**Regression**

Now we’ll extend the results of the previous file to cover multiple independent variables. Note that we can visually ascertain whether a linear regression is appropriate by separately graphing the dependent variable against each of the independent variables. If *all* the separate plots are linear, then linear regression should work.

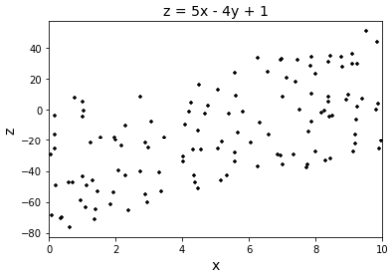
**Multiple Regression**

Now let’s examine a multiple independent variable case. For illustration’s sake let’s consider a case where multivariable regression is necessary, and where it is not. Below is depicted a case where z definitely depends on x and y.

A graph of a graph with numbers and a red line

Description automatically generated

And this would be kind of evident if we tried to plot z vs. just x or just y. Then we’d get:

 A graph of a number of dots

Description automatically generated

We can see the rough linear relationship between z and x, and z and y. But there is a lot of variation for an individual x or y. And in fact, z vs. x is z = 5x + [-4y] + 1. And when plot z vs. x, you’ll get the entire range of values in [-4y] for each value of x. This range is [-4y] = -4·20 = -80. Likewise when plot z vs. y, we get z = [5x] - 4y + 1, where [5x] encompasses the full range of values in [5x[ for each value of y. This range is [5x] = 5·10 = 50. Can clearly make out these ranges in the two plots above. And can see [-4y] > [5x], which is why the z vs. x plot is more scattered than the z vs. y plot. In either case, this is a lot of variation, which can obscure the linearity of the plot. And one might also say that the large variation about the regression line (not shown) in the z vs. x or z vs. y plot indicates the dependence of z on the other (y, or x, respectively) variable. To that point, consider z = 5x – 0y + 1,

A graph of a red line

Description automatically generated

Then data clearly only depends on x apropos the z variation. And the two graphs in each direction are:

A graph of a function

Description automatically generated A graph with black dots

Description automatically generated

Linear relationship w/ x is clearly observed. But relationship with y is non-existent. So from the small variation in the z vs. x plot, we might posit that x captures all the variation. And from the large variation in the z vs. y plot, we might say there is obvious evidence of dependence on another (x) variable.

Okay, well say we have a bunch of independent data points (x1i, x2i), etc., we want to fit data to the equation, for illustration’s sake,



where εi are independent normally distributed random variables with mean 0 and variance σ2. Note that we could also have f(x1, ln(x1)) = m1x1 + m2ln(x1) + b. This would be amenable to the same analysis as follows I think. Well we want to find the regression equation for the coefficients m1, m2, b. So say we have 5 data points. Then we can put our equation,



in the following form.



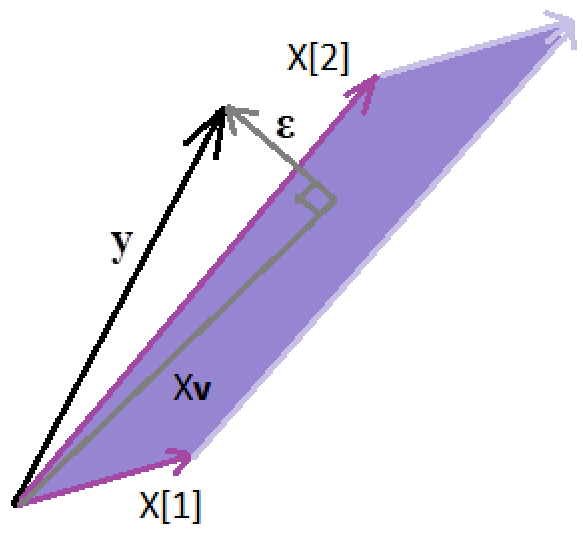
where,



And we want to find the **v** which again minimizes the expectation of the square error.



Instead of calculating this quantity and doing the whole calculus minimization thing, we can use a little bit of linear algebra. So, going back to the equation **Y** = X**v** + **ε**, we can graphically represent it as below. **y** is some vector. X**v** = X[1]v1 + X[2]v2 + X[3]v3 is some vector in the vector space spanned by the columns of X, i.e., X[1], X[2], X[3] (can’t picture X[3] in diagram). But **y** does not lie in this vector space. And **ε** is the vector which connects **y** and X**v**. Note the magnitude of **ε** is in fact SSEf. The **v** we want is the one which minimizes the distance between X**v** and **y**, i.e., which makes **ε** the smallest.



