Computational Cognitive Science 2026/27

Week 5

Frank Keller Chris Lucas

Solutions for Tutorial Tutorial 3: Model Comparison à la XKCD

Our data in this tutorial comes from a visual motion perception task (Karvelis et al., 2018): participants saw either moving dots or no stimuli for 3000 milliseconds, then reported the motion of direction, or an absence of stimuli if no dots were present. The independent variable `RISC` is a scale of schizotypical traits, measured on the healthy participants in the study. The dependent variable `false.detections` is a count of the number of trials in which participants falsely detected (i.e. hallucinated) stimuli — that is, they reported that moving dots were present when no dots were shown on that trial. For more details, see the original paper.

Consider an initial sample of 62 randomly selected participants (perhaps the first batch of subjects tested). We'll load in their data for analysis.

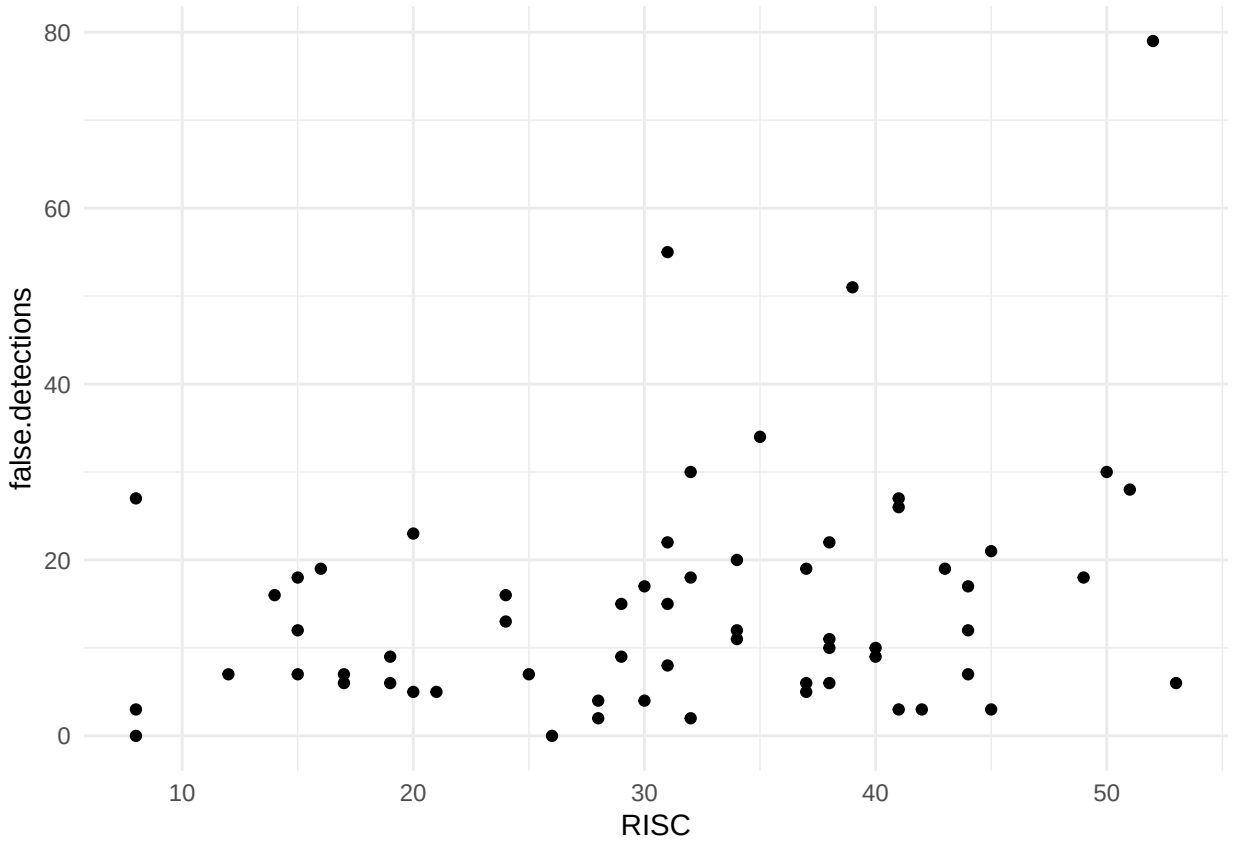
```
data <- read.csv("sample1.csv")
```

Inspired by Randall Munroe's XKCD, we will compare the fit of various models to some real-world data. As before, we'll use ggplot to visualize.

```
library(ggplot2)

base_plot <- ggplot(data) + # pass data to ggplot
  aes(RISC, false.detections) + # tell it what to plot on x and y axes respectively
  geom_point() + # add scatterplot layer
  theme_minimal() # nicer-looking style

base_plot
```



If you would like to get the full XKCD feel, you can optionally install the xkcd package along with an XKCD font for a bit more verisimilitude in plotting! To do so, set `eval=TRUE` below. If you don't want to, all the later code will run just fine without it.

