MATH5743M: Statistical Learning

Dr Seppo Virtanen, School of Mathematics, University of Leeds

Semester 2: 2022

Week 3: Linear Regression

The first statistical model we are going to examine is linear regression. Most of you will have encountered linear regression at some point, even if through a dimly-remembered process you learnt at school for finding the 'line of best fit'. What is often not appreciated is just how powerful and important linear regression is as a way of making sense of data. Take for example the tweet by Robin Hanson in Figure 1. Hanson argues that while today many companies sitting on data think they need the most sophisticated possible machine-learning tools, what they really need is someone to apply a linear regression model to data that is currently going unused. This is a good lesson: while there is a lot of hype at the moment about artificial intelligence and machine learning, many of the big gains in data science have come from cleaning up disorganised data and applying quite fundamental statistical tools to these.

robin hanson
@robinhanson

Good CS expert says: Most firms that thinks they want advanced AI/ML really just need linear regression on cleaned-up data.

RETWEETS 1,002 LIKES 1,661

6:19 PM - 28 Nov 2016

Figure 1: Tweet by Robin Hanson, Dec 2 2016

What is more, linear regression really forms the backbone of many more advanced methods. We won't be able to explore this fully in this course, but on a fundamental level a great many apparently sophisticated techniques can be seen as variations of linear regression. We will see some examples as we go along. For now, we start at the most basic level, inferring a linear relationship between one input and one output.

The linear model in one-dimension

For a one dimensional linear regression problem, we assume that the output, Y is a sum of three components:

- A scalar constant – β_0
- The input, X multiplied by a second constant – $\beta_1 x$

- A residual ϵ, which is unknown, but assumed to come from a Normal distribution with zero mean and unknown variance: $\epsilon \sim (0, \sigma^2)$

Therefore we may write the model in full as:

$$Y = \beta_0 + \beta_1 X + \epsilon, \; \epsilon \sim \mathcal{N}(0, \sigma) \tag{1}$$

The Likelihood

The key step in progressing from writing down the model to doing statistical inference is to translate the equation above into a *likelihood* function. To do this we need to consider what our model says about the *probability* of observing an output $Y = y$, given that we have seen an input $X = x$.

Recall from last week that the likelihood is the probability of generating the data that we see, conditioned on a particular model and set of parameters. In the linear regression model we assume that each output value is generated *independently* from a normal distribution with some variance σ^2, and with a mean that is a linear function of the associated *input*

$$\begin{aligned}
\mathcal{L}(\beta_0, \beta_1, \sigma) &= P(\text{Data} \mid \beta_0, \beta_1, \sigma) \\
&= P(Y = y \mid X = x, \beta_0, \beta_1, \sigma) \\
&= \mathcal{N}(y; \beta_0 + \beta_1 x, \sigma^2) \\
&= \prod_{i=1}^{n} \mathcal{N}(y_i; \beta_0 + \beta_1 x_i, \sigma^2)
\end{aligned} \tag{2}$$

Note, importantly, that the final product over the n data points is possible because we assumed that each output was generated *independently*. Therefore the probability for all of the data points is simply the product of the probability of observing each, individually.

Recall as well from last week that it is usually easier to work with the *log-likelihood*, both for mathematical and numerical reasons. Therefore, using the above we will also define the log-likelihood:

$$\log \mathcal{L} = \sum_{i=1}^{n} \log \mathcal{N}(y_i; \beta_0 + \beta_1 x_i, \sigma^2) \tag{3}$$