From the diagram we can see that **ε** will be smallest when it is perpendicular to X**v**. Well in fact, we can make a stronger statement. It is perpendicular to not just X**v**, but X**w** as well, for any **w**. So we can say,



Since this is true for any **w**, it must be the case that,



We can then solve for **v** and we get:



And the error would be:



Now fill



into the last two terms,



Maybe I’ll fill the v in again,



So we have:



I don’t feel like this would be the way one would actually calculate SSEf though, as inverting matrices takes a lot of computations.

**Point Estimator for v**

So like before, we’ll recognize that v is really a random variable V. So we’ll make it capital.



where **Y** is the random variable at the top of the page. Note that if we assume **Y** is normally distributed (i.e., that **ε** is normally distributed), then **V** is as well. Can show that <V> = v, since:



So indeed,



Next on our list was to get the variance. Then we’ll have the entire probability distribution of **V**.



So we have:



We could use this formula to work out the variances of the univariable regression model in previous file too. I’m not sure what this works out to in the multi-varable case. We’d have to take the inverse of a dim(**v**)×dim(**v**) matrix. Well, let’s just examine our illustrative example case,



Yeah, I don’t want to do the inverse. Okay maybe just the diagonal elements though. So first, the determinant,



in which case, doing the transpose of the matrix of cofactors and stuff,



To make practical use of these formulas it is nice to have a way to estimate σ2. Turns out an unbiased estimator of σ2 comes from SSEf.



where f is the number of d.o.f. in the model, i.e., the number of fit parameters. This is a natural generalization of the univariable linear regression model, where we’ll recall f was equal to 2. Remember SY2 is a random variable. And since SY2 is unbiased, we have:



So if we know know σ2 for sure, then we can say the ZVi guy,



follows a unit normal distribution. And if we don’t know it for sure (not sure how we would), then we can say,



follows a Studen’t T distribution with ν = n-f d.o.f. If ν > 30 or so, then this is just a normal distribution for all intents.

**Hypothesis Testing of Parameters**

If we are running an experiment, and get some particular values, <**V**i> = vi, for our linear regression, we may wish to see whether our results invalidate someone else’s linear regression values, or the standard/accepted linear regression values, vi\* . We can do a hypothesis test for this. We know how Vi are distributed, according to our experiments, and so I guess we could form the Z-statistics,



and then calculate the p-values, the probabilities that values at least as extreme as vi\* occur. So we’d calculate,



p(z) would be a Student’s T distribution with ν = n-f d.o.f.

**Confidence Intervals for Parameters**

We can also calculate confidence intervals for the statistics Vi. Just like we’ve done with other statistics, we’d say, at the 1-α confidence level,



where zα would be for the aforementioned Student’s T distribution.

**Goodness of Fit: R2 value**

Just like in the univariable case, R2 is given by:



where fi are the regression curve data points, which one could say is a random variable since it depends in known way (if have formula) on the random variable Yi, the data points we’re trying to fit. SSEf = sum of squares about f, and SSEm = sum of squares about mean. This is actually a general formula that applies to all curve fits. Apropos the fraction, the numerator is the ‘variation around the line, or f’, and the denominator is the ‘variation around the mean’). Can see R2 = 1 if there is no variation around the line, i.e., if the curve fits the data exactly. On the other hand, if the curve fits the data no better than the mean, then we get R2 = 0. If you have an R2 = 0.75, then you can say that 75% of the variation of the data is explained/predicted by the regression curve f. To make sense of that, consider following typical situation:

Chart, scatter chart

Description automatically generated

The green line represents the residual/variation in y about its mean . Could say that the purple line is the amount of this variation that is ‘explained’ by the regression curve *f*. Then the pink line would be the variation in y that itsn’t explained by *f*. This is just for heuristic purposes. So we’ll say:



Can therefore write R2 as:



R2 are not the whole story though. If our function f, contains many fitting parameters, then we can make the fit better and better and eventually increase the R2 to 1. But our fitting parameters wouldn’t necessarily be statistically significant. The next section tells us how to test the significance of our fitting parameters, or how to tell if one model is better than another one, sort of. Terminology note: Adjusted R2 = R2 – penalty for number of fitting parameters you have. This might provide a way to compare models with different numbers of fitting parameters. We’ll discuss this more below.