```
# check for xkcd package, install if missing
xkcd_available <- require("xkcd")
if (!xkcd_available) {
  install.packages("xkcd", repos="http://cran.uk.r-project.org")
  xkcd_available <- require("xkcd")
}

# Note: xkcd_available will be false if the installation didn't work,
# in which case we'll have to fall back to base_plot
if(xkcd_available) {

  # check for xkcd font, install if missing
  if (!('xkcd' %in% fonts())) {
    download.file("http://simonsoftware.se/other/xkcd.ttf",
                  dest="xkcd.ttf", mode="wb")
    system("mkdir  ~/.fonts")
    system("cp xkcd.ttf  ~/.fonts")
    #system("cp xkcd.ttf  /Library/Fonts") # uncomment if you're running on a Mac
    font_import(paths=c("~/.fonts", "/Library/Fonts"), pattern = "[X/x]kcd", prompt=FALSE)
    loadfonts()
  }

  # add xkcd layers to the plot we defined above
  base_plot <- base_plot +
```

```
    xkcdaxis(range(data$RISC), range(data$false.detections)) + # make plot axes look xkcd-ish
    theme(text=element_text(family="xkcd", size=14)) # use xkcd font
}

# Here it is again, in XKCD style
base_plot
```

Nested models

First, we'll fit the classic linear model: $y = \beta_1 x + \beta_0 + \epsilon$.

```
# for more details on R's linear model function:
# ?lm
lm_fit <- lm(false.detections ~ RISC, # formula: y ~ x
             data=data)
# intercept term is assumed, doesn't need to be specified
# so actual formula is y ~ x + 1

summary(lm_fit)
```

