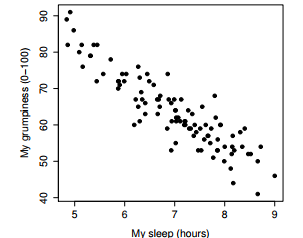
***Learning Statistics with R - University of Adelaide***

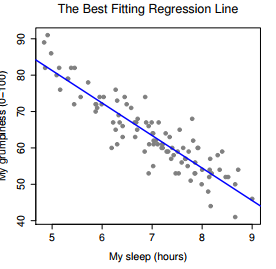
***Part V – Statistical Tools***

**15.** **Linear regression**

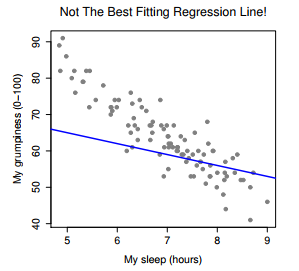
* **Linear regression** = standard tool statisticians rely on when analyzing the relationship between interval scale **predictors** + interval scale **outcomes**.
* Stripped to bare essentials, linear regression models = basically a slightly fancier version of the **Pearson correlation** but are much more powerful tools.
* Recall w/ out parenthood data set, we‘re trying to find out why Dan is so very grumpy all the time, + our working hypothesis = not getting enough sleep.
* Drew some scatterplots to help examine the relationship between amount of sleep vs. grumpiness the following day.



* Corresponds to a correlation = r = -.90, but what we find ourselves secretly imagining is something that looks closer to:



* i.e. Draw a straight line through the middle of the data.
* In statistics, this line = a **regression line** = goes through the middle of the data.



* This *doesn’t* fit data very well, so it doesn’t make a lot of sense to use it as a way of summarizing
* Formula for a straight line 🡺 **y = mx + b** w/ 2 variables = x + y and 2 coefficients, m + b.
* Coefficient **m** = slope of the line, + coefficient **b** = y-intercept of the line (value of y when x = 0)
* Similarly, a slope of **m** = increase x by 1 unit, y goes up by m units;
* A **–m** slope = y would go down m units
* If Y = outcome variable (DV) + X = predictor (IV), the formula that describes a regression:



* Now we have Xi + Yi rather than just X + Y 🡺 to note we’re dealing w/ actual data
* Xi = value of predictor variable for the i-th observation (# of hours of sleep on day i of study) + Yi = corresponding value of the outcome variable (grumpiness on that day
* We’re assuming this formula works for all observations in the data set (i.e., for all i).
* Yˆi 🡺 make the distinction between the *actual data* Yi + the *estimate* Yˆi (prediction regression line is making).
* Also changed coefficients from m + b to **b1 + b0** 🡪 how statisticians like to refer to regression coefficients.
* Regardless of a good or bad regression line, the data don’t fall perfectly on it/the data, **Yi,** are not identical to the predictions of the regression model, **Yˆi**
* Refer to the difference between model prediction + actual data point as a **residual**

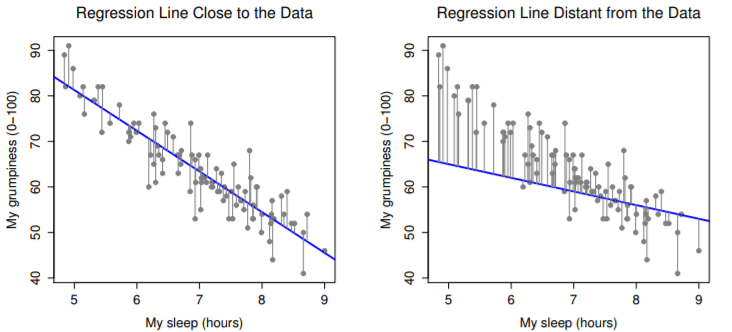


* In turn, we can write down the complete linear regression model as:



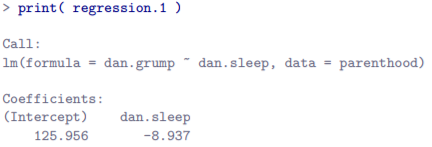
**15.2 Estimating a linear regression model**

* Now add some lines to show the size of the residual for all observations.



* When the regression line is good, **residuals** (lengths of the solid black lines) = small, but when the regression line is bad, residuals = a lot larger
* Best fitting regression line = one w/ smallest residuals.
* Since statisticians like to take squares of everything, say:
* “The estimated regression coefficients, ˆb0 + ˆb1, are those that minimize the sum of the squared residuals, which we either write as 
* ***NOTE:*** Our regression coefficients are *estimates* (trying to *guess* the parameters that describe a population) 🡪 have ˆb0 + ˆb1 rather than b0 + b1.
* ***ALSO NOTE*** 🡪 Since there’s actually more than 1 way to estimate a regression model, the more technical name for this estimation process is **ordinary least squares (OLS) regression**.
* At this point, we have a concrete definition for what our best choice of regression coefficients, ˆb0 + ˆb1 are
* If our optimal regression coefficients = those that minimize the **sum squared residuals**, how do we find these numbers?
* The actual answer to this question = complicated 🡺 
* where bˆ = a vector containing the estimated regression coefficients, X = the **design matrix** that contains the predictor variables (+ an additional column containing all ones
* A matrix made strictly of X = a matrix of the regressors)
* y = a vector containing the outcome variable.
* **lm()** =1st agr = a formula that specifies the regression model.
* For simple linear regression models w/ a single predictor variable + an intercept term, this is of the form **outcome ~ predictor**.
* However, more complicated formulas are allowed
* Output of lm() = a fairly complicated object w/ quite a lot of technical info buried under the hood, which is used by other functions 🡪 generally a good idea to create a variable to stores results of a regression.





