


# Tutorial 6 Solutions

## Question 1

(11.4) Auditors are often required to compare the audited (or current) value of an inventory item with the book (or listed) value. If a company is keeping its inventory and books up to date, there should be a strong linear relationship between the audit values and book values. A company sampled 10 inventory items and obtained the values in the table below.

Item	Audit Value ( $y_i$ )	Book Value ( $x_i$ )
1	9	10
2	14	12
3	7	9
4	29	27
5	45	47
6	109	112
7	40	36
8	238	241
9	60	59
10	170	167

- (a)  Fit the model  $Y = \beta_0 + \beta_1 x + \varepsilon$  to the data.

We begin by entering the data into **R** as a data frame. Recall that data frames in **R** are constructed by calling the `data.frame()` function and supplying name-value pairs.

```
audit <- data.frame(  
  audit_value = c(9, 14, 7, 29, 45, 109, 40, 238, 60, 170),  
  book_value = c(10, 12, 9, 27, 47, 112, 36, 241, 59, 167)  
)
```

We can build linear models in **R** using the `lm()` function. Call `?lm` in the console to pull up the documentation! For basic linear models, we will be supplying two components to the `lm()` function: a formula of variables we want included in the model, and the name of the data set from which the referenced variables exist.

In the most basic cases, a formula is constructed by supplying the dependent (response) variable, followed by a tilde, followed by the independent (predictor) variables joined by plus signs.

Reading the problems that we are trying to solve in (b) and (c), we can deduce that the book value is predictor and the audit value is the response. The linear model is constructed as follows:

```
audit_lm <- lm(audit_value ~ book_value, data=audit)
```

After fitting a model, we typically call the `summary()` function to inspect our model.

```
summary(audit_lm)
```

```
##  
## Call:  
## lm(formula = audit_value ~ book_value, data = audit)  
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.7557 -2.1477 -0.4228  1.4803  3.7178
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.7198     1.1764   0.612   0.558
## book_value     0.9914     0.0114  86.994 3.4e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.666 on 8 degrees of freedom
## Multiple R-squared:  0.9989, Adjusted R-squared:  0.9988
## F-statistic: 7568 on 1 and 8 DF, p-value: 3.401e-13
```


The summary gives a lot of information that we don't need at this point but will likely come back to at a future date. We can obtain the predicted coefficients without any additional, unnecessary output by wrapping our model with the `coef()` function, which will return a named vector of the predicted coefficients.

```
coef(audit_lm)
```

```
## (Intercept)  book_value
##    0.7198048    0.9913916
```

From the above, the equation of our fitted line is given by

$$\widehat{\text{Audit Value}} = 0.720 + 0.991 * \text{Book Value}$$

- (b)  What is your estimate for the expected change in audited value for a one-unit change in book value?

The estimate for the expected change in audited value for a one-unit change in book value is precisely the value of  $\hat{\beta}_1$ . From the equation above, we expect a change of 0.991 units in audit value for a one-unit change in book value.

- (c)  If the book value is  $x = 100$ , what would you use to estimate the audited value?

To estimate the audit value at a particular book value, we can simply plug the given book value into our equation obtained in (a).

**Caution:** It is not recommended to predict  $y$  values for  $x$  values that are outside the range of the  $x$  values in the data set. Doing so can lead to nonsensical results. As such, one should always check that the provided value(s) of  $x$  fall inside the range of the  $x$  values used to fit the model.

We can obtain a prediction from a fitted linear model by passing our linear model into the `predict()` function. The point(s) to be predicted upon should be supplied to the `newdata` argument and **must** be a data frame containing columns with the same names as the predictors in the model.

```
predict(audit_lm, newdata=data.frame(book_value = 100))
```


```
##      1
## 99.85896
```

For a book value of 100, we estimate the audited value to be 99.86.

## Question 2

(11.14) J.H. Matis and T.E. Wehrly report the following table of data on the proportion of green sunfish that survive a fixed level of thermal pollution for varying lengths of time.

Proportion of Survivors ( $y$ )	Scaled Time ( $x$ )
1.00	.10
.95	.15
.95	.20
.90	.25
.85	.30
.70	.35
.65	.40
.60	.45
.55	.50
.40	.55

- (a)  Fit the linear model  $Y = \beta_0 + \beta_1 x + \varepsilon$ . Give your interpretation.