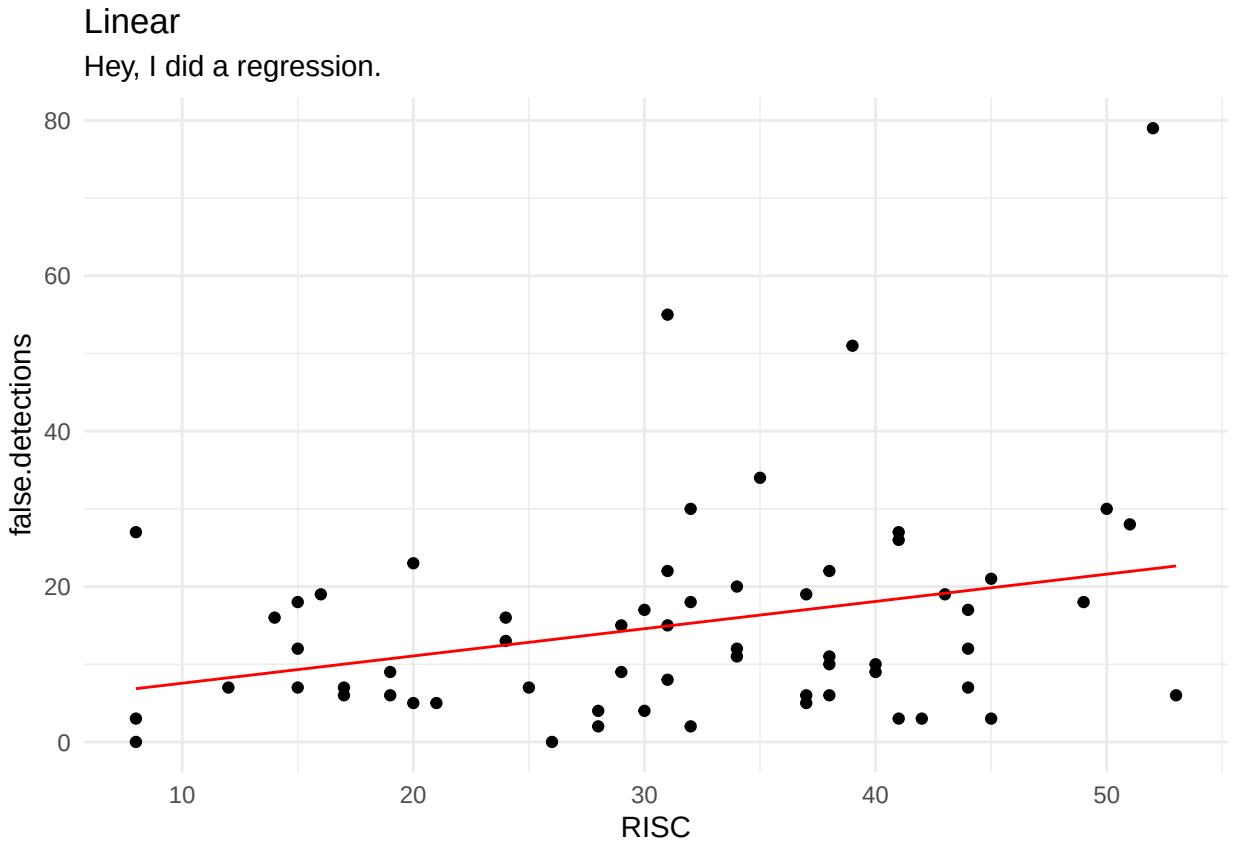
```
##
## Call:
## lm(formula = false.detections ~ RISC, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.838  -7.461  -2.749   4.469  56.705
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.0445     4.8104   0.841   0.4038
## RISC          0.3510     0.1439   2.438   0.0177 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.26 on 60 degrees of freedom
## Multiple R-squared:  0.09016,    Adjusted R-squared:  0.075
## F-statistic: 5.946 on 1 and 60 DF,  p-value: 0.01773
```

Look over the summary output. What are the estimated values of the coefficients β_0 and β_1? What are the standard errors? Does this look like a reasonable fit to you?

We can define a helper function to plot the model's predictions.

```
plot_preds <- function(preds, title="", subtitle="") {
  (base_plot + geom_line(aes(y=preds), col="red") + ggtitle(title, subtitle))
}

plot_preds(fitted(lm_fit), # 'fitted' extracts model's predictions for training data
           "Linear", "Hey, I did a regression.")
```