* Can see R gives us the intercept ˆb0 = 125.96 + slope ˆb1 = -8.94 🡺 the best-fitting regression line plotted above has this formula:



* The most important thing to be able to understand is how to interpret these coefficients.
* For slope, a regression coefficient of ˆb1 = -8.94 means if we increase Xi by 1, we decrease Yi by 8.94
* That is, *each additional hour of sleep gained will improve mood, reducing grumpiness by 8.94 points*
* For the intercept, since ˆb0 = the expected value of Yi when Xi = 0 🡪 implies if we get 0 hours of sleep (Xi = 0), grumpiness will go off the scale to an insane value of Yi = 125.96

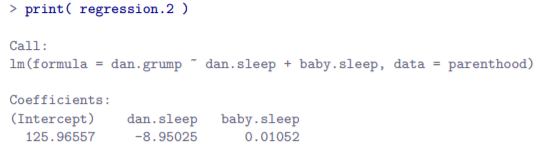
**15.3 Multiple linear regression**

* The simple linear regression model assumes there’s a single predictor variable you’re interested in
* Up to this point, every statistical tool used so far has assumed the analysis uses 1 predictor variable + 1 outcome variable.
* However, in many (perhaps most) research projects you actually have multiple predictors you want to examine.
* **Multiple regression** = add more terms to the regression equation 🡪 now we have 2 X variables, say 1st = Xi1 = amount of sleep I got + the 2nd = Xi2 = amount of sleep my son got, both on the i-th day

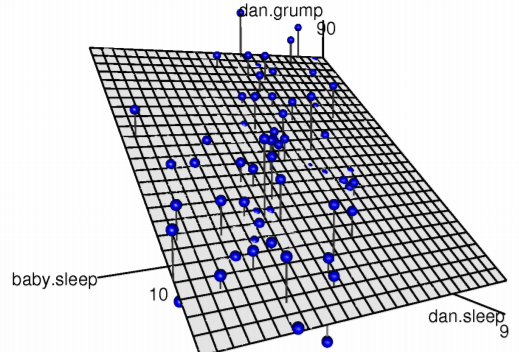


* Now have 3 coefficients to be estimated: b0 = intercept, b1 = coefficient associated w/ my sleep, + b2 = coefficient associated w/ my son’s sleep.
* # of coefficients to be estimated has changed, but basic idea of how the estimation works is unchanged: estimated coefficients ˆb0, ˆb1 + ˆb2 = those that minimize the **sum squared residuals**





* Coefficient for dan.sleep = large, suggesting every hour of sleep lost makes me a lot grumpier
* However, coefficient for baby.sleep = very small, suggesting it doesn’t really matter how much sleep my son gets
* What matters as far as my grumpiness goes is how much sleep *I* get.



* A 3D visualisation of the multiple regression model w/ 2 predictors, dan.sleep + baby.sleep; + 1 outcome variable, dan.grump.
* Together, these 3 variables form a 3D space: each observation (dots) = a point in this space.
* In much the same way a simple linear regression model forms a line in 2D space, this multiple regression model forms a plane in 3D space.
* **When estimating regression coefficients, we’re trying to do is find a plane as close to all the dots as possible.** (drawn using **scatter3d()** fromin the **car** package
* If you want > 2 predictors, just add more X terms + more b coefficients.
* In other words, if you have K predictor variables in a model, the regression equation looks like:



**15.4 Quantifying the fit of the regression model**

* We don’t yet know if this regression model is any good.
* regression.1 claims every hour of sleep improves mood by quite a lot, but might just be rubbish
* Remember, this model only produces a *prediction* Yˆi about what my mood is actually like = Yi.
* If these 2 are very close, the regression model has done a good job + if very different, a bad job.
* So, we’ve got the **sum of the squared residuals,** which we would hope to be pretty small.



* Specifically, we’d like for it to be very small *in comparison to the total variability in the outcome variable* = **total sum of squares**



* To get them, 1st put our values into vectors:



* Then store the equation for the regression from our model output



* Then calculate SSres and SStot

* A large SSres doesn’t mean very much, but SStot is a much bigger number, so this suggests our regression model was making good predictions.
* But *it’s not very interpretable*.
* To can fix this, convert these 2 fairly-meaningless #’s into a nice, interpretable number, **R2**
* Want R2 to be = 1 if the regression model makes *no errors* in predicting the data.
* If it turns out the residual errors (SSres) = 0, then we expect R2 = 1.
* Similarly, if the model is completely useless, we’d like R2 to be = 0.
* Useless model 🡺residual sum of squares is no smaller than the total sum of squares, **SSres = SStot.**



* **R2 = the coefficient of determination =** the proportion of the variance in the outcome that can be accounted for by the predictor.
* R2 = .816 means the predictor (my.sleep) explains 81.6% of the variance in the outcome (my.grump)
* At this point, revisit the earlier claim that regression, in this very simple form discussed so far, is basically the same thing as a correlation.
* Previously, **r** denoted a **Pearson correlation**.
* There is a relationship between the value of the correlation coefficient r + an R2 value from linear regression 🡪 the squared correlation, **r^2**, is identical to R2 for *a linear regression w/ only a single predictor*