We first enter the data into **R**.

```
sunfish <- data.frame(  
  prop_survivors = c(1.00, 0.95, 0.95, 0.90, 0.85, 0.70, 0.65, 0.60, 0.55, 0.40),  
  scaled_time = c(0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55)  
)
```

We can then fit the model, taking the same steps as before.


```
sunfish_lm <- lm(prop_survivors ~ scaled_time, data=sunfish)  
  
coef(sunfish_lm)
```

```
## (Intercept) scaled_time  
##      1.182424    -1.315152
```

The equation of the fitted line is given by:

$$\widehat{\text{Proportion of Survivors}} = 1.182 - 1.315 * \text{Scaled Time}.$$

This says that for a unit increase in scaled time, we should expect a decrease in the proportion of survivors by 1.315 units. However, since the response variable is a proportion and as such, should be a value between 0 and 1, we really should never see an increase in scaled time by an entire unit.

- (b)  Plot the points and graph the result of part (a). Does the line fit through the points?

I will be using the **ggplot2** package for plotting rather than the plotting methods that exist in base-**R**. To install the **ggplot2** package, in the console, type:

```
install.packages("ggplot2")
```

If **ggplot2** is already installed, we can load it by calling (in our script):

```
library(ggplot2)
```

**Note:** Packages need to be reloaded each time you start a new **R** session!

The construction of plots via **ggplot2** is done through the addition of layers upon layers.

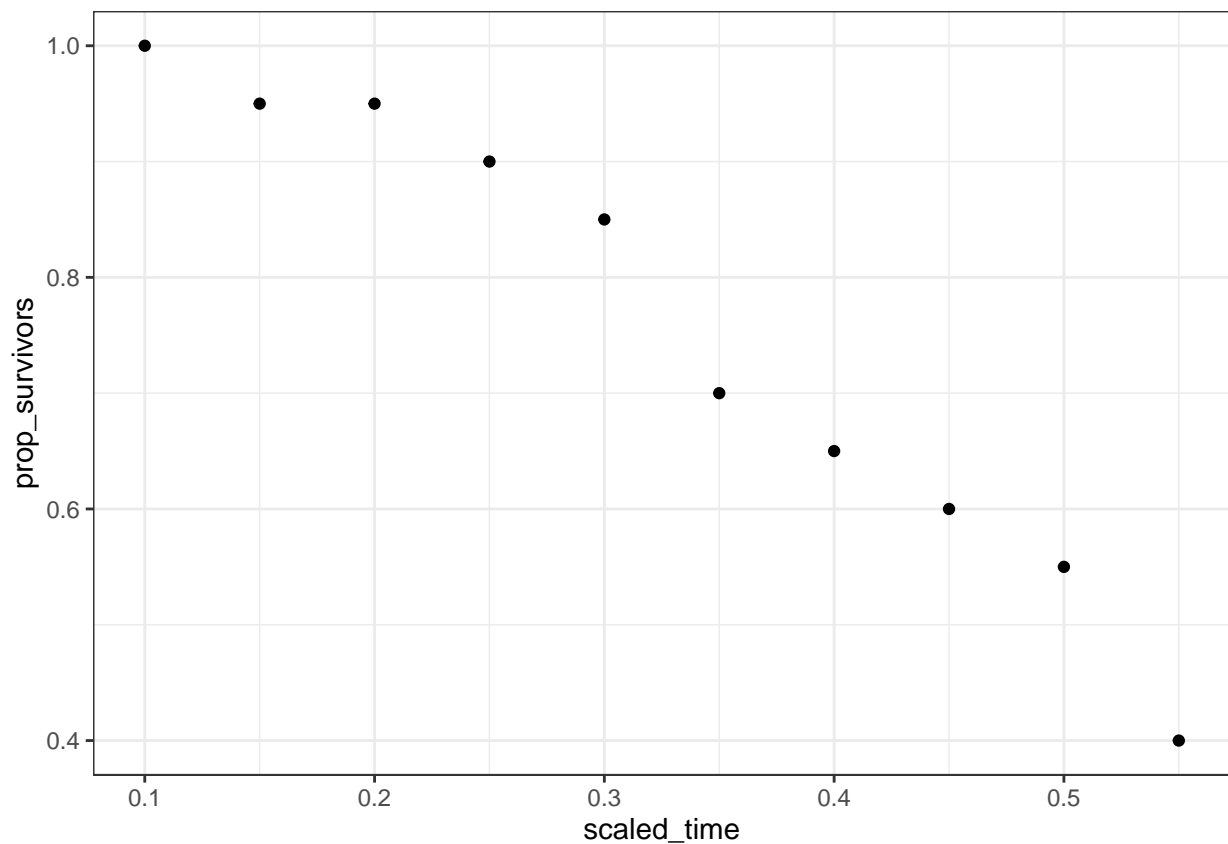
We start by initializing a blank canvas. For linear models, this is done by calling the `ggplot()` function (note that the package is called `ggplot2` but the function is called `ggplot()`) and passing in our linear model.

```
ggplot(sunfish_lm)
```