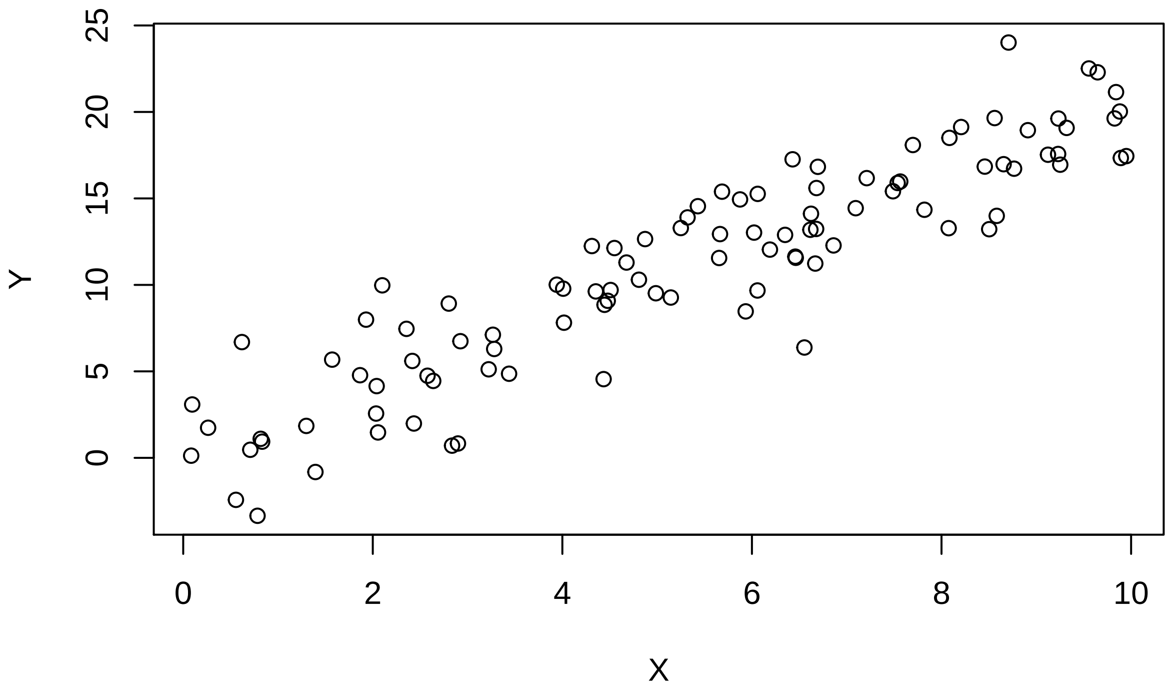
Visualising the likelihood

Lets consider data generated from a very simple linear model with a zero intercept. I will assume a gradient, β of 2 and a variance of 3^2.These are just arbitrary numbers for the purposes of illustration.

$$Y = \beta X + \epsilon, \; \epsilon \sim \mathcal{N}(0, \sigma^2 = 3^2), \; \beta = 2 \tag{4}$$

We can generate data from this model simply in R, taking a random sample of X values between 0 and 10

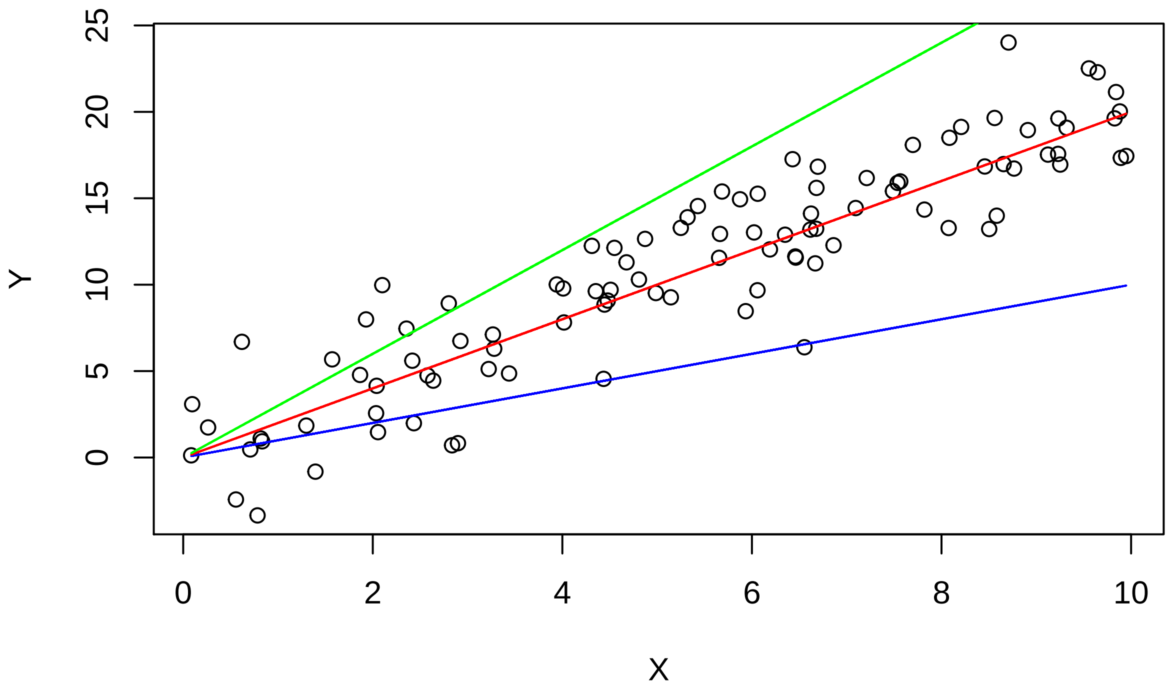
```r
#Define the number of data points
N = 100
X = runif(n=N, min=0, max=10)
Y = 2*X + rnorm(n=N, mean=0, sd= 3)
plot(X, Y)
```



