

# Many kinds of errors!

## Class 6 in MATH 615: Approaches to Quantitative Analysis in the Life Sciences

Parsimony revisited — predictive ability — overfitting — cross-validation — multivariate models — multiple candidate models — types of errors — data mining — data reduction

---

### Setup

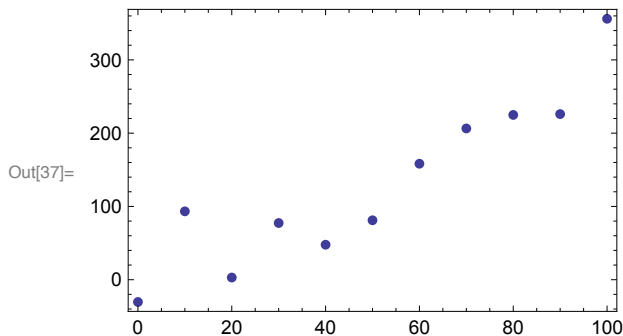
---

#### Overfitting example 1: Choosing a polynomial curve

Here are some data. It looks as if a curved function will be needed. Let's fit a quadratic function (also known as a second-order polynomial — one with a 'squared' term in it).

```
In[36]:= data = Map[{#, 2 + 0.5 * # + 0.03 * #^2 + RandomReal[NormalDistribution[0, 30]]} &, Range[0, 100, 10]];
```

```
ListPlot[ data]
```



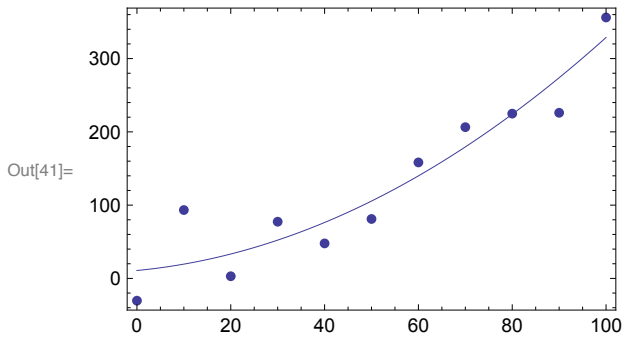
```
In[38]:= lm = LinearModelFit[data, {x, x^2}, {x}];  
lm["ANOVATable"]  
lm[x]
```

Out[39]=

	DF	SS	MS	F-Statistic	P-Value
x	1	111225.	111225.	62.609	0.0000472581
x <sup>2</sup>	1	5668.37	5668.37	3.19074	0.111876
Error	8	14212.1	1776.51		
Total	10	131106.			

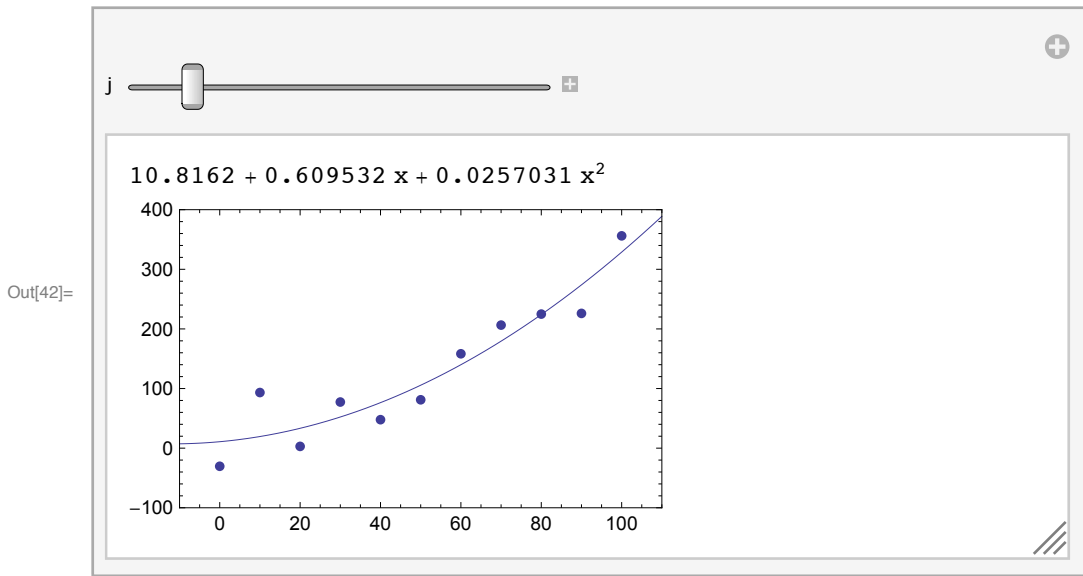
```
Out[40]= 10.8162 + 0.609532 x + 0.0257031 x2
```

```
In[41]:= Show[ListPlot[data], Plot[lm[x], {x, 0, 100}]]
```



