e) not possible

f) $\frac{1}{y} = 1 + e^{a+bx}$, so $\log(\frac{1}{y} - 1) = \log(\frac{1-y}{y}) = a + bx$, so $g(y) = \log(\frac{1-y}{y})$, $f(x) = x$, $\beta_0 = a$, and $\beta_1 = b$.

g) $\frac{100}{y} = 1 + e^{a+bx}$, so $\log(\frac{100}{y} - 1) = \log(\frac{100-y}{y}) = a + bx$, so $g(y) = \log(\frac{100-y}{y})$, $f(x) = x$, $\beta_0 = a$, and $\beta_1 = b$.

8.5 Moving up the ladder will spread the larger values more than the smaller values, resulting in a distribution that is right skewed.

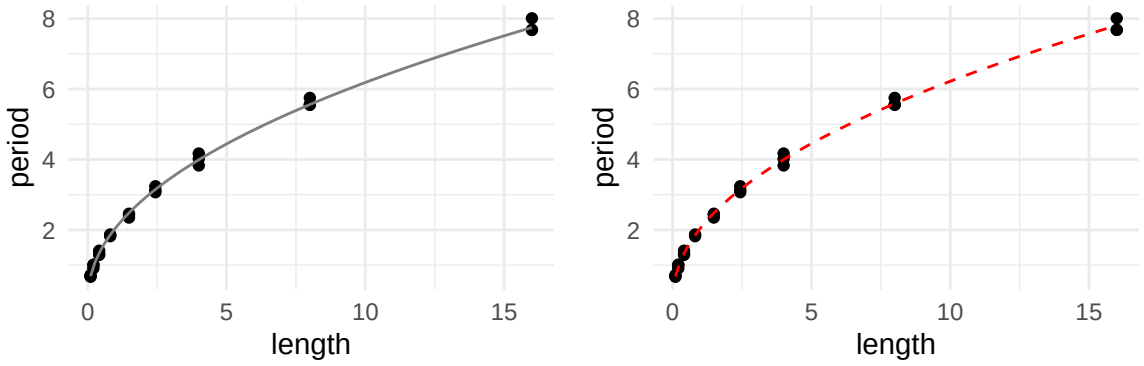
8.6 At first glance, the two models might appear equally good. In each case the power is a bit below 2 and the fits look good on top of the raw data.

```
model <- lm(log(period) ~ log(length), data = Pendulum)
model2 <- nls(period ~ A * length^power, data = Pendulum, start = list(A = 1, power = 2))
f <- makeFun(model)
g <- makeFun(model2)
gf_point(period ~ length, data = Pendulum) %>%
  gf_fun(f(x) ~ x, col = 'gray50')
gf_point(period ~ length, data = Pendulum) %>%
  gf_fun(g(x) ~ x, col = 'red', lty = 2)
coef(summary(model))
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   0.7207   0.006361   113.3 2.027e-35
## log(length)   0.4785   0.003938   121.5 3.536e-36
```

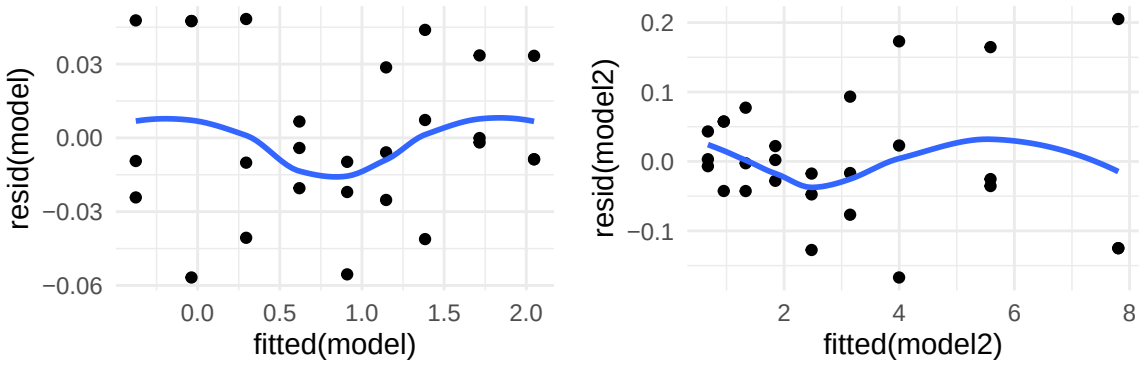
```
coef(summary(model2))
```

```
##       Estimate Std. Error t value  Pr(>|t|)
## A       2.0470   0.022027   92.93 2.844e-33
## power   0.4827   0.004829   99.96 4.611e-34
```