Looking at the data we obviously see a clear trend between X and Y. But, assuming we did not already know the answer, we would ask, 'what gradient would be seeing this data most probable?'

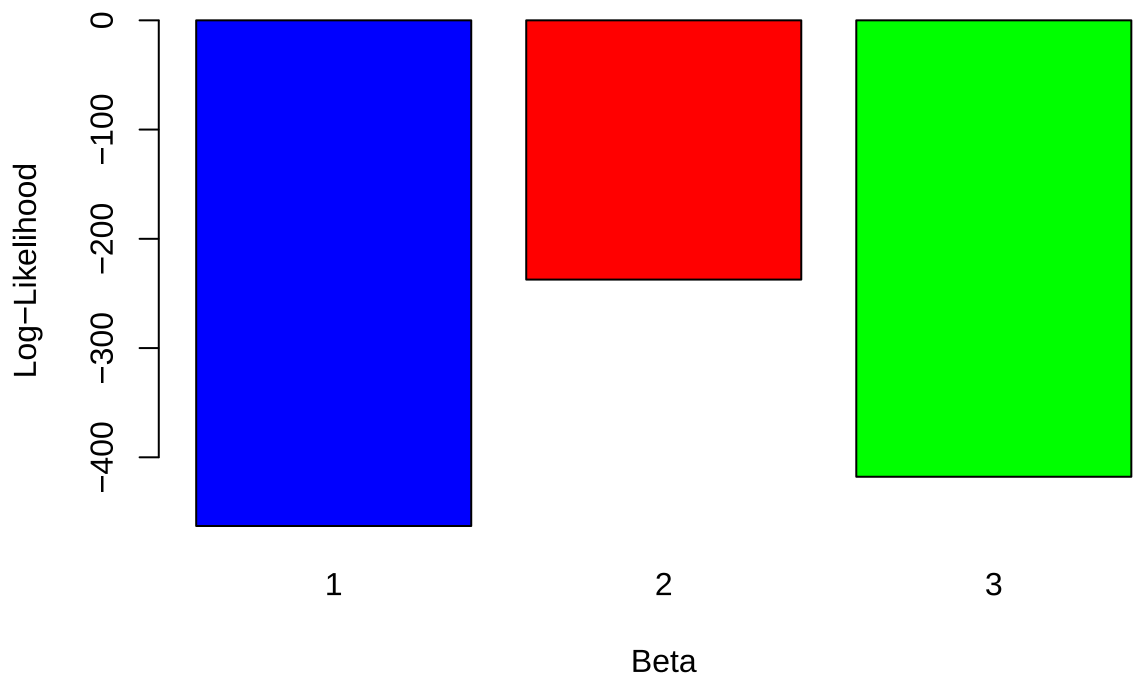
Consider guessing three possible values for the gradient: 1, 2 or 3. First lets see how well each of these fits visually by drawing the lines of best fit they imply on top of the data.

```r
plot(X, Y)
lines(X, 1*X, col="blue")
lines(X, 2*X, col="red")
lines(X, 3*X, col="green")
```