It looks like quite a good fit. But could we do better. Let's try some more complicated polynomials.

```
In[42]:= Manipulate[
  lm = LinearModelFit[data, Table[x^i, {i, 1, j}], {x}];
  Column[{lm[x], Show[ListPlot[data], Plot[lm[x], {x, -10, 110}, PlotRange -> All],
    PlotRange -> {{-10, 110}, {-100, 400}}, Axes -> False, Frame -> True]}],
  {{j, 2}, 1, 10, 1}, ContentSize -> 450]
```



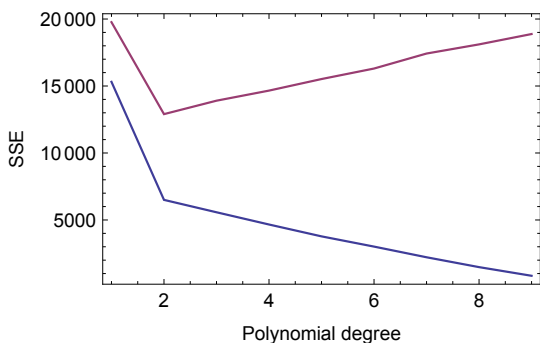
Wow... the higher-order polynomials fit *very* well indeed! Let's look at their ANOVA tables

```
lm = LinearModelFit[data, Table[x^i, {i, 1, 8}], {x}];
lm["ANOVATable"]
lm[x]
```

	DF	SS	MS	F-Statistic	P-Value
x	1	146446.	146446.	54879.5	0.0000182213
x <sup>2</sup>	1	15256.6	15256.6	5717.31	0.000174862
x <sup>3</sup>	1	308.139	308.139	115.473	0.00854916
x <sup>4</sup>	1	382.912	382.912	143.493	0.00689695
x <sup>5</sup>	1	929.088	929.088	348.169	0.00285985
x <sup>6</sup>	1	578.509	578.509	216.792	0.00458105
x <sup>7</sup>	1	1605.51	1605.51	601.653	0.00165796
x <sup>8</sup>	1	514.587	514.587	192.837	0.00514572
Error	2	5.337	2.6685		
Total	10	166027.			

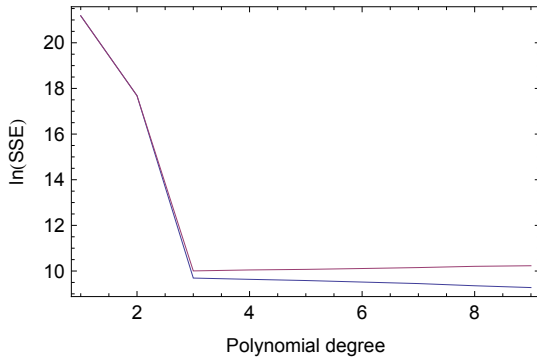
$$45.232 - 14.6438x + 0.264203x^2 + 0.0987109x^3 - 0.00673302x^4 + 0.000186535x^5 - 2.59283 \times 10^{-6}x^6 + 1.78932 \times 10^{-8}x^7 - 4.8809 \times 10^{-11}x^8$$

Let's do an experiment. I'm going to generate random datasets similar to the one above, and record the SSE — a measure of the difference between the model's predictions and the actual data. For each dataset I'm then going to generate another dataset *from the same model*, and look at the difference between the fitted line from the old dataset and new dataset. In other words, we are going to compare how well the fitted model predicted the data it was fitted to, vs. how well it predicts new data.