* In other words, running a Pearson correlation is more or less equivalent to running a linear regression model that uses only 1 predictor variable
* It’s quite common to report a slightly *different* measure of model performance, **adjusted R2**.
* Motivation behind calculating adjusted R2 = adding more predictors into the model will ALWAYS cause R2 to increase (or at least NOT *decrease*).
* Adjusted R2 introduces a slight change to the calculation
* For a regression model w/ K predictors, fit to a data set w/ N observations, adjusted R2 is:



* This adjustment = an attempt to take dF into account.
* Big advantage of the adjusted R2 = when you add more predictors to the model, adjusted R2 value only increases if the new variables improve model performance more than you’d expect by chance.
* Big *disadvantage* = adjusted R2 value *can’t be interpreted in the elegant way that R2 can*.
* R2 has a simple interpretation = proportion of variance in outcome explained by regression model
* To my knowledge, no equivalent interpretation exists for adjusted R2
* To report R2 or adjusted R2, is probably a matter of personal preference.
* If you *care more about interpretability*, R2 is better.
* If you care *more about correcting for bias*, adjusted R2 is probably better.
* R2 🡺 more important to be able to *interpret* a measure of model performance.
* But if worried the improvement in R2 you get by adding a predictor is just due to chance + not b/c it’s a better model, we’ve got hypothesis tests for that.

**15.5 Hypothesis tests for regression models**

* There are 2 different (but related) kinds of hypothesis tests for regression models:
* Test whether a regression model, *as a whole*, is performing significantly better than a null model
* Test whether a *particular regression coefficient* is significantly different from 0
* Can reuse the F-test + the t-test + import them into the regression framework.
* Testing the model as a whole
* Suppose you’ve estimated a regression model + the 1st first hypothesis test you might want to try 🡺 one in which the null H0 = there is no relationship between the predictors + the outcome,
* The alternative H1 = the data are distributed in exactly the way the regression model predicts.
* Formally, the null model corresponds to the fairly trivial regression model w/ 0 predictors + only include the intercept term b0



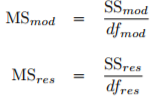
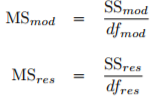
* If our regression model has K predictors, the alternative model is described using the usual formula for a multiple regression model:



* How can we test these 2 hypotheses against each other?
* Understand that just like w/ ANOVA, it’s possible to divide up total variance, **SStot**, into the **sum of the residual variances SSres** + the **regression model variance SSmod**.



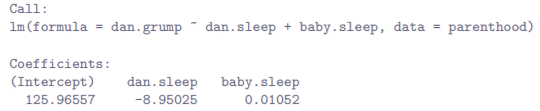
* Just like we w/ ANOVA, convert the sums of squares in to **mean squares** by dividing by dF

* So, how many degrees of freedom do we have?
* The *df associated w/ the model* is closely tied to the *# of predictors included*
* In fact, it turns out df\_mod = K + for the residuals, + we then have dF\_res = N - K - 1.
* Then calculate an F-statistic like this:



* This F statistic has exactly the same interpretation as the 1st one = Large F values indicate the null is performing poorly in comparison to the alternative.
* Tests for individual coefficients
* The above F-test = useful for checking a model, as a whole, is performing better than chance
* This is important: if a regression model doesn’t produce a significant result for the F-test, you probably don’t have a very good model (or, quite possibly, don’t have very good data)
* However, while failing this test = a pretty strong indicator a model has problems, passing the test (i.e., rejecting the null) *doesn’t imply that the model is good*
* The answer to why this is so can be found by looking at the coefficients for the regression.



* Notice the estimated regression coefficient for baby.sleep is tiny (0.01), relative to the value for dan.sleep (-8.95).
* Given these 2 variables are on the same scale (both measured in hours slept), this is suspicious.
* Could suspect it’s really only the amount of sleep *I* get that matters in order to predict mood.
* Can reuse a hypothesis test, the t-test w/ a null H0 = the true regression coefficient = 0 (b = 0), which is tested against the alternative H1 that it isn’t (b != 0).



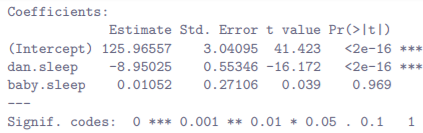
* If the CLT is kind to us, we might be able to guess that the sampling distribution of the *estimated* regression coefficient, **ˆb**, is a normal distribution w/ mean centered on b.
* This would mean that if the null were true, the sampling distribution of ˆb has mean = 0 + unknown SD.
* Assuming we can come up w/ a good estimate for the **standard error of the regression coefficient**, **se(ˆb)**, we’re in luck.
* That’s exactly the situation for the one-sample t-test
* Define a t-statistic:



* Skipping over reasons why, dF in this case = N - K - 1.
* The estimate of the SE of the regression coefficient, **se(ˆb)**, is not as easy to calculate as the SE of the mean used for the simpler t-tests (Chapter 13).
* **Vector of residuals** =  . For K predictors + the intercept, **estimated residual variance** =  . The **estimated covariance matrix** of the coefficients = , the main diagonal of which =  , our estimated standard errors
* For our purposes it’s sufficient to point out that the SE of the estimated regression coefficient depends on *both the predictor + outcome variables*, + is somewhat *sensitive to violations of the homoscedasticity assumption*
* In any case, this t-statistic can be interpreted in the same way as before
* Assuming you have a 2-sided alternative (don’t care if b > 0 or < 0), it’s the extreme values of t (a lot less or a lot greater than 0) that suggest you should reject the null.
* To compute all these quantities above, ask for a summary() of a regression model.
* 