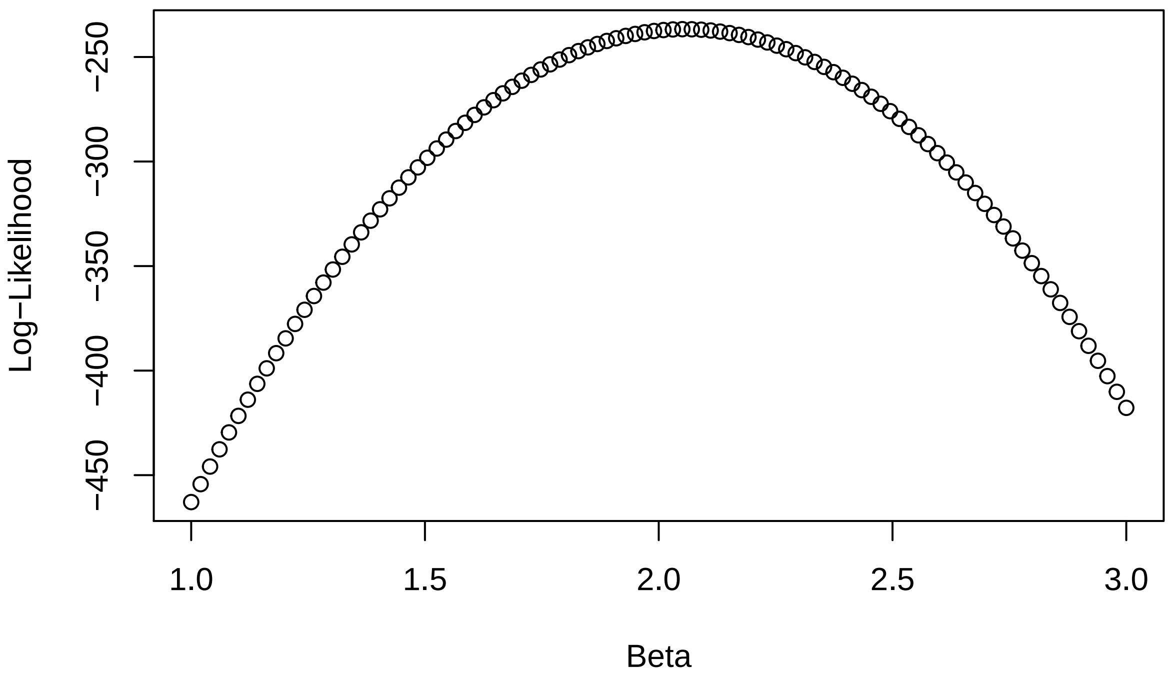
Visually it is clear that the red line, indicating $\beta = 2$, fits much better than either the blue line for $\beta = 1$ or the green line for $\beta = 3$. As it should do! How are these different degrees of visual fit reflected in the likelihood. Lets evaluate the log-likelihood function for each guess and find out. Here we'll introduce the **dnorm** function, which evaluates the normal distribution probability density function. Using the option **log=TRUE** forces it to return the log probability density, which is what we want.

```
#Define an empty vector of 3 elements
LL = rep(NA, 3)
#Define the 3 possible guesses
beta=c(1,2,3)
#Calculate the log-likelihood for each beta
for (i in 1:3){
  LL[i] = sum(dnorm(x=Y, mean=beta[i]*X, sd=3, log=TRUE))
}
#plot a bar chart showing the different log-likelihoods
barplot(LL, names.arg = c("1", "2", "3"), xlab="Beta",
        ylab="Log-Likelihood", col=c("blue", "red", "green"))
```



As expected, of the three guess, $\beta = 2$ has the highest log-likelihood, reflecting the better visual fit we saw before. Of course, with a computer we don't need to limit ourselves to just three guesses, we can evaluate the log-likelihood over many possible values of β. Lets try 100 values between 1 and 3.

```
#Define the number of guesses
N_guesses = 100
#Define an empty vector of N_guesses elements
LL = rep(NA, N_guesses)
#Define the guesses
beta=seq(1, 3, length.out=N_guesses)
#Calculate the log-likelihood for each beta
for (i in 1:N_guesses){
  LL[i] = sum(dnorm(x=Y, mean=beta[i]*X, sd=3, log=TRUE))
}
#This time plot the log-likelihood as points
plot(beta, LL, xlab="Beta", ylab="Log-Likelihood")
```



```r
idx = which.max(LL)
print(paste("MLE estimate of beta is: ", beta[idx], collapse=""))
```

```
## [1] "MLE estimate of beta is:  2.05050505050505"
```

