimating the model parameter σ. The model is not fitted to the entire data set because one value of `short.velocity` is missing.

```
Multiple R-squared: 0.1737,  Adjusted R-squared: 0.1343
```

The first item above is R^2, which in a simple linear regression may be recognized as the squared Pearson correlation coefficient (see Section 6.4.1); that is, $R^2 = r^2$. The other one is the adjusted R^2; if you multiply it by 100%, it can be interpreted as "% variance reduction" (this can, in fact, become negative).

```
F-statistic: 4.414 on 1 and 21 DF, p-value: 0.0479
```

This is an F test for the hypothesis that the regression coefficient is zero. This test is not really interesting in a simple linear regression analysis since it just duplicates information already given — it becomes more interesting when there is more than one explanatory variable. Notice that it gives the exact same result as the t test for a zero slope. In fact, the F test is identical to the square of the t test: $4.414 = (2.101)^2$. This is true in any model with 1 degree of freedom.

We will see later how to draw residual plots and plots of data with confidence and prediction limits. First, we draw just the points and the fitted line. Figure 6.1 has been constructed as follows:

```
> plot(blood.glucose,short.velocity)
> abline(lm(short.velocity~blood.glucose))
```

`abline`, meaning (a,b)-line, draws lines based on the intercept and slope, a and b, respectively. It can be used with scalar values as in `abline(1.1,0.022)`, but conveniently it can also extract the information from a linear model fitted to data with `lm`.

6.2 Residuals and fitted values

We have seen how `summary` can be used to extract information about the results of a regression analysis. Two further extraction functions are `fitted` and `resid`. They are used as follows. For convenience, we first store the value returned by `lm` under the name `lm.velo` (short for "velocity", but you could of course use any other name).

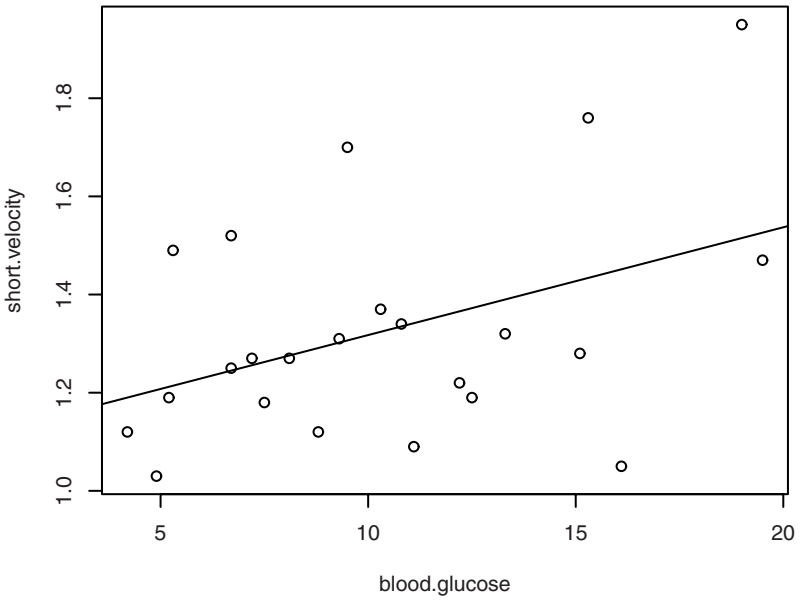

Figure 6.1. Scatterplot with regression line.

```
> lm.velo <- lm(short.velocity~blood.glucose)
> fitted(lm.velo)
       1        2        3        4        5        6        7
1.433841 1.335010 1.275711 1.526084 1.255945 1.214216 1.302066
       8        9       10       11       12       13       14
1.341599 1.262534 1.365758 1.244964 1.212020 1.515103 1.429449
      15       17       18       19       20       21       22
1.244964 1.190057 1.324029 1.372346 1.451411 1.389916 1.205431
      23       24
1.291085 1.306459
> resid(lm.velo)
          1           2           3           4           5
 0.326158532  0.004989882 -0.005711308 -0.056084062  0.014054962
          6           7           8           9          10
 0.275783754  0.007933665 -0.251598875 -0.082533795 -0.145757649
         11          12          13          14          15
 0.005036223 -0.022019994  0.434897199 -0.149448964  0.275036223
         17          18          19          20          21
-0.070057471  0.045971143 -0.182346406 -0.401411486 -0.069916424
         22          23          24
-0.175431237 -0.171085074  0.393541161
```

The function `fitted` returns fitted values — the y-values that you would expect for the given x-values according to the best-fitting straight

line; in the present case, `1.098+0.0220*blood.glucose`. The residuals shown by `resid` is the difference between this and the observed `short.velocity`.