Next, we add our data points to our blank canvas as a new layer. This is done by joining our canvas with the `geom_point()` function by a plus symbol. Within the `geom_point()` function, we also specify `aes(x=scaled_time, y=prop_survivors)` to indicate that we want the points to be plotted according to the values found under the respective variables.

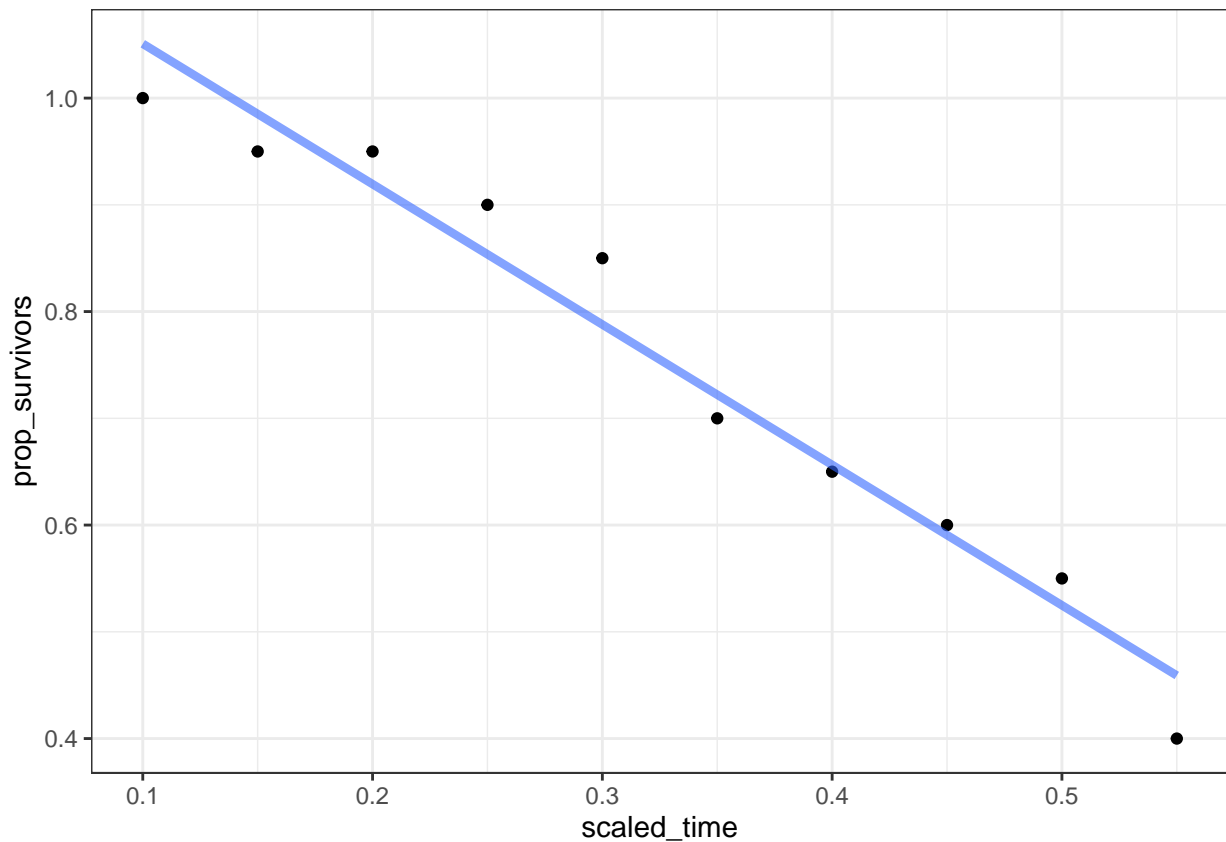
```
ggplot(sunfish_lm) +  
  geom_point(aes(x=scaled_time, y=prop_survivors))
```