Numerical optimisation

In the section above we visualised the log-likelihood as a function of the parameter we are trying to estimate, the gradient of our regression line. Recall that this is exactly how we first looked at optimising a function. In the example we looked at last week, we saw how a numerical optimisation routine could be used to get a faster and more accurate answer. We can do the same here, as below. Recall that the **optim** function is set up to *minimise*, so the first thing we have to do is define an objective function to minimise, which is the *negative* log-likelihood. Then we apply the **optim** function and print out the optimised value of the parameter, which is our regression gradient. I will use a starting guess for the regression gradient of 1, but this is simply an arbitrary guess; any reasonable value will do (the log-likelihood in this case only has one peak, so wherever we start we will converge to the same solution).

```r
neg_LL <- function(param){-sum(dnorm(x=Y, mean=param*X, sd=3, log=TRUE))}
optim_result = optim(par = 1, fn=neg_LL)
print(paste("MLE estimate of beta is: ", optim_result$par, collapse=""))
```

```
## [1] "MLE estimate of beta is:  2.05546875"
```

As we can see, we get a very similar answer as above - a good sanity check! The slight difference is simply due to the spacing between the points in our original evaluation of the log-likelihood.

Maximising the likelihood - the exact solution

As promised, this week we get our teeth stuck into a serious derivation. We have looked at how to visualise and numerically optimise the log-likelihood for the linear regression model. However, for this model there is an analytical solution that we can derive using a bit of calculus, and this solution is ultimately what most software you will encounter will use. As linear regression is such a fundamental model, it is worth spending a bit of time and effort to see both what that result is, and how it is derived.

We start, as always, from the definition of the log-likelihood:

$$\begin{aligned}
\log \mathcal{L}(\beta_0, \beta_1, \sigma) &= \sum_{i=1}^{n} \log \mathcal{N}(y_i; \beta_0 + \beta_1 x_i, \sigma^2) \\
&= \sum_{i=1}^{n} \log \left(\frac{1}{\sqrt{2\pi}\sigma} \exp \left(\frac{-(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2} \right) \right) \\
&= \sum_{i=1}^{n} - \frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2} - n \log(\sigma) - \frac{n}{2} \log 2\pi
\end{aligned} \tag{5}$$

We can do some calculus to find the maximum of the log-likelihood with respect to each of its three parameters. First, lets look at β_0. The condition for a maximum is:

$$\frac{\partial \log \mathcal{L}}{\partial \beta_0}|_{\hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}} = 0 \tag{6}$$

$$\begin{aligned}
\sum_{i=1}^{n} \frac{(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)}{\hat{\sigma}^2} &= 0 \\
\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) &= 0 \\
\Rightarrow \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}
\end{aligned} \tag{7}$$

Now lets consider β_1. The condition for a maximum is:

$$\frac{\partial \log \mathcal{L}}{\partial \beta_1}|_{\hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}} = 0 \tag{8}$$

therefore

$$\begin{aligned}
\sum_{i=1}^{n} \frac{(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i}{\hat{\sigma}^2} &= 0 \\
\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i &= 0 \\
\bar{yx} - \hat{\beta}_0 \bar{x} - \hat{\beta}_1 \bar{x^2} &= 0 \\
\bar{yx} - \bar{y}\bar{x} + \hat{\beta}_1 \bar{x}^2 - \hat{\beta}_1 \bar{x^2} &= 0, \text{ [using } \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}] \\
\Rightarrow \hat{\beta}_1 &= \frac{\bar{yx} - \bar{y}\bar{x}}{\bar{x^2} - \bar{x}^2} \\
&= \frac{\textsc{cov}(y, x)}{\textsc{var}(x)}
\end{aligned} \tag{9}$$

Finally we consider σ^2. The condition for a maximum is

$$\frac{\partial \log \mathcal{L}}{\partial \sigma}|_{\hat{\beta_0},\hat{\beta_1},\hat{\sigma}} = 0 \tag{10}$$

substituting the expression for $\mathcal{L}$ we have:

$$\begin{aligned}\sum_{i=1}^{n} \frac{(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2}{\hat{\sigma}^3} - \frac{n}{\hat{\sigma}} = 0 \\ \Rightarrow \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2\end{aligned} \tag{11}$$

Leaving us with the following three relationships that define the maximum-likelihood estimators for the parameters in this model

1. $\hat{\beta}_1 = \frac{\bar{yx} - \bar{y}\bar{x}}{\bar{x^2} - \bar{x}^2} = \frac{\text{COV}(y,x)}{\text{VAR}(x)}$
2. $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$
3. $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$