Note that the fitted values and residuals are labelled with the row names of the `thuesen` data frame. Notice in particular that they do not contain observation no. 16, which had a missing value in the response variable.

It is necessary to discuss some awkward aspects that arise when there are missing values in data.

To put the fitted line on the plot, you might, although it is easier to use `abline(lm.velo)`, get the idea of doing it with `lines`, *but*

```
> plot(blood.glucose,short.velocity)
> lines(blood.glucose,fitted(lm.velo))
Error in xy.coords(x, y) : 'x' and 'y' lengths differ
Calls: lines -> lines.default -> plot.xy -> xy.coords
```

which is true. There are 24 observations but only 23 fitted values because one of the `short.velocity` values is `NA`. Notice, incidentally, that the error occurs within a series of nested function calls, which are being listed along with the error message to reduce confusion.

What we needed was `blood.glucose`, but only for those patients whose `short.velocity` has been recorded.

```
> lines(blood.glucose[!is.na(short.velocity)],fitted(lm.velo))
```

Recall that the `is.na` function yields a vector that is `TRUE` wherever the argument is `NA` (missing). One advantage to this method is that the fitted line does not extend beyond the range of data. The technique works but becomes clumsy if there are missing values in several variables:

```
...blood.glucose[!is.na(short.velocity) & !is.na(blood.glucose)]...
```

It becomes easier with the function `complete.cases`, which can find observations that are nonmissing on several variables or across an entire data frame.

```
> cc <- complete.cases(thuesen)
```

We could then attach `thuesen[cc,]` and work on from there. However, there is a better alternative available: You can use the `na.exclude` method for `NA` handling. This can be set either as an argument to `lm` or as an option; that is,

```
> options(na.action=na.exclude)
> lm.velo <- lm(short.velocity~blood.glucose)
```

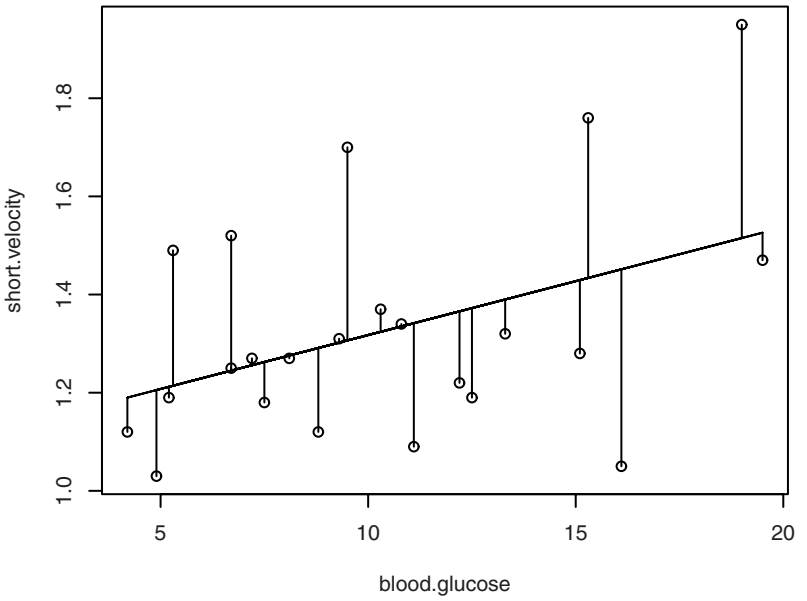

Figure 6.2. Scatterplot of `short.velocity` versus `blood.glucose` with fitted line and residual line segments.

```
> fitted(lm.velo)
       1        2        3        4        5        6        7
1.433841 1.335010 1.275711 1.526084 1.255945 1.214216 1.302066
       8        9       10       11       12       13       14
1.341599 1.262534 1.365758 1.244964 1.212020 1.515103 1.429449
      15       16       17       18       19       20       21
1.244964       NA 1.190057 1.324029 1.372346 1.451411 1.389916
      22       23       24
1.205431 1.291085 1.306459
```

Notice how the missing observation, no. 16, now appears in the fitted values with a missing fitted value. It is necessary to recalculate the `lm.velo` object after changing the option.