Finally, we add our fitted line to the plot as another layer. This is done by adding a `geom_line()` layer to our previous plot, once again using a plus symbol. Inside the `geom_line()` function, we also specify `aes(x=scaled_time, y=.fitted)` as the predicted line is constructed by passing each predictor value through our model to obtain predicted response values, and then joining these points together.

Outside of the `aes()`, I will also specify `colour="#3366FF"` (self-explanatory), `size=1.5` (self-explanatory), and `alpha=0.6` to add some transparency to the line.

```
ggplot(sunfish_lm) +  
  geom_point(aes(x=scaled_time, y=prop_survivors)) +  
  geom_line(aes(x=scaled_time, y=.fitted), colour="#3366FF", size=1.5, alpha=0.6)
```



Note that since both the points layer and line layer shared a common x-variable, it would have been equivalent to supply a x-aesthetic in the declaration of the canvas, which would be inherited by all future layers. You can try running the code below and you will notice that the output is identical:

```
ggplot(sunfish_lm, aes(x=scaled_time)) +  
  geom_point(aes(y=prop_survivors)) +  
  geom_line(aes(y=.fitted), colour="#3366FF", size=1.5, alpha=0.6)
```

Returning to the question, the line provides a reasonable fit for the data. Notice that our fitted line does not actually pass through *any* of the data points. However, it is *not* a requirement that our fitted line should pass through any of our data points for it to be a good fit.

Recall that the method of least squares finds a line that minimizes the sum of the squared distance between all points and the fitted line. This does not guarantee that our fitted line will pass through all (or any) given data points. It is a common misconception that the least squares line should pass through every point in the data set for it to be a good fit. Looking at the plot above, it would be impossible to find a single straight line that could pass through all of our data points!

### Question 3

Suppose that eight specimens of a certain type of alloy were produced at different temperatures, and the durability of each specimen was then observed. The observed values are given in the table below, where  $x_i$  denotes the temperature (in coded units) at which specimen  $i$  was produced, and  $y_i$  denotes the durability (in coded units) of that specimen.

$i$	Durability ( $y_i$ )	Temperature ( $x_i$ )
1	40	0.5
2	41	1.0
3	43	1.5
4	42	2.0
5	44	2.5
6	42	3.0
7	43	3.5
8	42	4.0

Let's initialize the data first!

```
alloy <- data.frame(
  durability = c(40, 41, 43, 42, 44, 42, 43, 42),
  temperature = c(0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0)
)
```

(a) Fit the straight line model  $Y = \beta_0 + \beta_1 x + \varepsilon$  using the method of least squares.

From class, it was shown that the minimization of the function

$$Q(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

yielded the following solutions for estimates of  $\beta_1$  and  $\beta_0$ :

$$\hat{\beta}_1 = \frac{\sum x_i y_i - n \bar{x} \bar{y}}{\sum x_i^2 - n \bar{x}^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

From the data provided, it can be found that:

- $n = 8$
- $\bar{x} = 2.25$
- $\bar{y} = 42.125$
- $\sum x_i y_i = 764$
- $\sum x_i^2 = 51$

Plugging these values into the equations for our estimates above, we obtain that

$$\hat{\beta}_1 = 0.548, \quad \hat{\beta}_0 = 40.893.$$

Therefore the equation of the fitted line is:

$$\widehat{\text{Durability}} = 40.893 + 0.548 * \text{Temperature}.$$

We can verify these results in **R**.

```
alloy_lm1 <- lm(durability ~ temperature, data=alloy)

coef(alloy_lm1)
```

```
## (Intercept) temperature
## 40.892857 0.547619
```

- (b) Fit the parabola model  $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$  using the method of least squares.