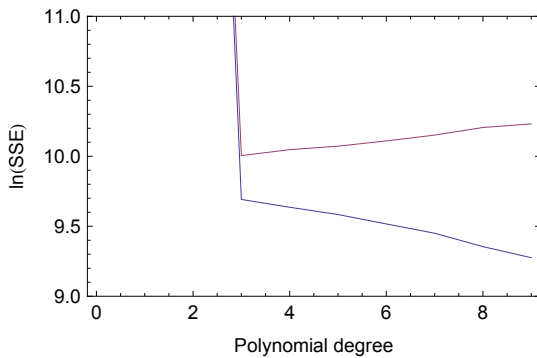


The blue line is the fit to the original data, and the red line is the fit to the new data. Can you tell what the order of the original model was? It was order 2 — a quadratic. The fit to the new data improves up to this point, and then gets worse again, even though the fit to the original data only gets better. The lesson is that fitting a high-order polynomial may give you a very good fit to your data, but the model does not generalize — it will be a bad predictor of other data. This means that it is not capturing the essence of the process.

Just to check, let's do it again for a third-order polynomial. The SSE values can be quite extreme, so we will log-transform them before plotting.



It's hard to see, but the red line begins to rise after the correct order. (Zooming in...)



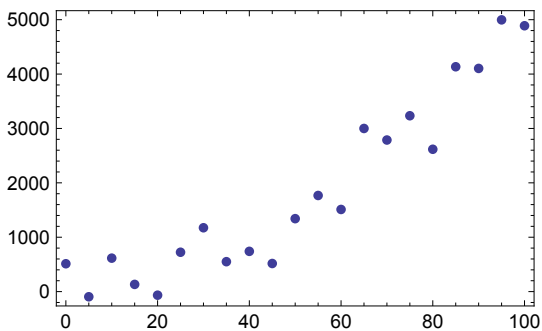
Choosing a too-complicated model is called **overfitting**. And what we did above gives a clue about one way to avoid it.

---

## Cross-validation

You will see from the above that the sum-of-square errors very clearly indicated the correct model, given a testing set of data from the same model. This provides a potential way to evaluate models that also addresses parsimony via predictive ability. If we only have a single set of real-world data, we can divide our single data set into two parts, one for 'training' (i.e., fitting) and one for testing. This process is called **cross-validation**. One method is to select out each of the data points in turn, fit to the rest, and measure the error from the selected point to the prediction of the fitted model. This is **leave-one-out cross-validation**. It's a very general model-selection method that can be used for any kind of statistical model for which predictive ability is the most important thing. Let's do it for our curve data:

```
data = Map[{#, 2 + 0.5 * # + 0.5 * #^2 + RandomReal[NormalDistribution[0, 500]]} &,
  Range[0, 100, 5]];
ListPlot[
  data]
```

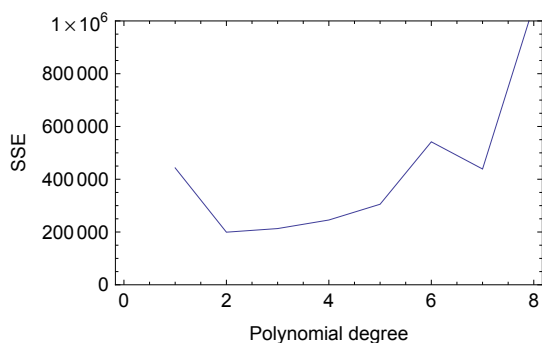


```

sse = Mean[Table[
  trainingData = Delete[data, {k}];
  testingData = data[[k]];
  lmList =
    Table[LinearModelFit[trainingData, Table[x^i, {i, 1, j}], {x}], {j, 1, 8}];
  squaredError = Table[(testingData[[2]] - lmList[[j]][testingData[[1]])^2,
    {j, 1, 8}]
  , {k, 1, Length[data]}]]
{442 995., 199 364., 213 071., 245 508., 305 266., 541 830., 438 797., 1.05814 × 106}

ListLinePlot[sse, FrameLabel → {"Polynomial degree", "SSE"},
  PlotRange → {Automatic, {0, 1 000 000}}]

```



It's not perfect, but it does eliminate the higher-order options.

FYI, for a linear model, there is (of course) a way to find these values without actually refitting the model a bunch of times.

---

## Multivariate model case study: predictors of exotic species richness

Go to MODIS notebook.

---

### Overfitting example 2: Too many predictors

If you recall from a previous class, adding polynomial terms is equivalent to making new predictor columns containing the  $x^2$ ,  $x^3$ , ... values. It follows that just as a regression dataset with  $n$  points can be perfectly fit by a  $(n - 1)$ th-degree polynomial, so it can be fit by a dataset with  $n - 1$  independent predictors, *no matter what actual numbers are in those predictors*.