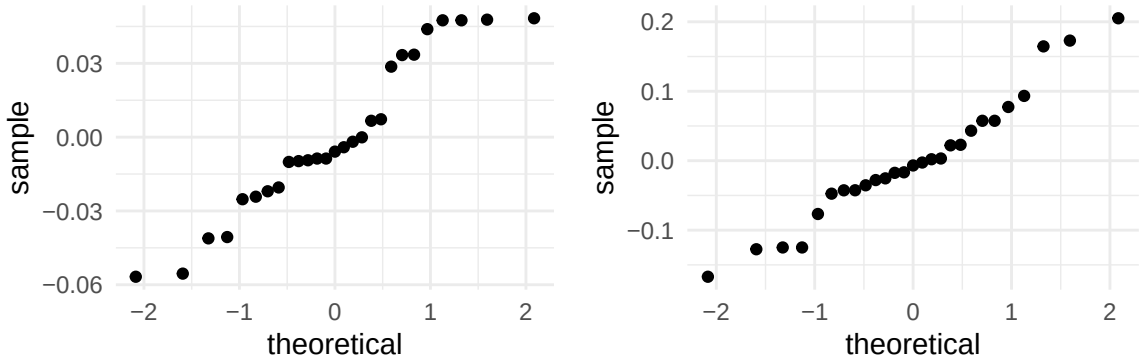
But if we look at the residuals, we see that the linear model is clearly better in this case. The non-linear model suffers from heteroskedasticity.

```
gf_point(resid(model) ~ fitted(model)) %>% gf_smooth()

## `geom_smooth()` using method = 'loess'

gf_point(resid(model2) ~ fitted(model2)) %>% gf_smooth()

## `geom_smooth()` using method = 'loess'
```



Both residual distributions are reasonably close to normal, but not perfect. In the ordinary least squares model, the largets few residuals are not as large as we would expect.

```
gf_qq( ~ resid(model))
gf_qq( ~ resid(model2))
```

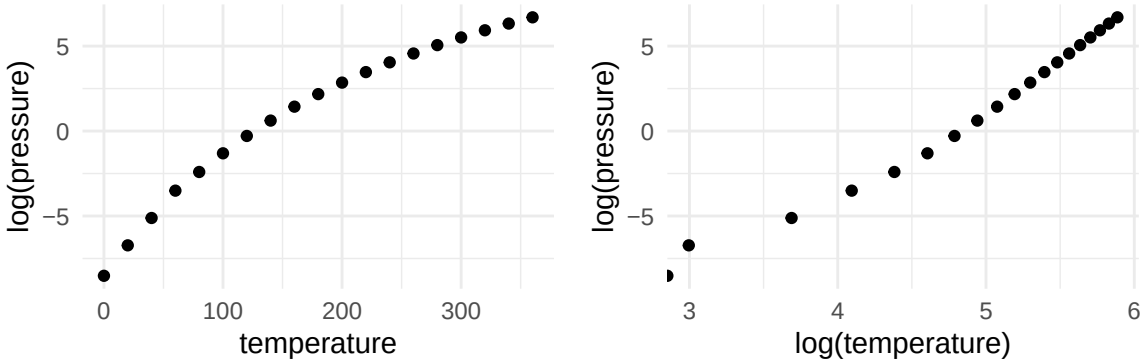


The residuals in the non-linear model show a clear change in variance as the fitted value increases. This is counteracted by the logarithmic transformation of the explanatory variable. (In other cases, the non-linaer model might have the preferred residual distribution.)