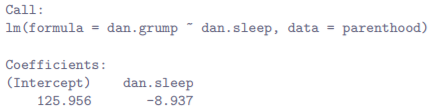








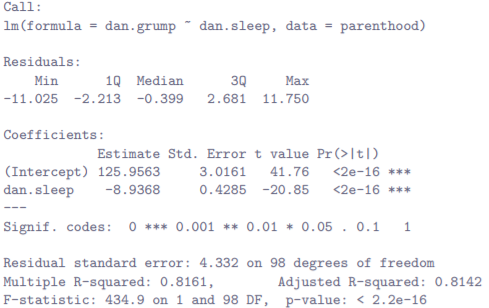
* Reminds what the actual regression model is + provides a quick summary of the residuals (**ε**(i) values), which can be convenient as a quick + dirty check that the model is okay.
* Remember, we *assumed these residuals were normally distributed, w/ mean 0*.
* In particular, it’s worth quickly checking to see if the median is close to 0, + to see if the 1st quartile is about the same size as the 3rd quartile.
* If they look badly off, there’s a good chance the assumptions of regression are violated.
* These ones look pretty nice to me, so let’s move on.
* Then we have coefficients of the model + test statistics,
* In this case, the model performs significantly better than you’d expect by chance (F(2, 97) = 215.2, p < .001), which isn’t all that surprising
* R2 = .812 indicates the model accounts for 81.2% of the variability in the outcome measure.
* *However*, looking at the t-tests for each individual coefficient (last col in coefficients table), = pretty strong evidence baby.sleep has no significant effect + that all work is being done by dan.sleep.
* Taken together, these results suggest the model is *actually the wrong model for the data:*
* Probably be better off dropping baby.sleep predictor entirely (no \*’s).
* In other words, the 1st model we started w/ is the better model



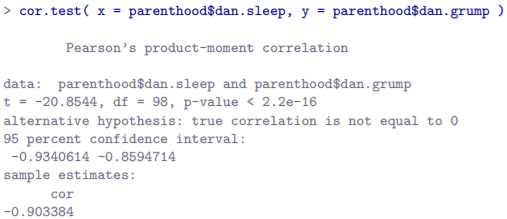
* Note that, although R has done multiple tests here, it hasn’t done a **Bonferroni correction** or anything.
* The above are *standard* one-sample t-tests w/ a 2-sided alternative.
* If you want to make corrections for multiple tests, you need to do that yourself

**15.6 Testing the significance of a correlation**

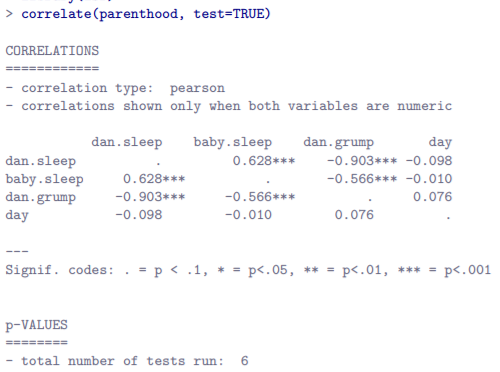
* Hypothesis tests for a single correlation
* Worth very briefly returning to the point made earlier 🡪 **Pearson correlations** = basically the same thing as **linear regressions** *w/ only a* ***single predictor*** *added to the model*.
* This means that the hypothesis tests described in a regression context can also be applied to correlation coefficients.

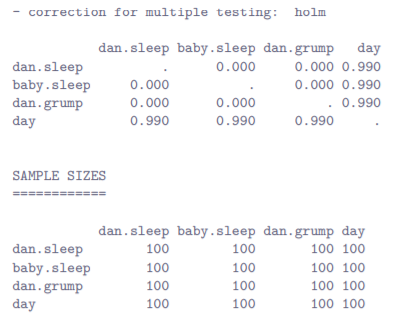


* Now compare to the output of **cor.test() 🡪** runs a hypothesis test to see if an observed correlation between 2 variables is significantly different from 0.



* Again, key thing to note = line that reports the hypothesis test itself, which seems to be saying that t(98) = -20.85, p < .001.
* It’s exactly the same test as in the linear regression summary
* **The test for the significance of a correlation is identical to the t-test run on a coefficient in a regression model.**
* Hypothesis tests for all pairwise correlations
* So cor.test() lets you run a hypothesis test on a *single* correlation + is an extension of cor()
* cor() isn’t restricted to computing a single correlation 🡪 can use it to compute all pairwise correlations among variables in a data set.
* cor.test() cannot do the same thing = Cannot use cor.test() to run hypothesis tests for all possible pairwise correlations among variables in a data frame
* If you’ve got some reason to be asking “is A related to B?”, you should absolutely run a test to see if there’s a significant correlation.
* But if you’ve got variables A, B, C, D + E + you’re thinking about testing correlations among all possible pairs of these, a statistician would want to ask: *what’s your hypothesis*?
* If you’re in the position of wanting to test all possible pairs of variables, you’re pretty clearly on an *expedition*, hunting around in search of significant effects when you don’t actually have a clear research hypothesis in mind.
* This is dangerous, + authors of cor.test() obviously felt they didn’t want to support that kind of behavior.
* On the other hand, we’ve encountered this situation before (Section 14.5) w/ **post hoc tests** in ANOVA.
* When running **post hoc tests**, we didn’t have any *specific* comparisons in mind, so we applied a correction (Bonferroni, Holm, etc.) to avoid the possibility of an inflated Type I (FP) error rate.
* From this perspective, it’s okay to run hypothesis tests on all pairwise correlations, ***but you must treat them as post hoc analyses + if so apply a correction for multiple comparisons***.
* That’s what **correlate()** in the **lsr** does.
* Get correlate() to output results of all pairwise tests by specifying **test=TRUE**.