A note on estimating σ^2

If you look at the MLE estimates above, you may notice something similar to a topic we discussed last week. The MLE estimator for σ^2 has a denominator of n, just like the MLE estimator for the variance in a single sample. Recall that the MLE estimator we looked at last week was *biased* – on average it would not give us the true variance. The same is true for the MLE estimator of σ^2 here. Just like before, we can instead use an *unbiased* estimator:

$$\hat{\sigma}^2_{\text{unbiased}} = \frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 \tag{12}$$

Note the $n - 2$ denominator. This is different to the estimator we saw last week where the denominator was $n - 1$. The general rule to remember for estimating variance parameters like this is that the denominator is $n - p - 1$, where p is the number of input variables you estimated regression coefficients for. So in last weeks case we did not estimate any regression coefficients and there were no input variables, so we used $n - 1$ (the 1 comes from estimating the mean, which is equivalent to estimating an intercept in a regression model). In this weeks one-dimensional regression we estimated coefficients for one input (as well as the intercept), so we use $n - 2$. Similarly, next week we will look at higher dimensional problems and the denominator will change accordingly.

Doing linear regression in practice: the glm command

Linear regression is an extremely common statistical procedure, and any serious statistical software will enable you to perform a linear regression very easily. The purpose of going through the mathematical details is so that you can understand exactly how statistical learning works; how a model of the data generating process is combined with an optimisation procedure to create a fitted model that can be used to analyse and predict. But it is also important that you learn how to efficiently perform these tasks computationally as well. In R the best tool for fitting linear models is the **glm** command within the **stats** package. GLM stands for Generalised Linear Model, which we will learn more about later in the course. Using the default options, this tool gives you the ability to quickly create and fit a linear model. To use it you need to understand how to set up your data in the correct way, how to interpret the resulting model that is produced, and how to use it to make predictions.

The first step we need is to understand how R likes to store data for statistical analysis. We have previously created simple vectors of numbers, but R has a particular structure known as a *data frame* that allows you to store data in a more easily read and understood format. A data frame is like a matrix with labelled columns, so that you can refer to each column by name, rather than by its numerical order. To demonstrate, lets create a data frame using the values of X and Y that we already analysed

```
mydata = data.frame(input=X, output=Y)
```

Here we create two columns labelled input and output and assign them the values of X and Y from before. We can see what the dat frame looks like by asking for the **head**, the first few rows

```
head(mydata)
```

```
##      input    output
## 1 9.554905 22.513724
## 2 6.462444 11.567287
## 3 7.210652 16.172983
## 4 5.320666 13.902074
## 5 6.058272  9.676456
## 6 8.560329 19.646652
```

You can see the two columns clearly, with each input having an associated output. If we want to access the values of particular columns we can refer to these by name. For example, there are at least three ways we can retrieve the 3rd value of the output.

```
mydata$output[3]
```

```
## [1] 16.17298
```

```
mydata[3, 'output']
```

```
## [1] 16.17298
```

```
mydata[3, 2]
```