The estimated power based on the linear model is 0.478 ± 0.004.

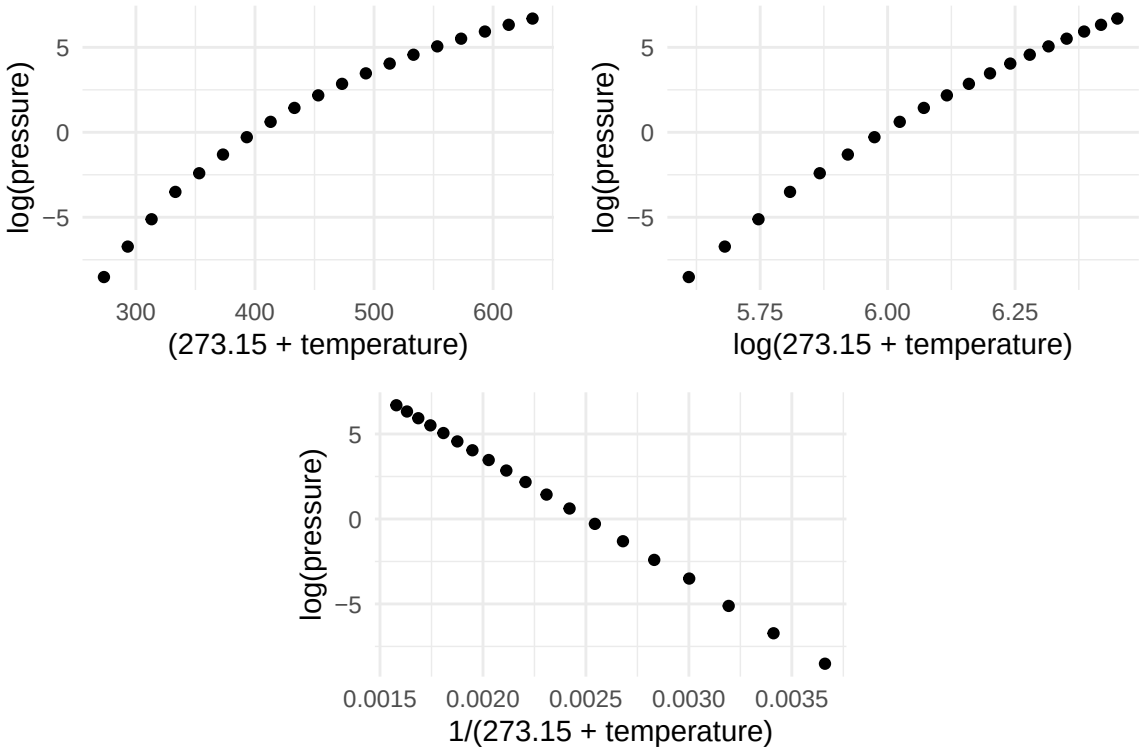
8.7 Using Tukey's buldge rules it is pretty easy to land at something like one of the following

```
gf_point(log(pressure) ~ temperature, data = pressure)
gf_point(log(pressure) ~ log(temperature), data = pressure)
```



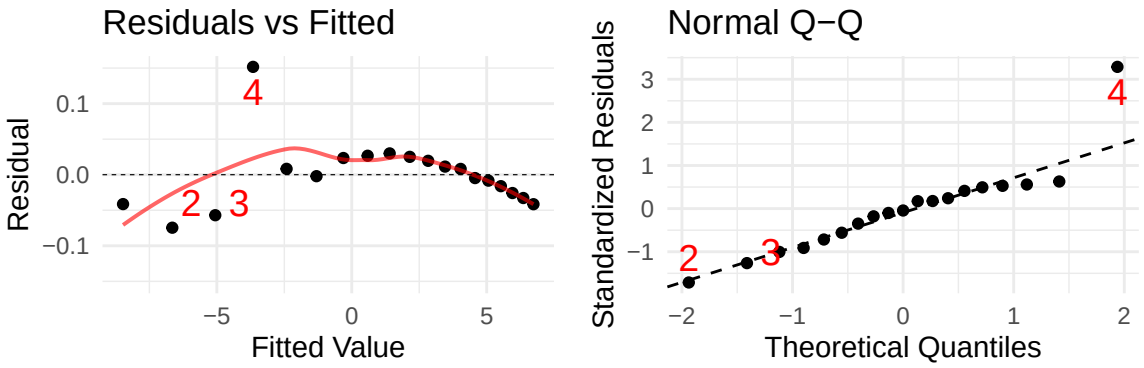
Neither of these is pefect (although both are much more linear than the original untransformed relationship). But if you think in degrees Kelvin instead, you might find a much better transformation.

```
gf_point(log(pressure) ~ (273.15 + temperature), data = pressure)
gf_point(log(pressure) ~ log(273.15 + temperature), data = pressure)
gf_point(log(pressure) ~ 1/(273.15 + temperature), data = pressure)
```