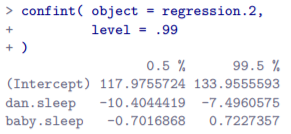
* The output contains 3 matrices.
* 1) **Correlation Matrix**
* 2) Matrix of p-values, using the Holm method to correct for multiple comparisons.
* change correction applied by specifying **p.adjust.method** argument
* 3) Matrix indicating sample size (# of pairwise complete cases) that contributed to each correlation
* If you really desperately want to do pairwise hypothesis tests on correlations, correlate() will let you do it.
* But be careful 🡪 could run these pairwise correlation tests + get 1 or 2 significant results that don’t make any sense.
* For some reason, the moment people see significance stars, they feel compelled to throw away common sense + assume the results must correspond to something real that *requires* explanation
* *In most such cases, my experience has been that the right answer is “it’s a Type I error”.*

**15.7 Regarding regression coefficients**

* Confidence intervals for the coefficients
* Like any population parameter, the regression coefficients **b** cannot be estimated w/ complete precision from a sample of data (part of why we need hypothesis tests)
* Given this, it’s quite useful to be able to report CI’s that capture our uncertainty about the true value of b.
* This is especially useful when a research question focuses heavily on an attempt to find out how *strongly* variable X is related to variable Y, since in those situations the interest is primarily in **the regression weight** **b**.
* Fortunately, CI’s for regression weights can be constructed in the usual fashion



* Where **se(ˆb)** = standard error of the regression coefficient + **t\_crit** = relevant critical value of the appropriate t distribution.
* For instance, for a 95% CI, critical value = the 97.5th quantile of a t-distribution w/ N – K – 1 dF
* In other words, this is basically the same approach to calculating CIs used before.
* To do in R use **confint()** w/
* an object 🡺 regression model (lm object) for which CIs are require)
* a **parm** 🡺 vector indicating which coefficients we should calculate intervals for
* either #’s or (more usefully) a character vector containing variable names
* By default, all coefficients are included, so usually don’t bother specifying this argument
* A **level** = A # indicating the confidence level that should be used.
* As is usually the case, default value = 0.95, so you wouldn’t usually need to specify this
* So, suppose I want 99% CIs for the coefficients in regression.2 model.



* Calculating standardized regression coefficients
* 1 more thing that you might want to do is to calculate **standardized regression coefficients**, often denoted **β**.
* The rationale behind standardized coefficients = in a lot of situations, variables are on fundamentally different scales.
* Ex: Regression model aims to predict IQ scores using educational attainment (# of years of education) + income as predictors.
* Obviously, educational attainment + income are not on the same scales
* # of years of schooling can only vary by 10s of years, whereas income would vary by 10,000s of dollars (or more).
* Units of measurement have a big influence on the regression coefficients: *the* ***b*** *coefficients only make sense when interpreted in light of the units,* both of predictor variables + the outcome
* This makes it very difficult to compare coefficients of different predictors.
* Yet there are situations where you want to make comparisons between different coefficients.
* Specifically, might want some kind of standard measure of *which* predictors have the *strongest* *relationship* to the outcome.
* This is what **standardized coefficients** aim to do
* Basic idea = **standardized coefficients** = coefficients you’d have obtained if you’d converted all variables to z-scores before running the regression.7
* Strictly, standardize ALL **regressors** = every “thing” that has a regression coefficient associated w/ it in the model.
* For the regression models so far, each predictor variable maps onto exactly 1 regressor + vice versa.
* However, that’s not actually true in general (Chapter 16)
* Idea here = by converting all predictors to z-scores, they all go into the regression on the same scale, thereby removing the problem of having variables on different scales.
* Regardless of what the original variables were, a β value = 1 means an increase in the predictor of 1 SD will produce a corresponding 1 SD increase in the outcome variable.
* Therefore, if **variable A has a larger absolute value of β than variable B, it’s deemed to have a stronger relationship w/ the outcome.**
* Worth being a little cautious here, since this relies *very heavily* on the assumption that a 1 SD change is fundamentally the same kind of thing for all variables.
* It’s not always obvious that this is true.
* Could standardize all variables yourself + then run a regression, but there’s a much simpler way
* As it turns out, the β coefficient for a predictor X + outcome Y has a very simple formula



* Where σX = SD of the predictor + σY = SD of the outcome variable Y .
* This makes matters a lot simpler, + lsr includes **standardCoefs()** that computes β coefficients



* This clearly shows dan.sleep has a much stronger effect than baby.sleep
* However, *this is a perfect example of a situation where it would probably make sense to use the original coefficients b rather than the standardized coefficients β.*
* Mine + baby’s sleep are already on the same scale: # of hours slept. Why complicate matters by converting these to z-scores?