To create a plot where residuals are displayed by connecting observations to corresponding points on the fitted line, you can do the following. The final result will look like Figure 6.2. `segments` draws line segments; its arguments are the endpoint coordinates in the order (x_1, y_1, x_2, y_2).

```
> segments(blood.glucose,fitted(lm.velo),
+          blood.glucose,short.velocity)
```

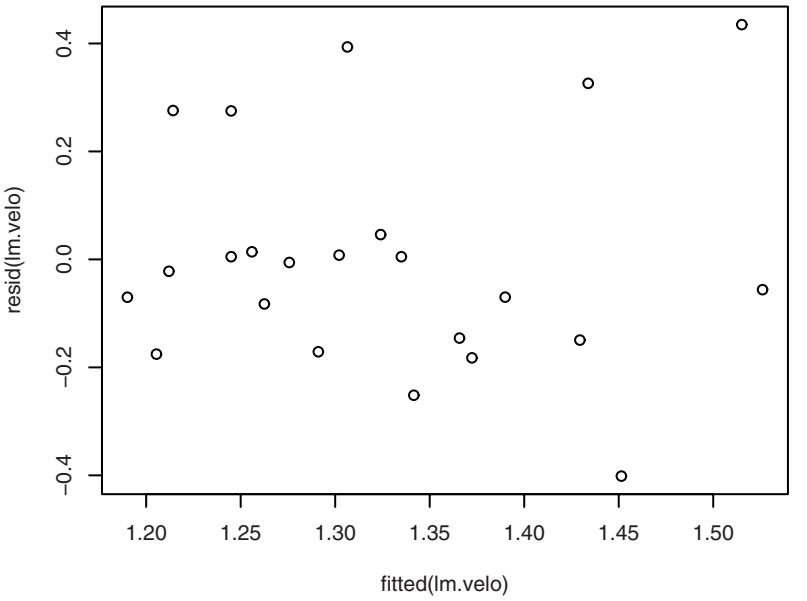

Figure 6.3. `short.velocity` and `blood.glucose`: residuals versus fitted value.

A simple plot of residuals versus fitted values is obtained as (Figure 6.3)

```
> plot(fitted(lm.velo),resid(lm.velo))
```

and we can get an indication of whether residuals might have come from a normal distribution by checking for a straight line on a Q–Q plot (see Section 4.2.3) as follows (Figure 6.4):

```
> qqnorm(resid(lm.velo))
```

6.3 Prediction and confidence bands

Fitted lines are often presented with uncertainty bands around them. There are two kinds of bands, often referred to as the "narrow" and "wide" limits.

The narrow bands, *confidence bands*, reflect the uncertainty about the line itself, like the SEM expresses the precision with which a mean is known. If there are many observations, the bands will be quite narrow, reflecting

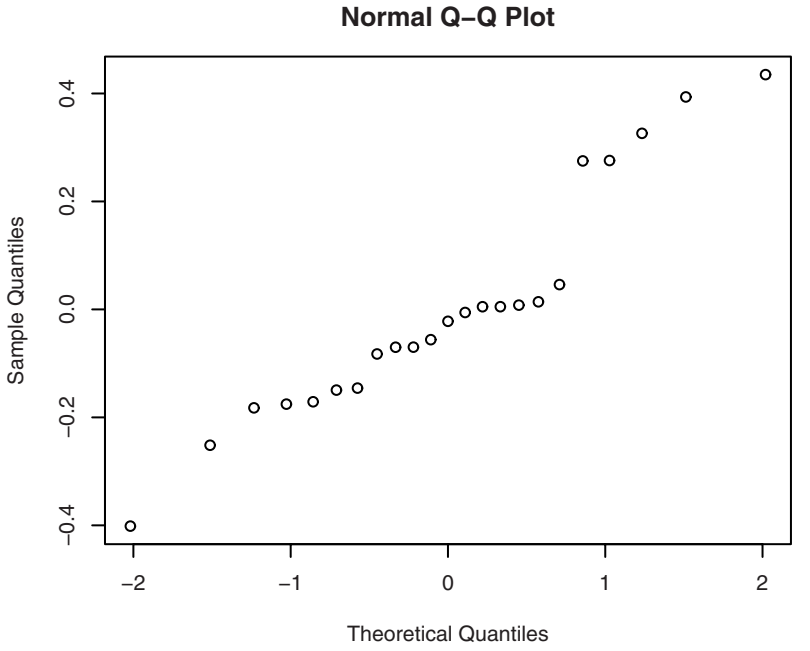

Figure 6.4. `short.velocity` and `blood.glucose`: Q–Q plot of residuals.