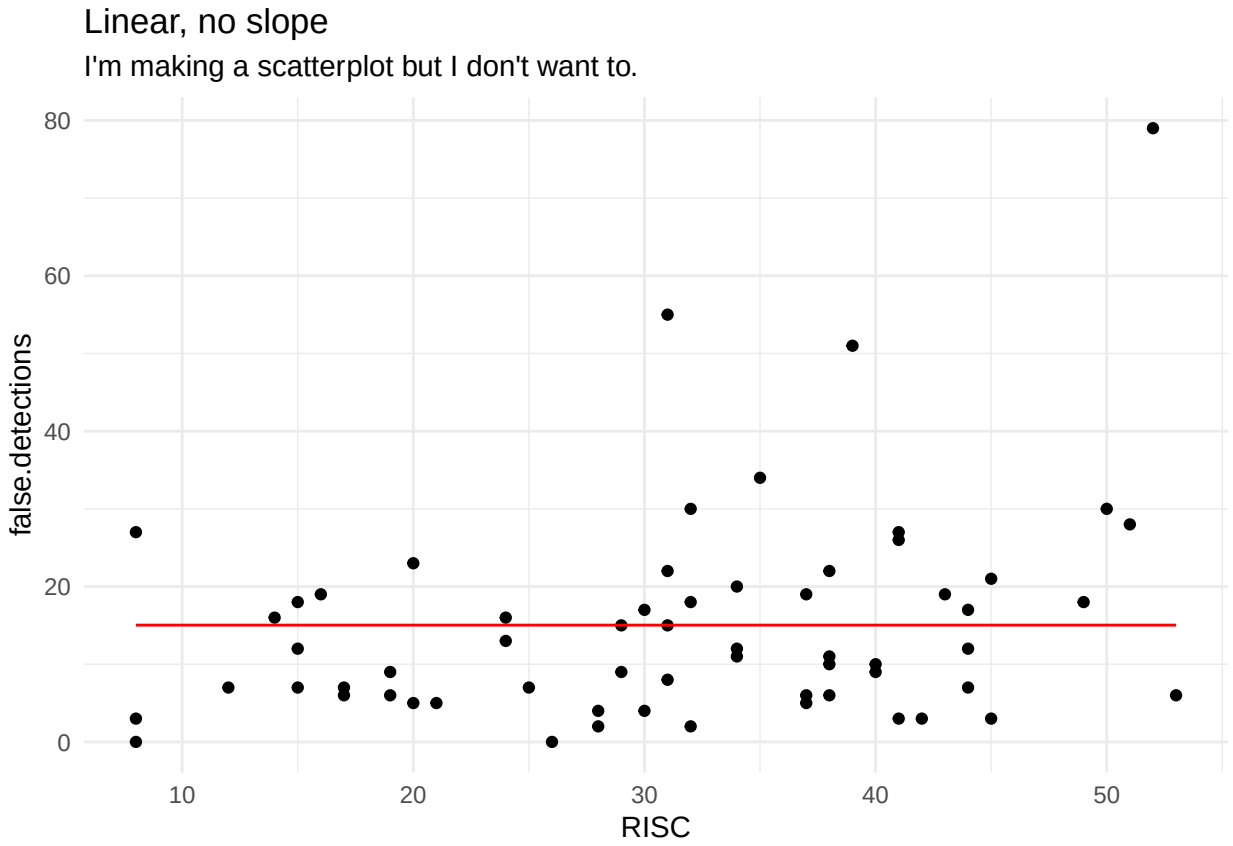
As noted in the original comic, we can also just fit the intercept term, β_0, to get a linear model with no slope.

```
intercept_fit <- lm(false.detections ~ 1, data=data)

summary(intercept_fit)
```

```
##
## Call:
## lm(formula = false.detections ~ 1, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.032  -9.032  -3.532   3.968  63.968
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   15.032      1.751   8.587 4.33e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.78 on 61 degrees of freedom
```

```
plot_preds(fitted(intercept_fit),
           "Linear, no slope", "I'm making a scatterplot but I don't want to.")
```



Question 1 Modify the model-fitting code above to fit an order-9 polynomial: $y = \sum_{i=0}^{9} \beta_i x^i + \epsilon$.

Solution:

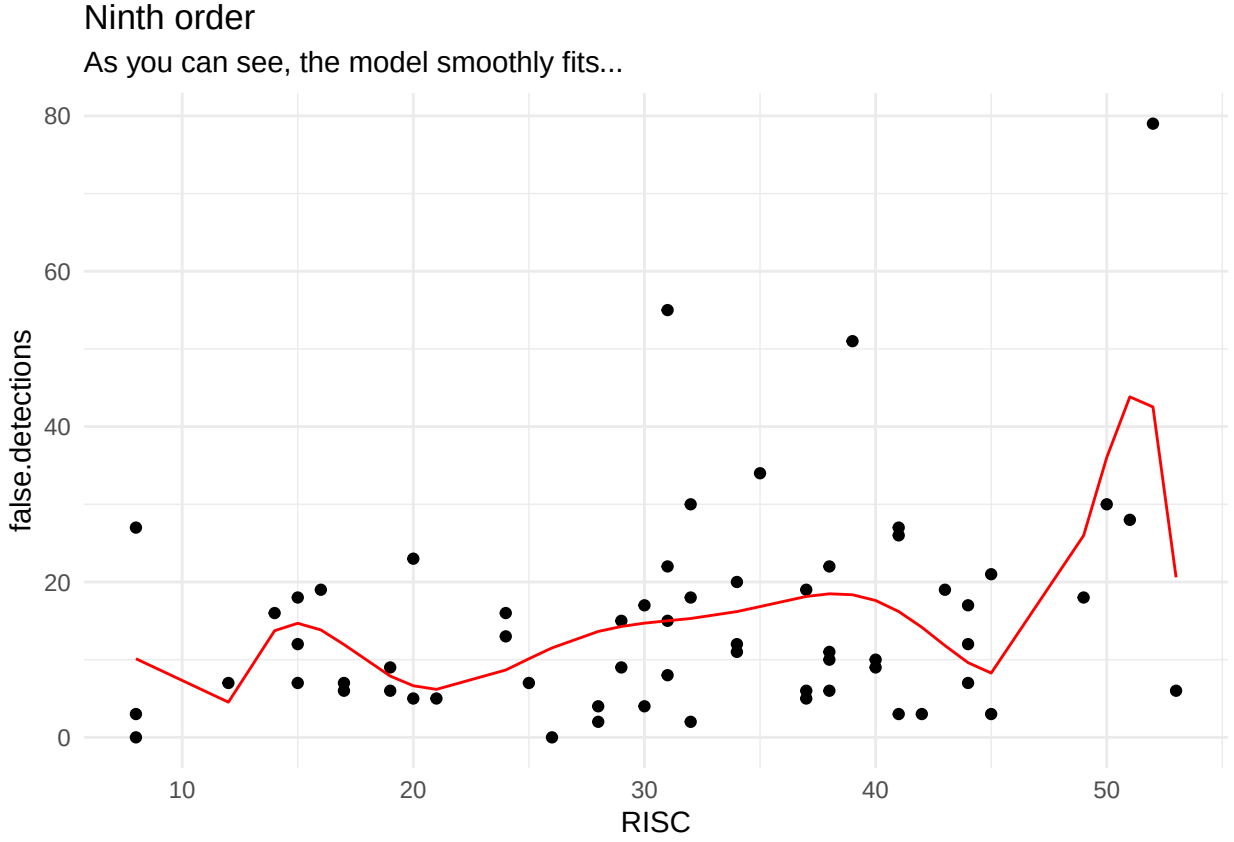
```
# a straightforward solution:
# ninth_fit <- lm(false.detections ~ RISC^9+RISC^8+RISC^7+RISC^6+RISC^5+RISC^4+RISC^3+RISC^2+RISC, data
# R also has a convenience function for polynomials
ninth_fit <- lm(false.detections ~ poly(RISC, 9), data=data)

summary(ninth_fit)
```

```
##
## Call:
## lm(formula = false.detections ~ poly(RISC, 9), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.831  -7.908  -2.254   4.201  39.995
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)       15.03       1.62   9.280 1.29e-12 ***
## poly(RISC, 9)1    32.33      12.76   2.534   0.0143 *
## poly(RISC, 9)2    17.25      12.76   1.353   0.1820
## poly(RISC, 9)3    11.46      12.76   0.898   0.3733
## poly(RISC, 9)4    14.32      12.76   1.122   0.2668
## poly(RISC, 9)5    11.82      12.76   0.927   0.3582
## poly(RISC, 9)6   -15.12      12.76  -1.185   0.2413
```

```
## poly(RISC, 9)7   -24.07      12.76  -1.887   0.0647 .
## poly(RISC, 9)8   -10.74      12.76  -0.842   0.4036
## poly(RISC, 9)9   -19.69      12.76  -1.544   0.1288
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.76 on 52 degrees of freedom
## Multiple R-squared:   0.27,  Adjusted R-squared:  0.1437
## F-statistic: 2.137 on 9 and 52 DF,  p-value: 0.04249
```

```
plot_preds(fitted(ninth_fit), "Ninth order", "As you can see, the model smoothly fits...")
```