**15.8 Assumptions of regression**

* This linear regression model relies on several assumptions.
* **Normality** = Like half the models in statistics, standard linear regression relies on an assumption of normality.
* Specifically, it assumes the residuals are normally distributed.
* It’s actually okay if the predictors X + the outcome Y are non-normal, so long as the residuals are normal
* **Linearity** = A fundamental assumption of the linear regression model = the relationship between X + Y actually be linear
* Regardless of whether it’s a simple or multiple regression
* **Homogeneity of variance**/**Homoscedasticity** = regression model assumes each residual **ε**i is generated from a normal distribution w/ mean = 0 + (more importantly for the current purposes) w/ SD σ that is the same for every single residual.
* In practice, it’s impossible to test that every residual is identically distributed.
* Instead, we care that the SD of the residual is the same for all values of Yˆ + (if being especially paranoid) + for all values of every predictor X in the model.
* **Uncorrelated predictors** = in a multiple regression model, don’t want predictors to be too strongly correlated w/ each other.
* Isn’t technically an assumption, but in practice it’s required.
* Predictors too strongly correlated w/ each other (**collinearity**) can cause problems when evaluating the model
* **Residuals are independent of each other** = really a catch all assumption to the effect that there’s nothing else funny going on in the residuals.
* If there is something weird (e.g., residuals all depend heavily on some other unmeasured variable) going on, it might screw things up.
* **No bad outliers =** not technically an assumption of the model (rather implied by all the others), but there is an implicit assumption a regression model isn’t being too strongly influenced by 1 or 2 anomalous DP’s
* This raises questions about adequacy of a model + trustworthiness of the data in some cases

**15.9 Model checking**

* **Regression Diagnostics** = the art of checking that assumptions of a regression model have been met, figuring out how to fix the model if assumptions are violated, + generally to check that nothing funny is going on.
* It’s an “art” w/ good reason 🡺 it’s not easy + while there are a lot of fairly standardized tools one can use to diagnose + maybe even cure problems that ail a model (if there are any), you really need to exercise a certain amount of judgment when doing this.
* It’s easy to get lost in all the details of checking this thing or that thing + it’s quite exhausting to try to remember what all the different things are.
* This has the very nasty side effect that a lot of people get frustrated when trying to learn all the tools, so instead they decide not to do any model checking.
* This section draws quite heavily from Fox + Weisberg (2011) text, the book associated w/ the **car package**, which is notable for providing some excellent tools for regression diagnostics + talks about them in an admirably clear fashion.
* The majority of **regression diagnostics** revolve around looking at the **residuals**
* 4 kinds of residuals
* **Ordinary residuals** 🡺 The actual, raw residuals
* = the difference between the fitted value Yˆi + the observed value Yi

* 1 drawback to using ordinary residuals = they’re always on a different scale, depending on what the outcome variable is + how good the regression model is.
* Unless you’ve decided to run a regression model w/out an intercept term, the ordinary residuals will have mean = 0 but the variance is different for every regression.
* In a lot of contexts, especially if only interested in the pattern of the residuals + *not their actual values*, it’s convenient to estimate the standardized residuals
* **Pearson residual (**identical to ordinary**)**
* **Standardized residuals =** residuals normalized in such a way as to have standard deviation = 1
* Divide the ordinary residual by an *estimate* of the (population) SD of these residuals.

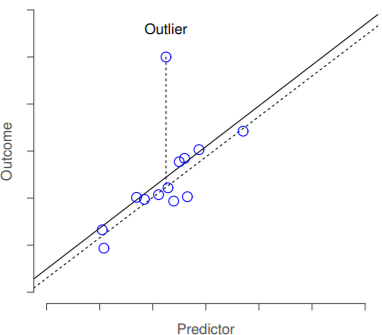
* where ˆσ = estimated population SD of the ordinary residuals, + h(i) = **hat value** of the i-th observation.
* For now, it’s enough to interpret standardized residuals as if we’d converted ordinary residuals to z-scores.
* Note the function uses a different name for the input argument, but it’s still just a linear regression object
* **Studentised/jackknifed** **residuals** = even fancier than standardized residuals.
* Idea = take the ordinary residual + divide it by some quantity in order to estimate some standardized notion of the residual, w/ a subtly different calculation

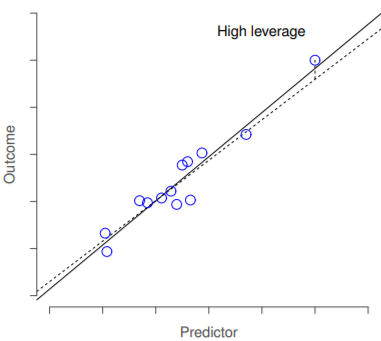


* Notice our estimate of the SD = **ˆσ(-i)** + this corresponds to the estimate of the residual SD that you’d have obtained if you just deleted the i-th observation from the data set.
* Sounds like the sort of thing that would be a nightmare to calculate, since it seems to be saying you must run N new regression models (even a modern CPU might grumble a bit at that, especially if w/ a large data set).
* Fortunately, this SD estimate is actually given by

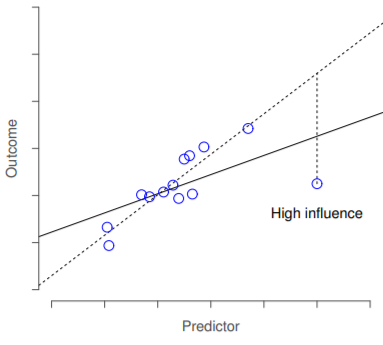
* You don’t often need to manually extract these residuals yourself, even though they’re at the heart of almost all regression diagnostics
* residuals(), rstandard() + rstudent() are all useful to know about, but most of the time various functions that run diagnostics will take care of these calculations for you.
* Even so, it’s always nice to know how to get hold of these things in case you ever need to do something non-standard
* 1 danger you can run into w/ linear regression models is an analysis might be disproportionately sensitive to a smallish number of unusual or anomalous observations.
* In linear regression = 3 conceptually distinct ways an observation might be called **anomalous**.
* All 3 are interesting but have rather different implications for an analysis.
* **1) Outlier** = an observation that is very different from what the regression model predicts.