a well-determined line. These bands often show a marked curvature since the line is better determined near the center of the point cloud. This is a fact that can be shown mathematically, but you may also understand it intuitively as follows: The predicted value at $\bar{x}$ will be $\bar{y}$, whatever the slope is, and hence the standard error of the fitted value at that point is the SEM of the ys. At other values of x, there will also be a contribution from the variability of the estimated slope, having increasing influence as you move away from $\bar{x}$. Technically, you also need to establish that $\bar{y}$ and $\hat{\beta}$ are uncorrelated.

The wide bands, *prediction bands*, include the uncertainty about future observations. These bands should capture the majority of the observed points and will not collapse to a line as the number of observations increases. Rather, the limits approach the true line ± 2 standard deviations (for 95% limits). In smaller samples, the bands do curve since they include uncertainty about the line itself, but not as markedly as the confidence bands. Obviously, these limits rely strongly on the assumption of normally distributed errors with a constant variance, so you should not use such limits unless you believe that the assumption is a reasonable approximation for the data at hand.

Predicted values, with or without prediction and confidence bands, may be extracted with the function `predict`. With no arguments, it just gives the fitted values:

```
> predict(lm.velo)
       1        2        3        4        5        6        7
1.433841 1.335010 1.275711 1.526084 1.255945 1.214216 1.302066
       8        9       10       11       12       13       14
1.341599 1.262534 1.365758 1.244964 1.212020 1.515103 1.429449
      15       16       17       18       19       20       21
1.244964       NA 1.190057 1.324029 1.372346 1.451411 1.389916
      22       23       24
1.205431 1.291085 1.306459
```

If you add `interval="confidence"` or `interval="prediction"`, then you get the vector of predicted values augmented with limits. The arguments can be abbreviated:

```
> predict(lm.velo,int="c")
        fit      lwr      upr
1  1.433841 1.291371 1.576312
2  1.335010 1.240589 1.429431
...
23 1.291085 1.191084 1.391086
24 1.306459 1.210592 1.402326
> predict(lm.velo,int="p")
        fit       lwr      upr
1  1.433841 0.9612137 1.906469
2  1.335010 0.8745815 1.795439
...
23 1.291085 0.8294798 1.752690
24 1.306459 0.8457315 1.767186
Warning message:
In predict.lm(lm.velo, int = "p") :
  Predictions on current data refer to _future_ responses
```

`fit` denotes the expected values, here identical to the fitted values (they need not be; read on). `lwr` and `upr` are the lower and upper confidence limits for the expected values, respectively, the prediction limits for `short.velocity` for new persons with these values of `blood.glucose`. The warning in this case does not really mean that anything is wrong, but there is a pitfall: The limits should not be used for evaluating the *observed* data to which the line has been fitted. These will tend to lie closer to the line for the extreme x values because those data points are the more influential; that is, the prediction bands curve the wrong way.

The best way to add prediction and confidence intervals to a scatterplot is to use the `matlines` function, which plots the columns of a matrix against a vector.

There are a few snags to this, however: (a) The `blood.glucose` values are in random order; we do not want line segments connecting points haphazardly along the confidence curves; (b) the prediction limits, particularly the lower one, extend outside the plot region; and (c) the `matlines` command needs to be prevented from cycling through line styles and colours. Notice that the `na.exclude` setting (p. 115) prevents us from also having an observation omitted from the predicted values.

The solution is to *predict in a new data frame* containing suitable x values (here `blood.glucose`) at which to predict. It is done as follows:

```
> pred.frame <- data.frame(blood.glucose=4:20)
> pp <- predict(lm.velo, int="p", newdata=pred.frame)
> pc <- predict(lm.velo, int="c", newdata=pred.frame)
> plot(blood.glucose,short.velocity,
+      ylim=range(short.velocity, pp, na.rm=T))
> pred.gluc <- pred.frame$blood.glucose
> matlines(pred.gluc, pc, lty=c(1,2,2), col="black")
> matlines(pred.gluc, pp, lty=c(1,3,3), col="black")
```

What happens is that we create a new data frame in which the variable `blood.glucose` contains the values at which we want predictions to be made. `pp` and `pc` are then made to contain the result of `predict` for the new data in `pred.frame` with prediction limits and confidence limits, respectively.