In class, it was shown that the normal equations in the case of fitting a parabola are:

$$\begin{aligned} n\beta_0 + \beta_1 \sum x_i + \beta_2 \sum x_i^2 &= \sum y_i \\ \beta_0 \sum x_i + \beta_1 \sum x_i^2 + \beta_2 \sum x_i^3 &= \sum x_i y_i \\ \beta_0 \sum x_i^2 + \beta_1 \sum x_i^3 + \beta_2 \sum x_i^4 &= \sum x_i^2 y_i \end{aligned}$$

For the given data, the normal equations take the form:

$$8\beta_0 + 18\beta_1 + 51\beta_2 = 337$$

$$18\beta_0 + 51\beta_1 + 162\beta_2 = 764$$

$$51\beta_0 + 162\beta_1 + 548.25\beta_2 = 2167.5$$

We can solve this system of linear equations as follows:

```
X <- matrix(
  c(8, 18, 51, 18, 51, 162, 51, 162, 548.25),
  nrow=3, ncol=3, byrow=TRUE
)

Y <- matrix(c(337, 764, 2167.5))

solve(X, Y)
```

```
## [1,]
## [1,] 38.4821429
## [2,] 3.4404762
## [3,] -0.6428571
```

Our parameter estimates are as follows:

$$\hat{\beta}_0 = 38.482, \quad \hat{\beta}_1 = 3.440, \quad \hat{\beta}_2 = -0.643.$$

Therefore, the equation of the parabola obtained by least squares is:

$$\widehat{\text{Durability}} = 38.482 + 3.440 * \text{Temperature} - 0.643 * \text{Temperature}^2$$



Note that the `solve()` function in **R** has two uses. If we supply one matrix to `solve()`, it computes the matrix inverse (if it exists). If we supply two matrices,  $X$  and  $Y$ , to `solve()`, it solves the system  $XB = Y$ , for  $B$ .


There are a few ways that we can go about fitting this model in **R**. My approach will involve creating an extra column in my data set for the square of the temperature and then passing this new data set to `lm()`.

```
alloy <- alloy |>
  transform(temperature_sq = temperature^2)

alloy_lm2 <- lm(durability ~ temperature + temperature_sq, data=alloy)

coef(alloy_lm2)
```

```
##      (Intercept)      temperature temperature_sq
##      38.4821429      3.4404762      -0.6428571
```

- (c)  Sketch on the same graph the eight data points and their predicted values according to the two different models.

To construct our plot, I will take the newest alloy data set, as it contains the degree 1 and degree 2 temperature values, and augment it to contain the fitted values from the straight line model and the parabola model. To obtain the predictions from each model, we can pass the model to the `predict()` function. When using `predict()` without supplying a value to the `newdata` argument, it returns the predictions using the predictor values found in the original data. Since `predict()` returns a vector, we can create new columns in our data set via `transform()`.

To augment our data set, we can use the following code:

```
alloy_augment <- alloy |>
  transform(
    .fitted_lm1 = predict(alloy_lm1),
    .fitted_lm2 = predict(alloy_lm2)
  )

alloy_augment
```