* Dotted lines = regression line that would’ve been estimated w/out the anomalous observation included, + the corresponding residual (Studentised residual).
* Solid line = regression line w/ the anomalous observation included.
* outlier has an unusual value on the outcome (y-axis) but not the predictor (x-axis) + lies a long way from the regression line
* In practice, we operationalize this concept by saying an **outlier** = *an observation that has a very large Studentised residual*, **ε\***i.
* Outliers are interesting = a big outlier might correspond to junk data (variables might’ve been entered incorrectly or some other defect may be detectable)
* Shouldn’t throw an observation away just b/c it’s an outlier.
* But the fact it’s an outlier is often a cue to look more closely at that case + try to find out why it’s so different.
* 2) **High leverage** observation **=** an observation that is very different from all other observations
* *Doesn’t necessarily have to correspond to a large residual*
* If the observation happens to be unusual on all variables *in precisely the same way*, it can actually lie very close to the regression line.
* 
* Anomalous observation here is unusual *both* in terms of predictor (x) + outcome (y), but this unusualness is highly consistent w/ the pattern of correlations that exists among the other observations
* As a consequence, the observation falls very close to the regression line + does not distort it
* The **leverage** of an observation is operationalized in terms of its **hat value**, h(i)
* The formula for the hat value is rather complicated, but its interpretation is not
* The **hat matrix** = that matrix H that converts the vector of observed values Y into a vector of fitted values Yˆ, such that yˆ = H(y).
* Name comes from the fact this matrix *“puts a hat on Y”.*
* The **hat value** of the i-th observation = the i-th diagonal element of this matrix (technically should be writing it as h(i, i) rather than h(i)).



* **Hat value h(i)** = a measure of the extent to which the i-th observation is in control of where the regression line ends up going.
* Can extract hat values w/ **hatvalues( model = regression.2 )**
* In general, if an observation lies far away from others in terms of the predictor variables, it will have a large hat value
* As a rough guide, **high leverage** = when the hat value is > 2-3 times the average
* Note the sum of the hat values is constrained to be equal to K + 1
* High leverage points are also worth looking at in more detail but are much less likely to be a cause for concern unless they are *also outliers*.
* 3) **High Influence** observation = an outlier w/ high leverage (is very different to all others in some respect) + *also lies a long way from the regression line.*



* The anomalous observation is highly unusual on the predictor variable (x) + falls a long way from the regression line.
* Consequence = regression line is highly distorted even though (in this case) the anomalous observation is entirely typical in terms of the outcome variable (y).
* Notice the contrast to the previous 2 figures:
* Outliers don’t move a regression line much, + neither do high leverage points.
* But something that is an outlier AND has high leverage = big effect on a regression line.
* High influence = the biggest worry.
* We operationalize **influence** in terms of a measure known as **Cook’s distance**

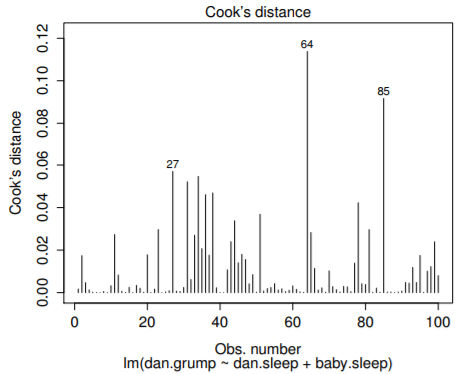
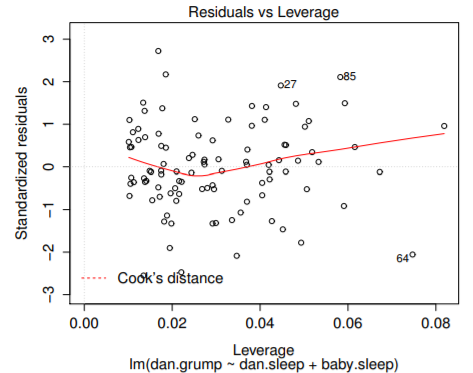


* Notice this = a multiplication of something measuring *outlier-ness* of an observation (left) + something measuring the *leverage* of the observation (right).
* To have a large Cook’s distance, an observation must be a fairly substantial outlier AND have high leverage.

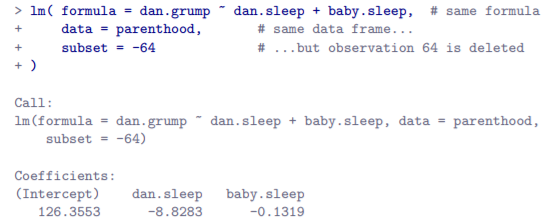


* Rough guide 🡺 **Cook’s distance > 1** = often considered a large value (quick + dirty rule), though 4/N has also been suggested as a possible rule of thumb.
* \*\*\*Don’t usually need to make use of these functions, since you can have R automatically draw the **critical plots**