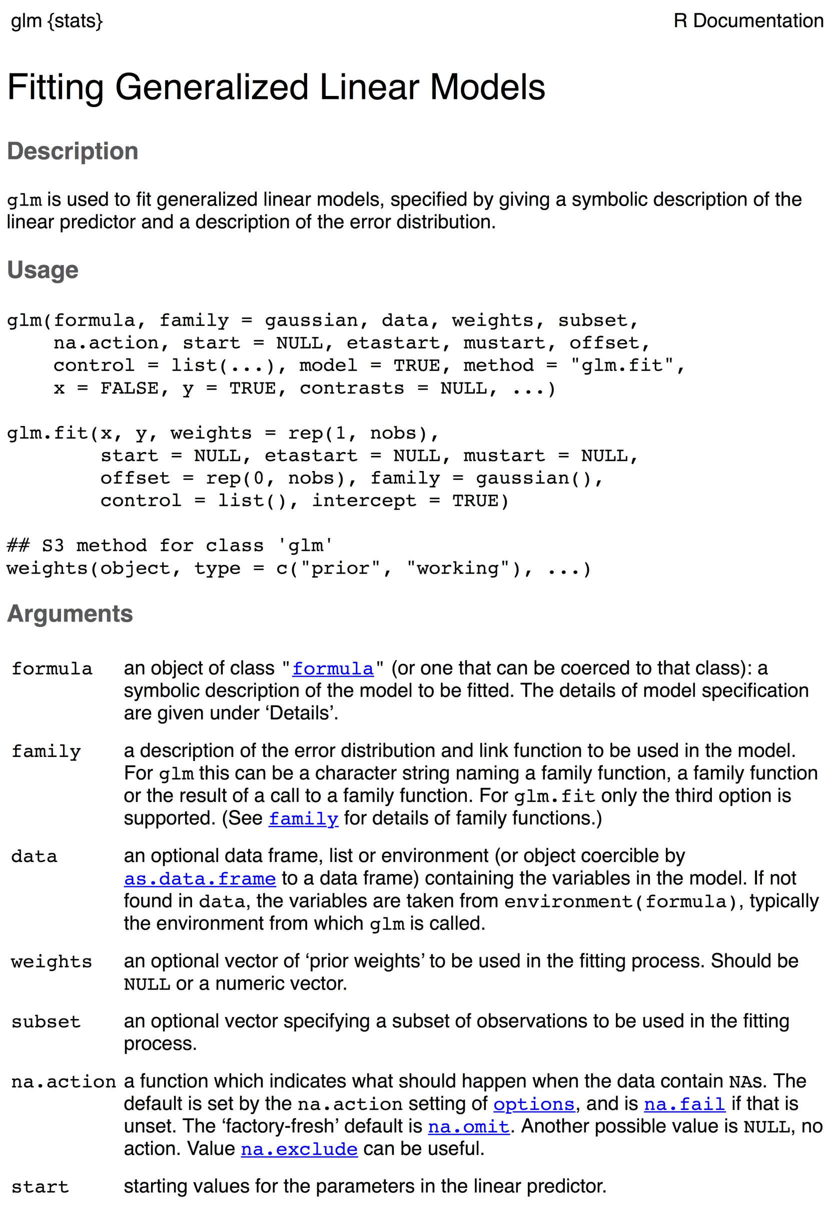
```
## [1] 16.17298
```

Notice that the first two of these do not depend on the order in which I assigned data to the data frame. They are therefore more robust to mistakes.

With our newly created data frame we are ready to use the **glm** command to perform a linear regression. First we check the help file for **glm** to see what arguments are necessary (see Figure 2 for output)

```
help(glm)
```

The *formula* argument specifies the names of the inputs and outputs we want to use, corresponding to names in a data frame specified by the *data* argument. For now we can forget about the other arguments and leave them at their default values. To perform the same regression as we did above ourselves using **glm**, we simply specify the data frame we just created and the names for the inputs and outputs. We store the resulting model that is produced by **glm** as 'mymodel', and use the **summary** command to produce a summary of the fitted model



glm {stats} R Documentation

Fitting Generalized Linear Models

Description

`glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, contrasts = NULL, ...)

glm.fit(x, y, weights = rep(1, nobs),
        start = NULL, etastart = NULL, mustart = NULL,
        offset = rep(0, nobs), family = gaussian(),
        control = list(), intercept = TRUE)

## S3 method for class 'glm'
weights(object, type = c("prior", "working"), ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`family`	a description of the error distribution and link function to be used in the model. For `glm` this can be a character string naming a family function, a family function or the result of a call to a family function. For `glm.fit` only the third option is supported. (See `family` for details of family functions.)
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`start`	starting values for the parameters in the linear predictor.

Figure 2: Help file for glm command

```
mymodel = glm(output ~ input, data = mydata)
summary(mymodel)
```

```
##
## Call:
## glm(formula = output ~ input, data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -7.0965  -1.4128  -0.1398   1.8761   6.1500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.13013    0.54238    0.24    0.811
## input        2.03646    0.08972   22.70   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.414095)
##
##     Null deviance: 3932.91  on 99  degrees of freedom
## Residual deviance:  628.58  on 98  degrees of freedom
## AIC: 473.62
##
## Number of Fisher Scoring iterations: 2
```

The summary provided gives us lots of information. For the moment we are most interested in the values of the coefficients, which match closely to those we found by doing our own optimisation before. Note as well that the summary provides 'standard errors' for each coefficient. These tell us how uncertain we are about each the value of coefficient. You can think of each estimate of a coefficient being a random variable whose value is normally distributed. The standard error is the estimated standard deviation of this distribution.