Ah, that last one looks quite good. Let's try that one.

```
model <- lm(log(pressure) ~ I(1/(273.15 + temperature)), data = pressure)
mplot(model, 1:2)


## [[1]]


## `geom_smooth()` using formula 'y ~ x'


##
## [[2]]
```



Things still aren't perfect. There's a bit of a bow in the residual plot, but the size of the residuals is quite small relative to the scale of the log-of-pressure values:

```
range( ~ resid(model))


## [1] -0.07456  0.15171


range( ~ log(pressure), data = pressure)


## [1] -8.517  6.692
```

Also the residual for the fourth observation is quite a bit larger than the rest.

8.9

a) We can build a confidence interval for the mean by fitting a model with only an intercept term.

```
grades <- ACTgpa
confint(lm(ACT ~ 1, data = grades))


##             2.5 % 97.5 %
## (Intercept) 24.25   27.9
```

But this isn't the only way to do it. Here are some other ways.

```
# here's another way to do it; but you don't need to know about it
t.test(grades$ACT)

##
##  One Sample t-test
##
## data:  grades$ACT
## t = 29, df = 25, p-value <2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  24.25 27.90
## sample estimates:
## mean of x
##     26.08

# or you can do it by hand
x.bar <- mean( ~ ACT, data = grades); x.bar

## [1] 26.08

n <- nrow(grades); n

## [1] 26

t.star <- qt(.975, df = n-1); t.star

## [1] 2.06

SE <- sd( ~ ACT, data = grades) / sqrt(n); SE

## [1] 0.8874

ME <- t.star * SE; ME

## [1] 1.828
```

So the CI is 26.077 ± 1.828. Of course, that is too many digits, we should do some rounding to 26.1 ± 1.8.

b)
```
confint(lm(GPA ~ 1, data = grades))

##             2.5 % 97.5 %
## (Intercept) 3.185  3.579

# this could also be done the other ways shown above.
```

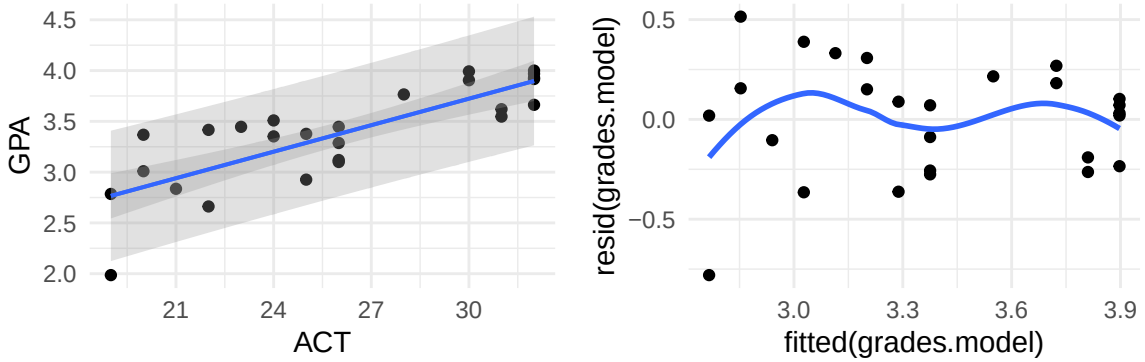
c)
```
grades.model <- lm(GPA ~ ACT, data = grades)
f <- makeFun(grades.model)
f(ACT = 25, interval = "confidence")

##     fit   lwr  upr
## 1 3.288 3.167 3.41
```