As these are all nested models, we can compare them using the likelihood ratio test in R's `anova` function.

```
anova(intercept_fit, lm_fit, ninth_fit,
      test="LRT") # specify likelihood ratio test
```

```
## Analysis of Variance Table
##
## Model 1: false.detections ~ 1
## Model 2: false.detections ~ RISC
## Model 3: false.detections ~ poly(RISC, 9)
##   Res.Df     RSS Df Sum of Sq Pr(>Chi)
## 1     61 11589.9
## 2     60 10544.9  1    1045.0  0.01127 *
## 3     52  8460.1  8    2084.8  0.11840
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Question 2 Can you think of any reasons you might not want to use the likelihood ratio test? Are there other approaches you could use to compare nested models?

Solution: *The likelihood ratio test can only be used on nested models, so we can't use it for broader model comparison, but that isn't an issue in this case. In general, the LRT uses null hypothesis significance testing (as shown in the anova result above:* `Pr(>Chi)` *is the p-value, here nonsignificant). It indicates whether there is reason to reject a simpler model in favor of a the more complex model. However, we may wish to measure the strength of the evidence for each model given the data – the LRT could fail to reject the simpler model because there is evidence supporting, or because there is an absence of good evidence one way or another. The Savage-Dickey density ratio is a Bayesian alternative that overcomes this limitation.*

AIC & BIC

A more general approach which does not require models to be nested is to use AIC or BIC.

```
models = list(intercept_fit, lm_fit, ninth_fit)
model_AIC = AIC(intercept_fit, lm_fit, ninth_fit)

cbind(model_AIC,
      BIC = sapply(models, BIC),
      log_lik = sapply(models, logLik))
```

```
##               df      AIC      BIC   log_lik
## intercept_fit  2 504.2554 508.5096 -250.1277
## lm_fit         3 500.3970 506.7784 -247.1985
## ninth_fit     11 502.7396 526.1380 -240.3698
```

Question 3 How do AIC and BIC estimate model complexity? What's the difference between the two? Are there cases when you can't use these metrics?

Solution: *Both AIC and BIC use number of parameters as a proxy for model complexity. In general, they differ in how parameters are penalized — AIC weights each parameter by a constant penalty factor of 2, while BIC penalizes each parameter by* `log(N)` *where* `N` *is the number of observations.*

The result is that model complexity is penalized more heavily by BIC, as we can see by comparing the model fits shown here. The 9th-order model has the highest likelihood, and its AIC (502.7) is only slightly higher than the linear model (500.4), and both are lower than then intercept-only model (AIC 504.3). However, the BIC of the 9th-order model (526.1) is much higher than both the intercept-only model (508.5) and the linear model (506.8), reflecting BIC's higher penalty for additional parameters. If we removed the linear model from consideration, the AIC and BIC would point us toward different model choices in this case.

Held out set

Another approach to model comparison (arguably better in some ways) is to focus on predictive accuracy: If we're interested in a model's ability to generalize to unseen data, evaluating on a sequestered set lets us look at prediction directly, without having to penalize more complex models.

Let's load the data from the remaining 20 participants (perhaps the second batch tested), and see how well our models predict their behavior. For simplicity, we'll use mean squared error to evaluate performance here, but in most settings log predictive density would be preferred (Gelman et al., 2014).

```
sample2 <- read.csv("sample2.csv")
```

Question 4 Calculate the mean squared error (MSE) for each model's predictions on the new data. You can use the function `predict`; for example, `preds <- predict(lm_fit, sample2)` gets the linear model's predictions for `sample2`, which you can then use to calculate MSE.) Which one has the lowest mean squared error on the new data?

Solution:

```
mse <- function(model, newdata=sample2) {
  preds <- predict(model, newdata)
  sqe <- (newdata[["false.detections"]]-preds)^2
  sqrt(mean(sqe))
}

sapply(list(intercept_fit, lm_fit, ninth_fit), mse)
```

```
## [1] 15.36170 16.44377 58.61646
```

Cross-validation

We can use *cross-validation* to get a more robust comparison of different models' ability to generalize to unseen data, by taking all the data available and systematically using different subsets to estimate test error.