Here is a dataset with 10 observations, 9 *random* predictor columns and 1 *random* response column (the last one).

```
TableForm[randomData = Table[RandomReal[], {10}, {10}]]
```

```
0.767906    0.894311    0.57115    0.637411    0.736555    0.271252    0.472919
0.0530867   0.672004   0.364231   0.807409   0.0651388   0.887885   0.0527685
0.586031    0.423184   0.286438   0.831436   0.196192    0.909584   0.800545
0.885502    0.268964   0.536945   0.666534   0.64892     0.0595592  0.467586
0.211452    0.547701   0.180663   0.233074   0.733557    0.395829   0.908072
0.94288     0.109638   0.0116652  0.692729   0.0108535   0.264967   0.065212
0.504902    0.896746   0.798904   0.892126   0.450425    0.808007   0.949318
0.594296    0.828688   0.0611895  0.66534    0.36879     0.95117    0.297325
0.120338    0.962739   0.1786     0.855425   0.444998    0.481924   0.316922
0.944711    0.791618   0.171134   0.5988     0.877618    0.444944   0.843788
```

Let's fit the model using all the predictor columns and look at the  $R^2$ .

```
lm = LinearModelFit[randomData,
  {x1, x2, x3, x4, x5, x6, x7, x8, x9}, {x1, x2, x3, x4, x5, x6, x7, x8, x9}];
lm["RSquared"]
lm["ANOVATable"]
```

```
Power::infy: Infinite expression  $\frac{1}{0}$  encountered. >>
```

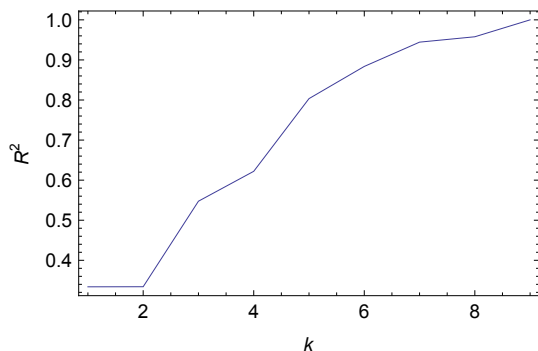
1.

	DF	SS	MS	F-Statistic	P-Value
x1	1	0.307099	0.307099	0.	1.
x2	1	0.000199129	0.000199129	0.	1.
x3	1	0.196319	0.196319	0.	1.
x4	1	0.0685882	0.0685882	0.	1.
x5	1	0.166858	0.166858	0.	1.
x6	1	0.0738788	0.0738788	0.	1.
x7	1	0.0557598	0.0557598	0.	1.
x8	1	0.0123268	0.0123268	0.	1.
x9	1	0.0388459	0.0388459	0.	1.
Error	0	$7.87636 \times 10^{-29}$	ComplexInfinity		
Total	9	0.919875			

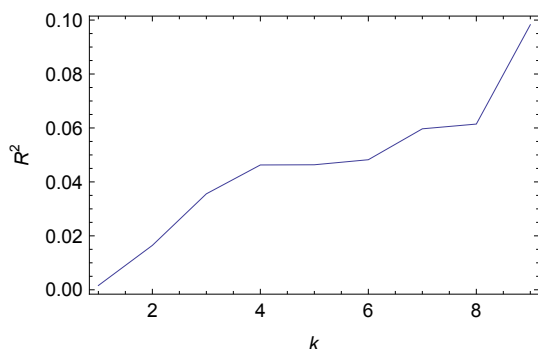
It's one — a perfect fit! But the data are entirely random — there's no relationship at all. We have over-fitted again. The number of predictors should not be anywhere near the number of observations. In the case above, the model begins to seriously over-fit at  $k \sim 4$  or 5.

```
ListLinePlot[
  Table[lm = LinearModelFit[randomData[All, Flatten[{Range[k], 10}]]], Take[{x1, x2,
    x3, x4, x5, x6, x7, x8, x9}, k], Take[{x1, x2, x3, x4, x5, x6, x7, x8, x9}, k]];
  {k, lm["RSquared"]}, {k, 1, 9}], FrameLabel -> {"k", "R2"}]
```

Power::infy: Infinite expression  $\frac{1}{0}$  encountered. >>



Let's do it again, with with 100 observations (and still just 10 predictors)



Look at the  $R^2$  values: the problem is now much less. Even with 10 predictors we are only predicting about 10% of the variation, instead of 100%.