d)
```
f(ACT = 30, interval = "prediction")
```

```
##     fit   lwr   upr
## 1 3.723 3.101 4.346
```

e)
```
gf_point(GPA ~ ACT, data = grades) %>%
  gf_lm(interval = "confidence") %>%
  gf_lm(interval = "prediction")
gf_point(resid(grades.model) ~ fitted(grades.model)) %>% gf_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```



There are no major concerns with the regression model. The residuals look pretty good. (There is perhaps a bit more variability in GPA for the lower ACT scores and if you said you were worried about that, I would not argue.)

The prediction intervals are very wide and hardly useful, however. It's pretty hard to give a precise estimate for an individual person – there's just too much variability from preson to person, even among people with the same ACT score.

8.10 The best of these four models is a model that says drag is proportional to the square of velocity. Given the design of the experiment, it makes the most sense to fit velocity as a function of drag force. Here are several ways we could do the fit:

```
model1 <- lm(velocity^2 ~ force.drag, data = Drag)
model2 <- lm(velocity ~ sqrt(force.drag), data = Drag)
model3 <- lm(log(velocity) ~ log(force.drag), data = Drag)
```

```
coef(summary(model1))
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -0.06227   0.221931 -0.2806 7.805e-01
## force.drag   0.08400   0.002321 36.1966 3.449e-32
```

```
coef(summary(model2))
```

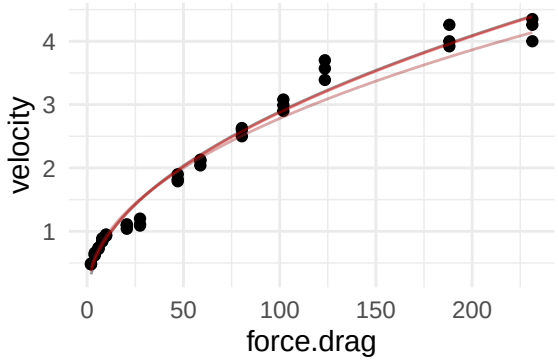
```
##                   Estimate Std. Error t value  Pr(>|t|)
## (Intercept)      -0.03586   0.054832 -0.6539 5.169e-01
## sqrt(force.drag)  0.29098   0.006807 42.7478 5.239e-35
```

```
coef(summary(model3))
```

```
##                 Estimate Std. Error t value  Pr(>|t|)
## (Intercept)      -1.1623    0.04540  -25.60 2.021e-26
## log(force.drag)   0.4745    0.01241   38.23 4.104e-33
```

Note that `model1`, `model2`, and `model3` are not equivalent, but they all tell roughly the same story.