For the plotting, we first create a standard scatterplot, except that we ensure that it has enough room for the prediction limits. This is obtained by setting `ylim=range(short.velocity, pp, na.rm=T)`. The function `range` returns a vector of length 2 containing the minimum and maximum values of its arguments. We need the `na.rm=T` argument to cause missing values to be skipped for the range computation; notice that `short.velocity` is included to ensure that points outside the prediction limits are not missed (although in this case there are none). Finally, the curves are added, using as x-values the `blood.glucose` used for the prediction and setting the line types and colours to more sensible values. The final result is seen in Figure 6.5.

6.4 Correlation

A correlation coefficient is a symmetric, scale-invariant measure of association between two random variables. It ranges from -1 to $+1$, where the extremes indicate perfect correlation and 0 means no correlation. The sign is negative when large values of one variable are associated with small values of the other and positive if both variables tend to be large or small

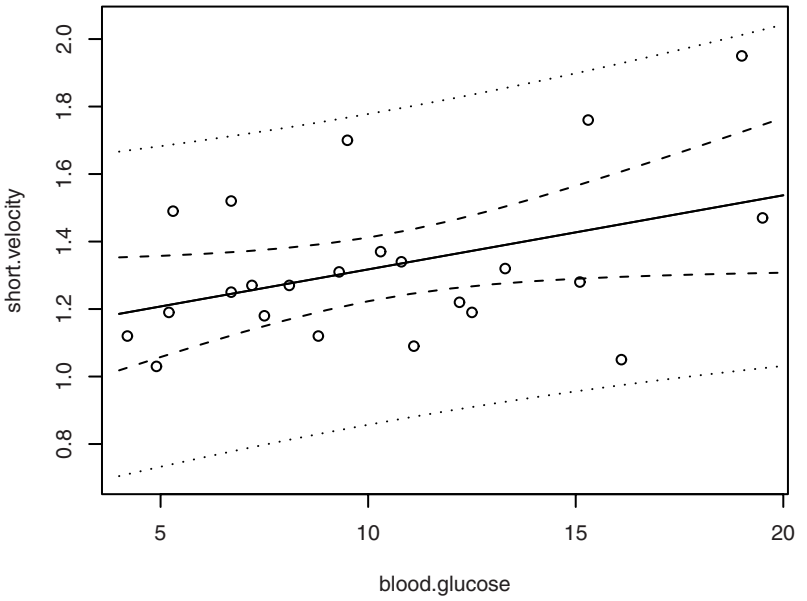

Figure 6.5. Plot with confidence and prediction bands.

simultaneously. The reader should be warned that there are many incorrect uses of correlation coefficients, particularly when they are used in regression-type settings.

This section describes the computation of parametric and nonparametric correlation measures in R.

6.4.1 *Pearson correlation*

The Pearson correlation is rooted in the two-dimensional normal distribution where the theoretical correlation describes the contour ellipses for the density. If both variables are scaled to have a variance of 1, then a correlation of zero corresponds to circular contours, whereas the ellipses become narrower and finally collapse into a line segment as the correlation approaches ± 1.

The empirical correlation coefficient is

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

It can be shown that $|r|$ will be less than 1 unless there is a perfect linear relation between x_i and y_i, and for that reason the Pearson correlation is sometimes called the "linear correlation".

It is possible to test the significance of the correlation by transforming it to a t-distributed variable (the formula is not particularly elucidating so we skip it here), which will be identical with the test obtained from testing the significance of the slope of either the regression of y on x or vice versa.

The function `cor` can be used to compute the correlation between two or more vectors. However, if it is naively applied to the two vectors in `thuesen`, the following happens:

```
> cor(blood.glucose,short.velocity)
Error in cor(blood.glucose, short.velocity) :
        missing observations in cov/cor
```

All the elementary statistical functions in R require either that all values be nonmissing or that you explicitly state what should be done with the cases with missing values. For `mean`, `var`, `sd`, and similar one-vector functions, you can give the argument `na.rm=T` to indicate that missing values should be removed before the computation. For `cor`, you can write

```
> cor(blood.glucose,short.velocity,use="complete.obs")
[1] 0.4167546
```

The reason that `cor` does not use `na.rm=T` like the other functions is that there are more possibilities than just removing incomplete cases or failing. If more than two variables are in play, it is also possible to use information from all nonmissing *pairs* of measurements (this might result in a correlation matrix that is not positive definite, though).