We can therefore construct 95% (or any percentile) confidence intervals for the true coefficient value using this. Recall from semester 1 how you construct a confidence interval for the mean of a population. You use the data provided to estimate the population mean (simply the mean of your data), and the standard error (the sample variance divided by the number of observation in the data, n). From this you construct an interval. For example, for a 95% C.I. that interval is: sample mean $\pm$ standard error $\times t_{2.5\%,n-1}$, where $t_{2.5\%,n-1}$ indicate the critical value for a t-distribution with $n-1$ degrees of freedom where $P(x > t_{2.5\%,n-1}) = 0..025$. Recall that we use a t-distribution when sample sizes are small (generally $n < 50$), to account for the extra uncertainty this introduces - as n grows beyond 50 it rapidly, $t_{2.5\%,n-1}$ rapidly tends towards a stable value of around 1.96. We used a t-distribution with $n-1$ degrees of freedom because when we estimate the *variance* of the population from the data we have only $n-1$ independent data points; because the mean was also estimated from the data, then if we are given the mean and $n-1$ of the observations we can always reconstruct the nth value.

In the case of the linear regression coefficients, we can follow a very similar process to construct confidence intervals for the estimates of the coefficient values. The key difference is to recognise that we have further reduced the number of effectively independent data points, the degrees of freedom, in estimating the standard error. This is because we have estimated both the intercept value, β_0 and the gradient, β_1, before we use these to generate the estimate of the variance. Therefore the number of degrees of freedom is reduced to $n-2$. If you were to give me $n-2$ of the observations, along with the estimated values of β_0 and β_1, I could always tell you what the missing two observations were. Therefore the 95% C.I. for any regression coefficient in this one-dimensional model is:

estimate $\pm$ standard error $\times t_{2.5\%,n-2}$

These confidence intervals work exactly the same as those you will have learnt about in the first semester or your previous statistics courses: we interpret them as meaning that there is a $\alpha\%$ probability that the real coefficient value lies within the $\alpha\%$ confidence interval. In most practical applications we will almost always use $\alpha = 95$ (as above), though you can easily adapt for any desired interval by finding the appropriate critical t-value.

To give an example of this in practice, imagine that we have performed a linear regression analysis. Our statistical software (probably R) gives us the following summary of the fitted model

Formula: $y \sim x$. $n = 20$

	Estimate	Standard Error	t
Intercept	2.1	0.3	7.0
x	1.2	0.2	7.5

How should be we determine a confidence interval for the coefficient associated with x? In this table, the first column gives us the best (maximum-likelihood) estimate for the coefficient – the coefficient for x is in the second row. So $\hat{\beta_1} = 1.2$. The second column gives the standard error associated with this estimate. The third column, t, simply represents the number of standard errors that the estimate is away from zero, i.e. estimate/standard error.

To construct a 95% C.I. we need the estimate, the standard error, and n, the number of observations. Here n is shown clearly at the top as being 20. For many model summaries this will not always be so clearly visible – you may need to check yourself how many observations are contained in the data. Using the expression for the confidence interval given above, our 95% C.I. is

C.I. for $\beta_1 : \hat{\beta_1} \pm \text{standard error} \times t_{2.5\%,18}$ C.I. for $\beta_1 : 1.2 \pm 0.2 \times 2.10$

This gives a minimum value inside the interval of 0.78 – substantially above zero. This gives us a good indication that x is a useful predictor for y.

You will often see a p-value stated alongside the parameter estimates in a model summary. Last semester you learned how confidence intervals and significance tests were related. A confidence interval for a model coefficient that does not contain zero indicates that we can reject the null hypothesis that the true coefficient is zero, just as with estimates for a population mean. You will often see this used by researchers to identify whether an input is a predictor, or even a *cause* of the output. This should be treated with some suspicion; we will see in later weeks that there are better ways to identify the best predictors for a given output, and claims of causality obtained simply from statistical analyses are often questionable. Nonetheless, a confidence interval that places a coefficient far from zero, and/or a very low p-value (below 0.05 typically) are indications that the input in question provides some information about the output.

Try constructing a 95% confidence interval for the gradient coefficient in the from the model summary above (from the simulated data). Does the interval contain the actual value used to generate the data?

The role of uncertainty

Estimating the uncertainty in regression coefficients is very important. We may calculate an estimate for the regression coefficient linking some input, x to some output y and find that the value giving the best fit is 1. But imagine if the standard error on this estimate is 0.9; then there is a substantial probability (can you calculate it?) that the real relationship between x and y is not only not equal to 1, but may even be negative. In such a case we must accept that there is not much the data can tell us about the real relationship, and we may need to collect more data before we can say anything with confidence.