```
f1 <- makeFun(model1)
f2 <- makeFun(model2)
f3 <- makeFun(model3)
gf_point(velocity ~ force.drag, data = Drag) %>%
  gf_fun(sqrt(f1(x)) ~ x, alpha = .4) %>%
  gf_fun(f2(x) ~ x, alpha = .4, color = 'red') %>%
  gf_fun(f3(x) ~ x, alpha = .4, color = 'brown')
```



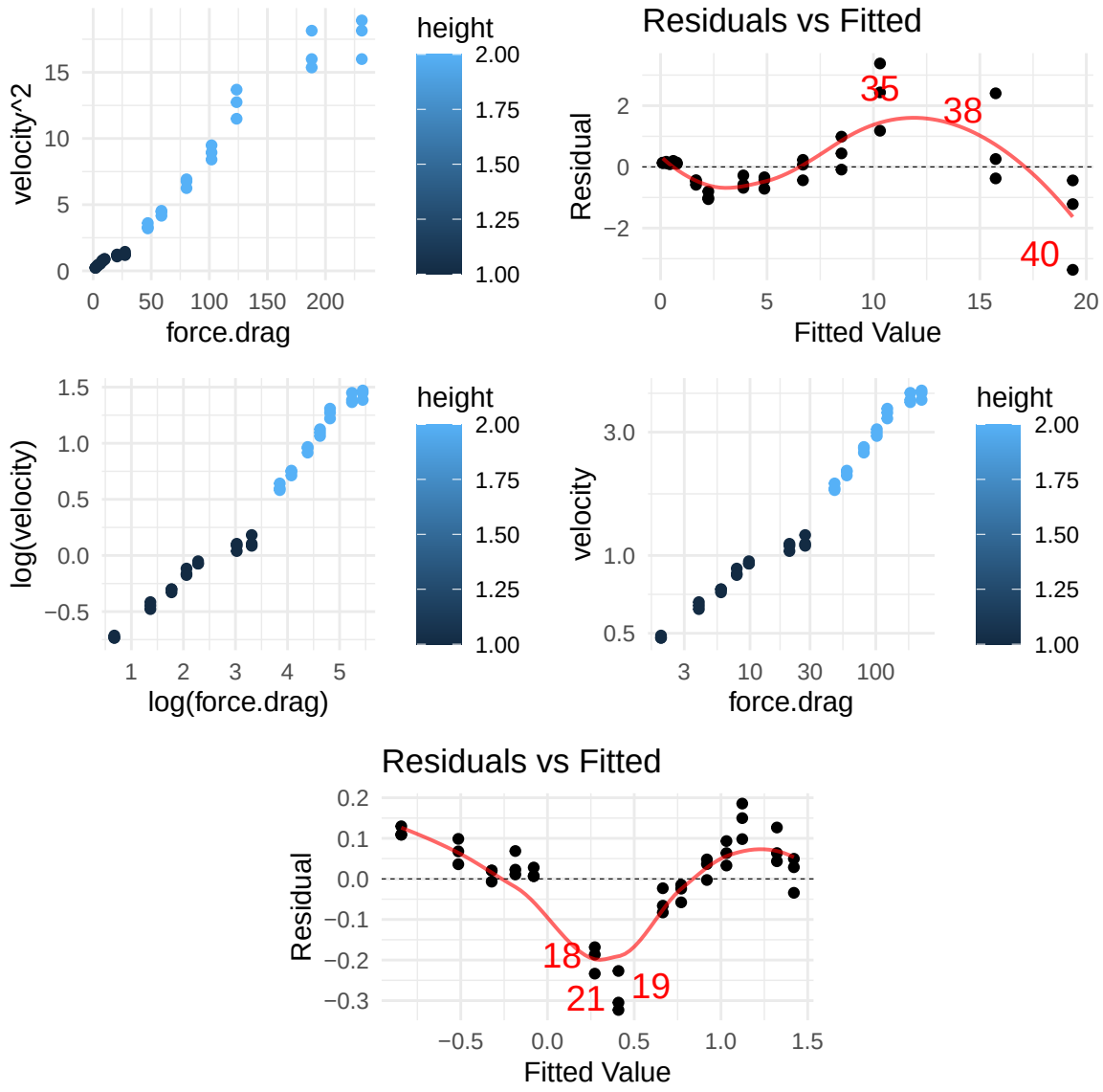
The fit for these models reveals some potential errors in the design of this experiment. Separating out the data by the height used to determine velocity suggests that perhaps some of the velocity measurements are not yet at terminal velocity. In both groups, the velocities for the greatest drag forces are not as fast as the pattern of the remaining data would lead us to expect.