### ■ Take-home message

If you want to detect the influence of predictors, then collect a *lot* of data. We will discuss power more formally in another class, but the bottom line is that the sample size you need scales with the number of parameters in your model, which scales with the number of potential predictors. What that means is that, whether you are designing a controlled experiment in the lab, or collecting data in the field, it is important to design your protocol so that you are testing the impact of just one, or a small number of things, otherwise the sample size you need will be unworkably large.

### ■ Looking ahead

One solution to too many predictors is **data reduction**, a topic for a future class.

---

## Overfitting example 3: Too many tests

Thinking about the 'too many predictors' problem above, you might say to yourself "Aha — I'll avoid the overfitting problem by testing each of my possible predictors separately." Alas, it's not that easy. You will recall that in traditional hypothesis testing,

you generate a  $p$  value, which is the frequency of getting a value of interest calculated from your data (such as the  $F$  statistic) *by chance* under the null hypothesis. If  $p$  is small, you conclude that your data did not come from the null hypothesis — you reject it. And the value below which  $p$  is considered to be small is  $\alpha$ , usually set at 0.05 for science.

However, if you think about this, it means that there is a 0.05 probability that you *would* get such a small  $p$  value by chance, which is a 0.05 probability that you would make a mistake, and reject the null hypothesis when it really does apply. Thus with  $\alpha$  you are setting an **error rate**. It is known as the **Type I error** rate — the rate of incorrectly rejecting the null hypothesis (seeing a relationship where none exists).

As you might expect, there is also a **Type II error** — incorrectly accepting the null hypothesis when it is really wrong (missing a relationship when one exists). This error rate, usually represented as  $\beta$ , is not one that you can simply set. What do you think determines it?

Returning to Type I error rates: if you have a 0.05 chance of a ‘false positive’ result in any given test, what happens if you do multiple tests? The probability of getting at least one Type I error in  $n$  tests is  $1 - (1 - \alpha)^n = 1 - 0.95^n$ . Here are the values for one through ten tests:

1	0.05
2	0.0975
3	0.142625
4	0.185494
5	0.226219
6	0.264908
7	0.301663
8	0.33658
9	0.369751
10	0.401263

If you do ten tests, there is a 40% chance that at least one of them will ‘test’ significant even if there is no real underlying effect. This is a real problem. There is a way for correcting for this problem, known as the Bonferroni correction. There are actually two types of Bonferroni correction.

### ■ Bonferroni correction for overall hypothesis

The first type of correction applies if you were using your separate tests as independent evaluations of a single hypothesis. For example, your ten predictor columns above might all be of the same variable  $x$ , but measured at different locations. Your ten separate tests would all be of the null hypothesis that ‘ $y$  isn’t a function of  $x$ .’ In that case, you simply adjust the  $\alpha$  value you use in each test by dividing it by the number of tests. In this case, that would be 10, so  $\alpha = 0.005$ . Here is the table of cumulative error probability with the adjusted  $\alpha$ :

1	0.005
2	0.009975
3	0.0149251
4	0.0198505
5	0.0247512
6	0.0296275
7	0.0344794
8	0.039307
9	0.0441104
10	0.0488899

You will see that it reaches approximately 0.05 after all ten tests, giving you the overall error rate you wanted. A more exact method is to use an  $\alpha$  threshold of exactly  $1 - (1 - \alpha)^{1/n}$ , which for  $n = 10$  is 0.0051162.

## ■ Holm-Bonferroni ‘step down’ correction for separate hypotheses

A different kind of correction applies if you are separately testing different hypotheses. Suppose your ten predictors are of different measurements all applying to the same dependent variable, and you want to know which ones have some predictive power. There is still room for error, because some of the relationships are likely to erroneous. In that case, the procedure is a bit more complicated. You perform all your tests, and rank them from smallest  $p$ -value (most significant) to largest. You then go down the list, testing each one against an alpha value that is the standard value (0.05) divided by  $n - (r - 1)$  where  $r$  is the rank of the test. So for ten tests, the most significant test is tested against  $\alpha = 0.05/10 = 0.005$ , the second against  $\alpha = 0.05/9 \approx 0.00556$ , the third against  $\alpha = 0.05/8 = 0.00625$ , and so on. As soon as one test fails to reject the null hypothesis, you stop and accept the null hypothesis for all the other tests.