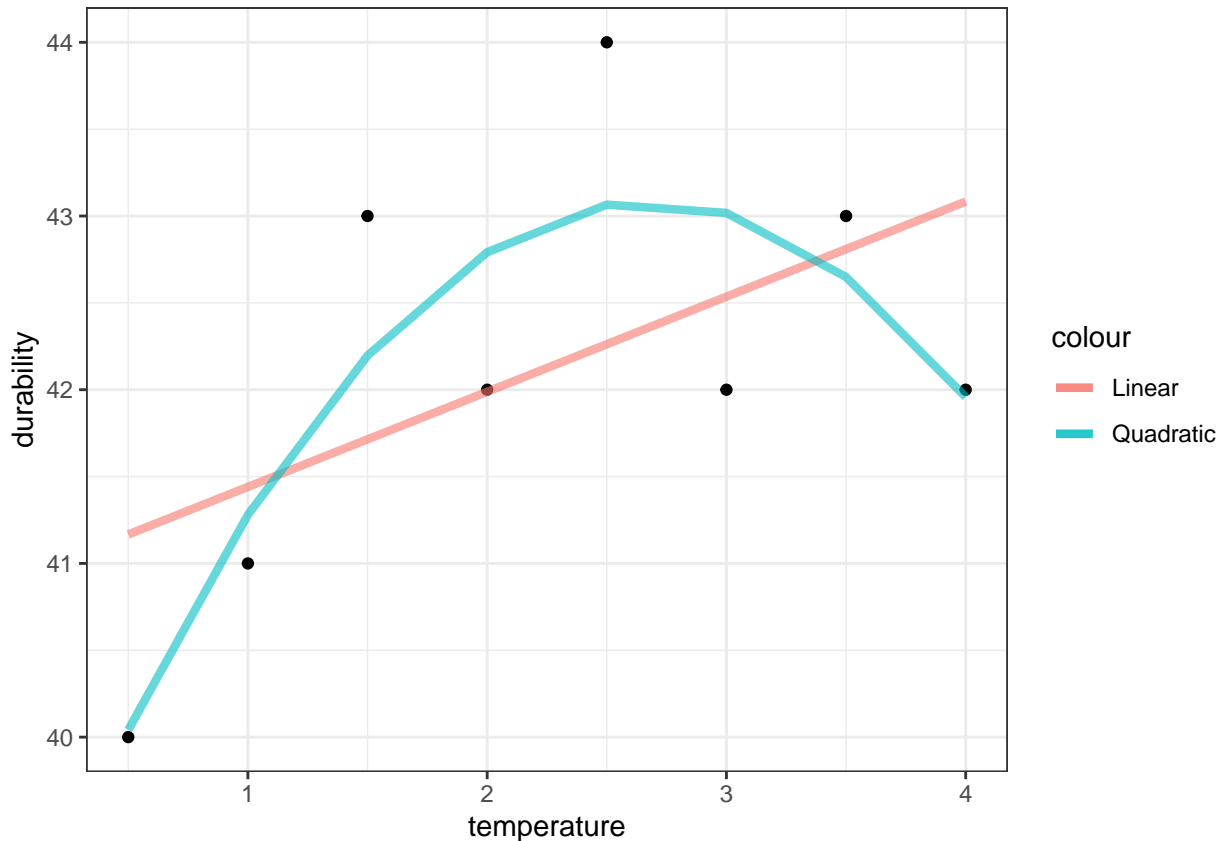
	durability	temperature	temperature_sq	.fitted_lm1	.fitted_lm2
## 1	40	0.5	0.25	41.16667	40.04167
## 2	41	1.0	1.00	41.44048	41.27976
## 3	43	1.5	2.25	41.71429	42.19643
## 4	42	2.0	4.00	41.98810	42.79167
## 5	44	2.5	6.25	42.26190	43.06548
## 6	42	3.0	9.00	42.53571	43.01786
## 7	43	3.5	12.25	42.80952	42.64881
## 8	42	4.0	16.00	43.08333	41.95833

We can sketch the required graph in `ggplot2` using the following code:

(Try running the code line by line to see how the plot is constructed by the overlaying of individual layers!)

```
ggplot(alloy_augment, aes(x=temperature)) +
  geom_point(aes(y=durability)) +
  geom_line(aes(y=.fitted_lm1, colour="Linear"), size=1.5, alpha=0.6) +
```

```
geom_line(aes(y=.fitted_lm2, colour="Quadratic"), size=1.5, alpha=0.6)
```