```
data <- rbind(data, sample2) # combine all our data
```

```
# specify number of splits to partition data into
n_splits = 6

# assign each row to a split
# this assumes the rows are randomly ordered
splits = rep_len(1:n_splits, length.out=nrow(data))

# you can use `splits` to index your data frame
# for example, to test on split 1:
# training_data <- data[splits != 1,]
# test_data <- data[splits == 1,]
```

Question 5 Cross-validate the intercept, linear, and 9th-order models using the splits defined above. Which model has the lowest average test error across splits? Do you get different results if you try different numbers of splits (e.g. by setting `n_splits` to another value)?

Solution:

```
# specify each model as its input formula
formulae <- paste0("false.detections ~ ", c("1", "RISC", "poly(RISC, 9)"))
formulae <- sapply(formulae, as.formula) # R can represent these as 'formula' objects

# function to obtain test split MSE from each model
get_test_error <- function(train, test, f_list=formulae) {
  sapply(f_list, # for each formula (= type of model)
         function(f) {
           model <- lm(f, data=train) # train model
           test_error <- mse(model, test) # get test error
           test_error
  })
}
```

```
# apply function over splits: data not in split = train, data in split = test
test_error <- sapply(1:n_splits, function(s) get_test_error(data[splits != s,], data[splits == s,]))

# get the model with the average test error over all splits
mean_test_error <- apply(test_error,
                         1, # apply function over rows
```

```
                          mean)

# which model has the lowest average error?
mean_test_error
```

```
##             false.detections ~ 1          false.detections ~ RISC
##                          13.39937                         13.40653
## false.detections ~ poly(RISC, 9)
##                          15.50658
```

With the 6 splits defined above, the intercept-only model has the lowest average test error by a tiny margin. In contrast, the linear model comes out on top fairly consistently for larger numbers of splits.

Setting the maximum possible number of splits, i.e. `n_splits = nrow(data)` *or* `n_splits = 82`*, performs leave-one-out cross-validation, where each model is fit on all the data except one point and then evaluated by predicting the held-out data point. In this case, the linear model prevails, although the intercept model shows only slightly worse generalization error, suggesting that we wouldn't do too badly just predicting the mean on future trials. The 9th-order model is quite a bit worse than the others.*

Question 6 Is there anything about the data that suggests RMSE isn't an ideal loss function to use? What might we use instead? Are there downsides to using that alternative?

Solution: *There are many things an answer could focus on; this is intended to provoke a discussion. One thing we might observe is that the data points are not normally distributed; they are all positive and discrete by design and have some extreme values. These extreme values may have an outsized effect on the curve fits. There are many possible alternatives. One might be to do away completely with a descriptive model of the sort we're looking at here and model the process by which false detections relate to RISC. That would be a nice approach, but also a major project. A more modest improvement might be to use a more distributionally appropriate loss function/error model, or to transform our data to make a normal distribution more appropriate (supposing that doesn't interfere with the scientific claims we'd like to evaluate). Doing so might require more computationally expensive and approximate solutions, but these are unlikely to be problematic given the small size and low dimensionality of the data set we're examining.*

Question 7 In light of these analyses, what might be conclude about the relationship between RISC and false detections?

Solution: *First, the 9th order polynomial appears to be fitting noise, and is not a very good model of the underlying relationship. There might be a weak linear relationship, but we do not have good evidence to argue for one; our analyses (using extremely simple models and omitting some potentially useful information) generally agree with the conclusions of the original paper by Karvelis: "...schizotypal traits were found to have no effect on the number of hallucinations (RISC: $r = 0.126$, $p=0.163$". That said, we might want to (1) use a richer and more realistic model; and (2) word our conclusions cautiously, e.g., "schizotypal trains were not found to have linear relationship on the number of hallucinations."*

References

Karvelis, Povilas, Aaron R Seitz, Stephen M Lawrie, and Peggy Seriès (2018). "Autistic Traits, but Not Schizotypy, Predict Increased Weighting of Sensory Information in Bayesian Visual Integration." ELife 7. https://doi.org/10.7554/eLife.34115.

Gelman, Andrew, Jessica Hwang, and Aki Vehtari (2014). "Understanding Predictive Information Criteria for Bayesian Models." Statistics and Computing 24:6: 997–1016. https://doi.org/10.1007/s11222-013-9416-2.