You can obtain the entire matrix of correlations between all variables in a data frame by saying, for instance,

```
> cor(thuesen,use="complete.obs")
               blood.glucose short.velocity
blood.glucose      1.0000000      0.4167546
short.velocity     0.4167546      1.0000000
```

Of course, this is more interesting when the data frame contains more than two vectors!

However, the calculations above give no indication of whether the correlation is significantly different from zero. To that end, you need `cor.test`. It works simply by specifying the two variables:

```
> cor.test(blood.glucose,short.velocity)

        Pearson's product-moment correlation

data:  blood.glucose and short.velocity
t = 2.101, df = 21, p-value = 0.0479
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.005496682 0.707429479
sample estimates:
      cor
0.4167546
```

We also get a confidence interval for the true correlation. Notice that it is exactly the same p-value as in the regression analysis in Section 6.1 and also that based on the ANOVA table for the regression model, which is described in Section 7.5.

6.4.2 *Spearman's ρ*

As with the one- and two-sample problems, you may be interested in nonparametric variants. These have the advantage of not depending on the normal distribution and, indeed, being invariant to monotone transformations of the coordinates. The main disadvantage is that its interpretation is not quite clear. A popular and simple choice is Spearman's rank correlation coefficient ρ. This is obtained quite simply by replacing the observations by their rank and computing the correlation. Under the null hypothesis of independence between the two variables, the exact distribution of ρ can be calculated.

Unlike group comparisons where there is essentially one function per named test, correlation tests are all grouped into `cor.test`. There is no special `spearman.test` function. Instead, the test is considered one of several possibilities for testing correlations and is therefore specified via an option to `cor.test`:

```
> cor.test(blood.glucose,short.velocity,method="spearman")

        Spearman's rank correlation rho

data:  blood.glucose and short.velocity
S = 1380.364, p-value = 0.1392
alternative hypothesis: true rho is not equal to 0
sample estimates:
     rho
0.318002

Warning message:
```

```
In cor.test.default(blood.glucose, short.velocity, method="spearman"):
  Cannot compute exact p-values with ties
```

6.4.3 *Kendall's τ*

The third correlation method that you can choose is Kendall's τ, which is based on counting the number of *concordant* and *discordant* pairs. A pair of points is concordant if the difference in the x-coordinate is of the same sign as the difference in the y-coordinate. For a perfect monotone relation, either all pairs will be concordant or all pairs will be discordant. Under independence, there should be as many concordant pairs as there are discordant ones.

Since there are many pairs of points to check, this is quite a computationally intensive procedure compared with the two others. In small data sets such as the present one, it does not matter at all, though, and the procedure is generally usable up to at least 5000 observations.

The τ coefficient has the advantage of a more direct interpretation over Spearman's ρ, but apart from that there is little reason to prefer one over the other.

```
> cor.test(blood.glucose,short.velocity,method="kendall")

        Kendall's rank correlation tau

data:  blood.glucose and short.velocity
z = 1.5604, p-value = 0.1187
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.2350616

Warning message:
In cor.test.default(blood.glucose, short.velocity, method="kendall"):
  Cannot compute exact p-value with ties
```

Notice that neither of the two nonparametric correlations is significant at the 5% level, which the Pearson correlation is, albeit only borderline significant.

6.5 Exercises

6.1 With the `rmr` data set, plot metabolic rate versus body weight. Fit a linear regression model to the relation. According to the fitted model,

what is the predicted metabolic rate for a body weight of 70 kg? Give a 95% confidence interval for the slope of the line.

6.2 In the `juul` data set, fit a linear regression model for the square root of the IGF-I concentration versus age to the group of subjects over 25 years old.

6.3 In the `malaria` data set, analyze the log-transformed antibody level versus age. Make a plot of the relation. Do you notice anything peculiar?

6.4 One can generate simulated data from the two-dimensional normal distribution with a correlation of ρ by the following technique: (a) Generate X as a normal variate with mean 0 and standard deviation 1; (b) generate Y with mean ρX and standard deviation $\sqrt{1-\rho^2}$. Use this to create scatterplots of simulated data with a given correlation. Compute the Spearman and Kendall statistics for some of these data sets.