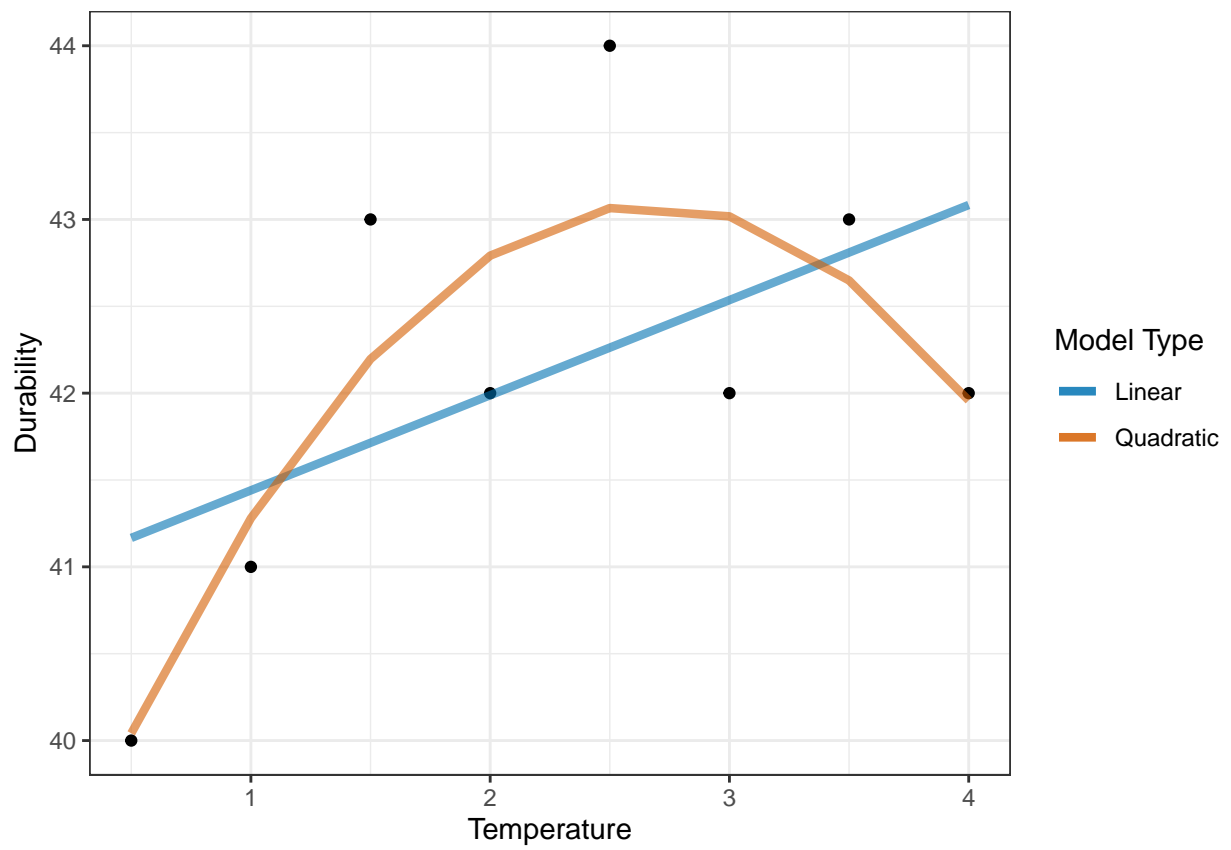


Note that in the above example, we supply a string to the `colour` aesthetic in the `geom_line()` layers. This essentially tags the points on the respective lines such that points with different tags will be coloured differently.

We can get a bit fancier by applying a custom colour scale layer. In the call to this custom colour scale layer, we supply a name to the colour scale (which will show up in the legend), a vector of colours, and optionally (not shown here) a vector of the order of the groupings to appear in the legend, supplied to the `breaks` argument (note that groups are ordered alphabetically by default).

I will also use `labs()` to adjust the axis titles – I like my axis titles to begin with a capital letter.

```
ggplot(alloy_augment, aes(x=temperature)) +
  geom_point(aes(y=durability)) +
  geom_line(aes(y=.fitted_lm1, colour="Linear"), size=1.5, alpha=0.6) +
  geom_line(aes(y=.fitted_lm2, colour="Quadratic"), size=1.5, alpha=0.6) +
  scale_colour_manual(name="Model Type", values=c("#0072B2", "#D55E00")) +
  labs(x="Temperature", y="Durability")
```



From the plot above, it appears that there is a quadratic trend in the points. As such, the quadratic fit does a better job at capturing the overall trend in the data.