**Hypothesis Testing Different Regression Models**

Makes an interesting point about why arbitrary curve fitting is not necessarily valuable. Given a data set, you can precisely fit all the data if have polynomial curve of high enough degree. But question is, if you generate experimental data again, from the same experiment, will that same curve explain *that* data well? And typically, the answer is no. What will typically explain the data best, is a best fit based on a model that does actually reflect the underlying behavior of the data. Problem is that the higher polynomials will end up fitting the ‘noise’ rather than the ‘phenomenon’. So to test whether a model is fitting the phenomenon well, you should ‘cross validate’, which is split your data up into groups, curve fit to one of the data sets, and then compute the R2 (see below) for the other data sets with that same curve. You’ll find that the model which fits the phenomenon has the best average R2 values.

Or another way…say we have a model with **v**(2) = (v0, v1) and another model with **v**(5) = (v0, v1, v2, v3, v4). For example the first could be fitting a line through some data, and the second fitting a cuartic polynomial. Now let’s define a Null Hypothesis.

H0 = assumption that the data is described by model **Y** = X**v**(f0) + ε, which has f0 degrees of freedom.

And let SSEf0 be the sum of the square errors for this model.Then let’s compare to another model with f > f0 degrees of freedom, fitting **Y** = X**v**(f) + ε. And let SSEf be its sum of squared errors. We would anticipate this to be smaller of course, i.e., SSEf0 > SSEf. The alternative hypothesis would be:

HA = assumption that at least one of the extra f – f0 parameters in the new model Y = Xv(f) + ε is non-zero.

Given the Null Hypothesis, turns out SSEf follows a known probability distribution. Well we can form a test statistic,



and n = number of data points. We might recall SSEf/(n-f) is just the point estimator for σ2 in the f-model. Might interpret Z as:



Turns out this follows an F-distribution (B is the β function).



which is the probability density of getting an Z-value of x, given the null hypothesis is true. So we can calculate a p-value,



which would be the probability that we’d get an Z-statistic Z\* or higher, out of the new model, if the Null hypothesis were true. So if the f model has true explanatory power, then we should find Z\* >> 0 and the p-value should be small (less than 0.05 at the 95% significance level). We can test different models this way. We can let the f0 model be one modeling just the mean, or a linear regression, or a quadratic regression, etc. But note for instance that if f = n, i.e., if the number of fit parameters equals the number of data points (allowing an exact fit of f to our data points), then Z\* = 0, and so our p-value would be 1 I think. And this would mean that our extra parameters are meaningless.

**Polynomial Regression**

There isn’t a dedicated polynomial regressor in sklearn. But that’s okay. Polynomial regression is just linear regression. And all the stuff we did above for linear regression applies to polynomial regression too I think. Say we have dataframe, X,

|  |  |
| --- | --- |
| x1 | x2 |
| 1.2 | -3.1 |
| 3.9 | -2.2 |
| 4.2 | -1.4 |

If wanted to do, say, quadratic regression on X, instead of linear regression, then we could just add a few columns,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1 | x2 | x12 | x22 | x1x2 |
| 1.2 | -3.1 | 2.4 | 9.6 | -3.7 |
| 3.9 | -2.2 | 15.2 | 4.8 | -8.6 |
| 4.2 | -1.4 | 17.6 | 2.0 | -5.9 |

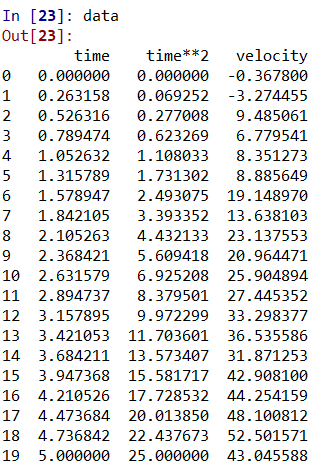
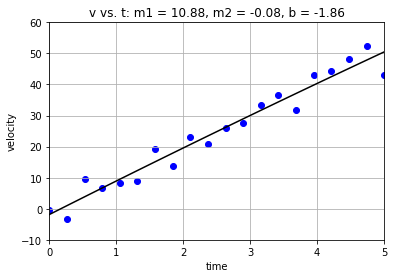
and do linear regression on this. One can write a Polynomial\_Regression\_Class in Python that’s a subclass of sklearn’s LinearRegression class. And can program it to take specified numerical columns, say, ‘a’, and ‘b’, and ‘c’, and add quadratic, cubic, etc., columns for regression. So if want to do quadratic regression on these, then we’d need to add columns: a2, ab, ac, b2, bc, c2. And if want to do cubic regression, we’d need to add a3, a2b, a2c, ab2, ac2, b3, b2c, bc2, c3. etc. Have noticed that with large data sets, I sometimes run into problem when trying to do polynomial regression. I get warning that matrix is ‘ill-conditioned’.