## ■ Better solution: do the proper test!

Sometimes Bonferroni tests are unavoidable, but in the vast majority of cases in which they are used, it is to compensate for poor experimental or sampling design. Most often, the ‘overall hypothesis’ correction is used when in fact, a single, multiple-variable test should have been used instead, but the researcher found it to be ‘too complicated’ and only knew how to test one variable at a time.

## AIC — comparison with traditional hypothesis testing, univariate regression

### ■ Code

### ■ Tests

### ■ *Power* — ability to detect a real relationship (1 – Type II error — failing to detect a real relationship)

These regression data *do* have a relationship. There are 1000 random datasets.

```
datasets = Table[
  x = RandomReal[UniformDistribution[{0, 1}], 300];
  y = Map[1 + 1 * # + RandomReal[NormalDistribution[0, 2]] &, x];
  Transpose[{x, y}], {1000}];
```

These are the  $p$ -values for each dataset from traditional hypothesis testing. The first five only are shown.

$p$
0.000951176
0.00181427
0.390806
0.295507
0.00304982

These are the AIC values for the null and regression models for each dataset. The first five pairs only are shown.

	AIC (Null)	AIC (True)
	1331.5	1322.49
	1238.48	1230.67
	1300.63	1301.89
	1280.49	1281.39
	1254.46	1247.61

The fraction of traditional tests that are (correctly) significant when  $\alpha = 0.05$ .

```
power = N[Length[Select[pValues, # <= 0.05 &]] / Length[pValues]]
0.705
```

The fraction of AIC comparisons in which the more complex model is the best (lower AIC).

```
powerAIC = N[Length[Select[aicValues, #[[1]] > #[[2]] &]] / Length[aicValues]]
0.85
```

The Type II error is just  $1 - \text{power}$ .

```
type2Error = 1 - power
0.295

type2ErrorAIC = 1 - powerAIC
0.15
```

- **Type 1 error** — concluding that there is a relationship (i.e., failing to reject a null model) when there really is no relationship

These regression data do *not* have a relationship.

```
datasets = Table[
  x = RandomReal[UniformDistribution[{0, 1}], 30];
  y = Map[1 + 0 * # + RandomReal[NormalDistribution[0, 1]] &, x];
  Transpose[{x, y}], {1000}];
```

These are the  $p$ -values for each dataset from traditional hypothesis testing. The first five only are shown.

p
0.167381
0.989388
0.520367
0.23167
0.0498461

These are the AIC values for the null and regression models for each dataset. The first five pairs only are shown.

	AIC (Null)	AIC (True)
	79.0655	78.9865
	81.8356	83.8354
	98.3519	99.9012
	83.8067	84.2465
	89.2603	87.0655

The fraction of traditional tests that are (incorrectly) significant when  $\alpha = 0.05$ . This number will always be close to 0.05, especially when the number of tests is large, because the  $\alpha$  value specifies the probability of *each* test making that mistake.

```
type1Error = N[Length[Select[pValues, # <= 0.05 &]] / Length[pValues]]
0.057
```

The fraction of AIC comparisons in which the more complex model is the best (lower AIC).

```
type1ErrorAIC = N[Length[Select[aicValues, #[[1]] > #[[2]] &]] / Length[aicValues]]
0.183
```

## ■ Conclusion

AIC is slightly less conservative — more power, but more Type I error. But note the discussion in Burnham and Anderson — information-based methods are picking the ‘best approximating’ model, so tests like this that assess its ability to pick a ‘true’ model are not strictly valid.

---

## AIC — comparison with traditional hypothesis testing, multiple regression (3x, 1y)

### ■ Code

### ■ Tests

This regression does have a relationship. But note that the three slope parameters are different:  $\beta_0 = 1$ ,  $\beta_1 = 3$ ,  $\beta_2 = -1$ ,  $\beta_3 = 0.5$ .

```
x1 = RandomReal[UniformDistribution[{0, 1}], 100];
x2 = RandomReal[UniformDistribution[{0, 1}], 100];
x3 = RandomReal[UniformDistribution[{0, 1}], 100];
y = MapThread[
  1 + 3 * #1 - 1 * #2 + 0.5 * #3 + RandomReal[NormalDistribution[0, 2]] &, {x1, x2, x3}];
data = Transpose[{x1, x2, x3, y}];
```

Here is the output from traditional hypothesis testing:

```
fit = LinearModelFit[data, {1,  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$ }, { $\beta_0$ ,  $\beta_1$ ,  $\beta_2$ };
fit["ANOVATable"]
fit["BestFitParameters"]
```

	DF	SS	MS	F-Statistic	P-Value
$\beta_0$	1	57.3635	57.3635	14.3249	0.000267783
$\beta_1$	1	15.6892	15.6892	3.91794	0.0506375
$\beta_2$	1	0.0801625	0.0801625	0.0200183	0.887782
Error	96	384.428	4.00446		
Total	99	457.561			

```
{1.5548, 2.67, -1.49843, 0.109531}
```

For this particular dataset,  $\beta_1$  was not judged to be significant at all, and  $\beta_2$  is borderline.