```
gf_point(velocity^2 ~ force.drag, data = Drag, color = ~ height)
mplot(model1, w = 1)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
gf_point(log(velocity) ~ log(force.drag), data = Drag, color = ~ height)
gf_point(velocity ~ force.drag, data = Drag, color = ~ height) %>%
  gf_refine(scale_x_log10(), scale_y_log10())
mplot(model3, w = 1)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



8.11 See previous problem.

9.1

a) The normal-quantile plot looks pretty good for a sample of this size. The residual plot is perhaps not quite as "noisy" as we would like (the first few residuals cluster above zero, then next few below), but it is not terrible either. Ideally we would like to have a larger data set to see whether this pattern persists or whether things look more noisy as we "fill-in" with more data.

b) A is a measure of background radiation levels, it is the amount of clicks we would get if our test substance were "at infinity", i.e., so far away (or beyond some shielding) that it does not affect the Geiger counter.

c) 114 ± 11

d) 5.5×10^{-6}. This gives strong evidence that there is some background radiation being measured by the Geiger counter.

e) k measures the rate at which our test substance is emitting radioactive particles. If k is 0, then our substance is not radioactive (or at least not being detected by the Geiger counter).

f) 31.5 ± 1.0

g) 1.1×10^{-9}. We have strong evidence that our substance is decaying and contributing to the Geiger counter clicks.

h) $H_0 : k = 29.812$; $H_a : k \neq 29.812$

```
t <- (31.5 - 29.812) / 1.0; t      # test statistic

## [1] 1.688

2 * (1 - pt(t, df = 8))            # p -value

## [1] 0.1299
```

Conclusion. With a p-value this large, we cannot reject the null hypothesis. Our data are consistent with the hypothesis that our test substance is the same as the standard substance.

9.2 We can use the product of conditional probabilities:

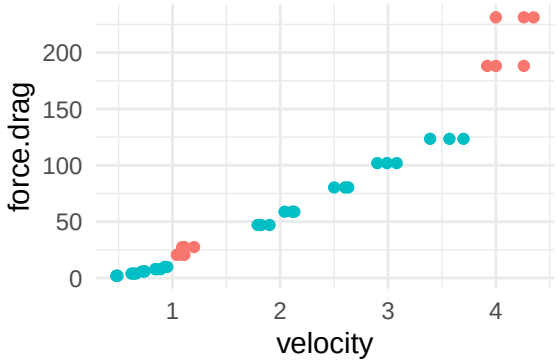
$$frac14 \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{1} = \frac{1}{24}$$

Since the probability of guessing the first glass correctly is $\frac{1}{4}$, the probability of guess the second correctly – assuming the first was correctly guessed – is $\frac{1}{3}$; the probability of guessing the third correctly – assuming the first two were correctly guessed – is $\frac{1}{2}$; and if the first three are guessed correctly, the last is guaranteed to be correct.

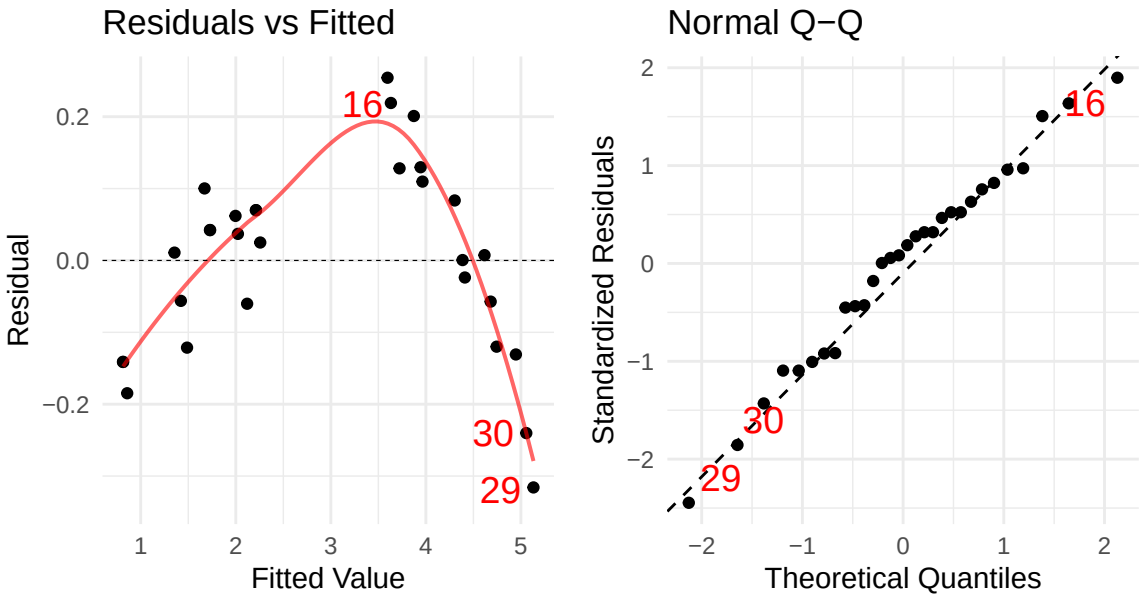
Altnernative solution: The wines can be presented in $4 \cdot 3 \cdot 2 \cdot 1 = 24$ different orders, so the probability of guessing correctly is $1/24$.