**Using Coefficients to Assess Feature Importance**

We can consider the coefficients, vi, as proportional to the relative importance of the variable xi. I think this is especially so when using Lasso Regression?

**Example**

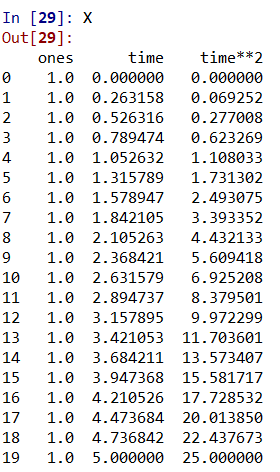
So I decided to fit a quadratic to the v vs. t data, specifically something like v = m1t + m2t2 + b.

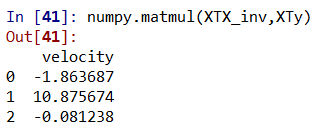
As can see the quadratic coefficient is quite small, as we’d expect, since it shouldn’t really be there at all. Let’s calcultate some of that stuff above ourselves. So first, the fit parameters would be:



where X = [(ones) (time) (time\*\*2)]



We can perform the requisite calculations in numpy. And we find for the regression coefficients, b, m1, and m2:



So,



Next we’d like to have a confidence interval for these guys. To work that out, first we need an estimate for σ2, the variance of the data points about the fit. This is σ2 ≈ SY2. We can get SSEf from subtracting the fit velocity values from the actual velocity values, squaring each, and summing. We find:



(which is pretty close to what I actually used for σ2 = 16) Okay, now using this, we can get the variances in the parameters via:



We find,



So 95% confidence intervals for our parameters are (using Student’s T distribution zα/2):



Let’s get the R2 coefficient. This is:



And finally, let’s do the F-statistic thing. We’ll take the Null Hypothesis to say that we can just characterize the data by the mean, or equivalently, the intercept b, so f0 = 1. And then we’ll take the Alternative Hypothesis to be that at least one of our f = 3 model’s extra parameters, i.e. m1, or m2, is non-zero.



Now we can use this to calculate the probability that our Null hypothesis is true.



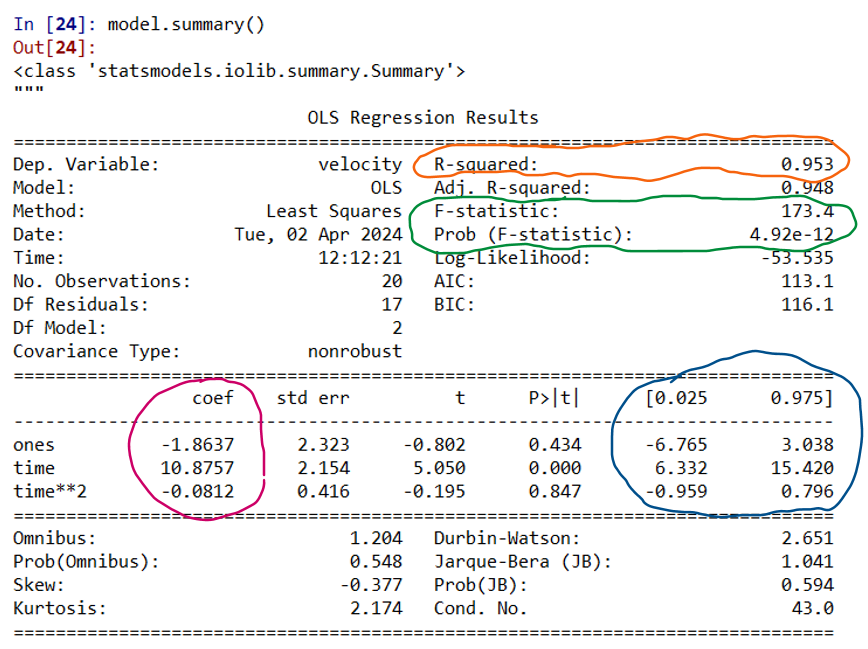
So we can see the likelihood that our velocities are just randomly distributed about the mean is astronomically low. But now let’s compare to our prior model (previous file), which tried to explain the velocity data via just time and not time2. So we’ll take our Null Hypothesis to say that we can characterize the data by the intercept and slope, b and m1, so f0 = 2. And then we’ll take the Alternative Hypothesis to be that at least one of our f = 3 model’s extra parameters, i.e., m2, is non-zero. Recalling the SSE = 248 of the linear model, we have:



Now we can use this to calculate the probability that our Null hypothesis is true.