Now we turn to the information-based approach, using AIC as our metric. These are the possible models (only the regression parameters are shown — the constant term is assumed):

```
xSubsetNames = Subsets [{"x1", "x2", "x3"}]
{{}, {x1}, {x2}, {x3}, {x1, x2}, {x1, x3}, {x2, x3}, {x1, x2, x3}}
```

The lines below fit the models and calculate various statistics from the output.

```
xSubsets =
  Map[Transpose[Prepend[], Table[1, {Length[y]}]]] &, Subsets[{x1, x2, x3}]];
lValues = Map[linregbylikelihood[Transpose[{y}], #, 0][[4]] &, xSubsets];
kValues = Map[linregbylikelihood[Transpose[{y}], #, 0][[2]] &, xSubsets];
aicValues = Map[linregbylikelihood[Transpose[{y}], #, 0][[5]] &, xSubsets];
```

We sort the AIC values.

```
sortedAICValues = Sort[aicValues]
{426.467, 428.446, 428.466, 430.403, 437.253, 438.795, 439.862, 441.216}
```

Calculate the difference between each value and the 'best' (smallest).

```
deltaAICValues = sortedAICValues - sortedAICValues[[1]]
{0., 1.97915, 1.99929, 3.93586, 10.786, 12.3278, 13.3945, 14.7485}
```

Convert to 'relative plausibility' values.

```
plausibilityValues = Exp[-0.5 * deltaAICValues]
{1., 0.371735, 0.36801, 0.139746, 0.00454823, 0.00210406, 0.0012343, 0.000627203}
```

And divide by the total to produce weights.

```
modelWeights = plausibilityValues / Total[plausibilityValues]
{0.52966, 0.196893, 0.19492, 0.0740178,
 0.00240901, 0.00111444, 0.000653758, 0.000332204}
```

Now we can present all of this in a table.

Model	-2 log-L	k	AIC	$\Delta$ AIC	Plausibility	Weights
x1x2	418.467	4	426.467	0.	1.	0.52966
x1x2x3	418.446	5	428.446	1.97915	0.371735	0.196893
x1	422.466	3	428.466	1.99929	0.36801	0.19492
x1x3	422.403	4	430.403	3.93586	0.139746	0.0740178
x2	431.253	3	437.253	10.786	0.00454823	0.00240901
x2x3	430.795	4	438.795	12.3278	0.00210406	0.00111444
	435.862	2	439.862	13.3945	0.0012343	0.000653758
x3	435.216	3	441.216	14.7485	0.000627203	0.000332204

These are the fitted parameter values associated with each model (again, ignoring the constant).

$$\begin{pmatrix} \{2.68665, -1.50444\} \\ \{2.67, -1.49843, 0.109531\} \\ \{2.80126\} \\ \{2.77086, 0.194579\} \\ \{-1.71897\} \\ \{-1.68277, 0.540057\} \\ \{\} \\ \{0.654365\} \end{pmatrix}$$

For each predictor variable, we can extract the weights from each of the models in which it appears, and sum them as a measure of **importance**.

x1

Parameter	w
2.68665	0.52966
2.67	0.196893
2.80126	0.19492
2.77086	0.0740178

x2

Parameter	w
-1.50444	0.52966
-1.49843	0.196893
-1.71897	0.00240901
-1.68277	0.00111444

x3

Parameter	w
0.109531	0.196893
0.194579	0.0740178
0.540057	0.00111444
0.654365	0.000332204

Overall

Parameter	Importance
x1	0.995491
x2	0.730076
x3	0.272357

We can also calculate a weighted average of the parameter estimates associated with each variable.

Parameter	Weighted average
x1	2.71206
x2	-1.5038
x3	0.135071

Finally putting it all together:

Parameter	Importance	Weighted Average
x1	0.995491	2.71206
x2	0.730076	-1.5038
x3	0.272357	0.135071

How does this compare to the hypothesis-testing approach above?