9.3 Let's remove the fastest values at each height setting. Although they are the fastest, it appears that terminal velocity has not yet been reached. At least, these points would fit the overall pattern better if the velocity were larger.

```
gf_point( force.drag ~ velocity, data = Drag,
          show.legend = FALSE,
  color = ~ (velocity < 3.9) & !(velocity > 1 & velocity < 1.5)) %>%
  gf_theme(legend.position = "top")
Drag2 <- Drag %>% filter((velocity < 3.9) & !(velocity > 1 & velocity < 1.5))
```




```
drag2.model <- lm(log(force.drag) ~ log(velocity), data = Drag2)
summary(drag2.model)


##
## Call:
## lm(formula = log(force.drag) ~ log(velocity), data = Drag2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.3158 -0.1050  0.0179  0.0801  0.2542
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.3659     0.0272    86.9   <2e-16
## log(velocity)   2.1142     0.0368    57.5   <2e-16
##
## Residual standard error: 0.137 on 28 degrees of freedom
## Multiple R-squared:  0.992,Adjusted R-squared:  0.991
## F-statistic: 3.3e+03 on 1 and 28 DF,  p-value: <2e-16


mplot(drag2.model, w = 1:2)


## [[1]]


## `geom_smooth()` using formula 'y ~ x'


##
## [[2]]
```



The model is still not as good as we might like, and it seams like the fit is different for the heavier objects than for the lighter ones. This could be due to some flaw in the design of the experiment or because drag force actually behaves differently at low speeds vs. higher speeds. Notice the data suggest an exponent on velocity that is just a tiny bit larger than 2:

```
confint(drag2.model)
```

```
##                2.5 % 97.5 %
## (Intercept)    2.310  2.422
## log(velocity) 2.039  2.190
```

So our data (for whichever reason, potentially still due to design issues) is not compatible with the hypothesis that this exponent should be 2.

9.4

```
SE <- 1.2 / sqrt(16); SE
```

```
## [1] 0.3
```

```
t <- (94.32 - 95) / SE; t
```

```
## [1] -2.267
```

```
p_val <- 2 * pt(t, df = 15); p_val     # p-value
```

```
## [1] 0.03863
```

95 will be outside the confidence interval because the p-value is less than 0.05. As a double check, here is the 95% confidence interval.

```
t_star <- qt(.975, df = 11); t_star

## [1] 2.201

94.32 + c(-1,1) * t_star * SE

## [1] 93.66 94.98
```

9.5

a) In this situation it makes sense to do a one-sided test since we will reject the alloy only if the amount of expansion is to high, not if it is too low.

```
SE <- 5.9/sqrt(42); SE

## [1] 0.9104

t <- (73.1 - 75) / SE; t

## [1] -2.087

pt(t, df = 41)

## [1] 0.02157
```

b) Corresponding to a 1-sided test with a significance level of $\alpha = 0.01$, we could make a 98% confidence interval (1% in each tail). The upper end of this would be the same as for a 1-sided 99% confidence interval

```
t_star <- qt(0.99, df = 41)
73.1 + c(-1, 1) * t_star * SE

## [1] 70.9 75.3
```

c) If all we are interested in is a decision, the two methods are equivalent. I confidence interval might be easier to explain to someone less familiar with statistics.