This indicates that it is 79.4% likely that we could get as good a fit with our f0 model as with our f model. And so we have basically no confidence that our extra time2 variable is doing anything useful. And of course, this congrues with our expectations. We can get a lot of these parameters from the statsmodels library in Python. Here’s its readout below. We can see the R2, the regression coefficients, the 95% confidence intervals, and the F-statistic thing. Looks it just does the F-statistic where we’re comparing to the mean model, where f0 = 1. I guess that makes sense.



So there.

**Appendix: Outliers**

I’d like to consider how we might detect outliers. Consider the graph,

A graph of a graph with numbers and a red line

Description automatically generated

An outlier would be considered something like the blue points, below:

A graph with a red line

Description automatically generated

Note we wouldn’t be able to detect it based on its x, y or z value per se´. As the z-value is within the range of other z values – approximately in the middle of the range in fact. And the (x,y)-values is also within the range. What makes it an outlier is that it is so far off the regression plane. Do we have to have a regression plane before we can classify something as an outlier? Maybe we could break the data up into bins, (ΔxΔy), and calculate the standard deviation, σ(ΔxΔy),, of points within the interval. Then if there is an interval where σ(ΔxΔy) is exceedingly large, perhaps there is an outlier there. This would work in principle. But breaking our feature space into such intervals would be impractical for high dimensional spaces. We could try to plot z vs. x, and then y. But the outliers will not necessarily show themselves. Say we plot z vs. y. Consider the blue point, at (x,y) = (2,20) or so. This point doesn’t have an unusually large z value. So when we plot all the points given by y = 20, which range between (-75,-25), it would inconspicuously show up in the middle of them. What if we plot z vs. x? When we plot all the points given by x = 2, which range between (-70,0), say, it would again show up inconspicuously. Another possibility is to use regression plane, and then eliminate the points that are outliers w/r to the plane. And then repeat the process. This is like our proposed method for univariable regression. So,

1. Draw a regression plane through all points, R0.
2. Identify outliers based off of R0. Take these out, and put them in a set S1. Using the quartile identification method, (Q1 – 1.5×IQR, Q3 + 1.5×IQR), seems best.
3. Redraw a regression plane, R1, through remaining points.
4. Put outliers, S1, back in, and identify new outliers based off of R1. We’ll call these S2. And take these points S2 out.
5. Redraw a regression plane, R2, through remaining points.
6. Put outliers, S2, back in, and identify new outliers based off of R2. We’ll call these S3. And take these points S3 out.
7. Redraw a plane line, R3, through remaining points.
8. Put outliers, S3, back in, and identify new outliers based off of R3. We’ll call these S4, etc.
9. And keep doing this until the outlier sets converge to a final result, i.e., Sn = Sn-1. And we’d say our final regression plane is Rn.

**Appendix: Multicolinearity Trap**

So the multicolinearity trap occurs when a column of data is correlated with another column of data. Don’t really want to try to draw that, but consider the graph below.

A graph of a red line

Description automatically generated

The x’s and y’s are uncorrelated here because they randomly vary all over their range, i.e., the entire x-y plane. On the other hand, they’d be corrrelated if the x’s and y’s were bundled around some curve in the x-y plane. They’d be linearly correlated if they lied all about some diagonal line in the x-y plane. This can be bad, I think, because of how regression coefficients are calculated. Recall our formula was:



and if the xa1 column is exactly linearly correlated with the xa2 column, then I think that makes X singular, and so that *might* mean that (XTX) doesn’t have an inverse. For instance, consider:



The first column is the sum of the second and third columns. And XTX doesn’t have an inverse. If the two columns are *approximately* linearly correlated, then that would just make calculating the inverse problematic/possibly unstable. So we want to avoid this. Good thing is, if xa1 is linearly correlated with xa2, then we only need one of the columns, not both. And so we can just drop one, which is what one does.