* Left = Cook’s distance for every observation, one of the standard regression plots produced by **plot()** when input = a linear regression object + by setting **which = 4**
* **Right =** Residuals vs. leverage, another standard regression plot produced by **plot**() when input = a linear regression object + by setting **which = 5**
* Special mention should be made of **influenceIndexPlot**() + **influencePlot**() in the car package
* These produce somewhat more detailed pictures default plots above
* There’s also **outlierTest**() that tests to see if any of Studentised residuals are significantly larger than would be expected by chance.
* If you do have large values of Cook’s distance, what should you do? 🡪 No hard + fast rules.
* Probably 1st thing to do = try running the regression w/ that point excluded + see what happens to model performance + to the regression coefficients.
* If they are substantially different = time to start digging into the data set + notes from the study try to figure out WHY the point is so different.
* If convinced this 1 DP is badly distorting results, might consider excluding it, but that’s less than ideal unless you have a solid explanation for WHY this particular case is qualitatively different from the others + therefore deserves to be handled separately
* An alternative = run a **robust regression** (later version)
* Ex: Delete the observation from day 64 = observation w/ largest Cook’s distance



* Can see the regression coefficients have barely changed in comparison to earlier
* In other words = really don’t have any problem as far as anomalous data are concerned
* 15.9.3 Checking the normality of the residuals Like many of the statistical tools we’ve discussed in this book, regression models rely on a normality assumption. In this case, we assume that the residuals are normally distributed. The tools for testing this aren’t fundamentally different to those that we discussed earlier in Section 13.9. Firstly, I firmly 10Though special mention should be made of the influenceIndexPlot() + influencePlot() functions in the car package. These produce somewhat more detailed pictures than the default plots that I’ve shown here. There’s also an outlierTest() function that tests to see if any of the Studentised residuals are significantly larger than would be expected by chance. 11An alternative is to run a robust regression; I’ll discuss robust regression in a later version of this book. - 480 - 0 20 40 60 80 100 0.00 0.02 0.04 0.06 0.08 0.10 0.12 Obs. number Cook’s distance lm(dan.grump ~ dan.sleep + baby.sleep) Cook’s distance 64 85 27 Figure 15.8: Cook’s distance for every observation. This is one of the standard regression plots produced by the plot() function when the input is a linear regression object. It is obtained by setting which=4. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 0.00 0.02 0.04 0.06 0.08 −3 −2 −1 0 1 2 3 Leverage Standardized residuals lm(dan.grump ~ dan.sleep + baby.sleep) Cook’s distance Residuals vs Leverage 64 85 27 Figure 15.9: Residuals versus leverage. This is one of the standard regression plots produced by the plot() function when the input is a linear regression object. It is obtained by setting which=5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . - 481 - Value of residual Frequency −10 −5 0 5 10 0 2 4 6 8 10 12 Figure 15.10: A histogram of the (ordinary) residuals in the regression.2 model. These residuals look very close to being normally distributed, much moreso than is typically seen w/ real data. This shouldn’t surprise you... they aren’t real data, + they aren’t real residuals! . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . - 482 - −2 −1 0 1 2 −2 −1 0 1 2 3 Theoretical Quantiles Standardized residuals lm(dan.grump ~ dan.sleep + baby.sleep) Normal Q−Q 78 81 36 Figure 15.11: Plot of the theoretical quantiles according to the model, against the quantiles of the standardized residuals. This is one of the standard regression plots produced by the plot() function when the input is a linear regression object. It is obtained by setting which=2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . believe that it never hurts to draw an old fashioned histogram. The command I use might be something like this: > hist( x = residuals( regression.2 ), # data are the residuals + xlab = Value of residual, # x-axis label + main = , # no title + breaks = 20 # lots of breaks + ) The resulting plot is shown in Figure 15.10, + as you can see the plot looks pretty damn close to normal, almost unnaturally so. I could also run a Shapiro-Wilk test to check, using the shapiro.test() function; the W value of .99, at this sample size, is non-significant (p .84), again suggesting that the normality assumption isn’t in any danger here. As a third measure, we might also want to draw a QQ-plot using the qqnorm() function. The QQ plot is an excellent one to draw, + so you might not be surprised to discover that it’s one of the regression plots that we can produce using the plot() function: > plot( x = regression.2, which = 2 ) # Figure 15.11 The output is shown in Figure 15.11, showing the standardized residuals plotted as a function of their theoretical quantiles according to the regression model. The fact that the output appends the model specification to the picture is nice. - 483 - 50 60 70 80 40 50 60 70 80 90 Fitted Values Observed Values Figure 15.12: Plot of the fitted values against the observed values of the outcome variable. A straight line is what we’re hoping to see here. This looks pretty good, suggesting that there’s nothing grossly wrong, but there could be hidden subtle issues. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15.9.4 Checking the linearity of the relationship The third thing we might want to test is the linearity of the relationships between the predictors + the outcomes. There’s a few different things that you might want to do in order to check this. Firstly, it never hurts to just plot the relationship between the fitted values Yˆ i + the observed values Yi for the outcome variable, as illustrated in Figure 15.12. To draw this we could use the fitted.values() function to extract the Yˆ i values in much the same way that we used the residuals() function to extract the i values. So the commands to draw this figure might look like this: > yhat.2 <- fitted.values( object = regression.2 ) > plot( x = yhat.2, + y = parenthood$dan.grump, + xlab = Fitted Values, + ylab = Observed Values + ) One of the reasons I like to draw these plots is that they give you a kind of big picture view. If this plot looks approximately linear, then we’re probably not doing too badly (though that’s not to say that there aren’t problems). However, if you can see big departures from linearity here, then it strongly suggests that you need to make some changes. In any case, in order to get a more detailed picture it’s often more informative to look at the relationship between the fitted values + the residuals themselves. Again, we could draw this plot using low - 484 - 50 60 70 80 −10 −5 0 5 10 Fitted values Residuals lm(dan.grump ~ dan.sleep + baby.sleep) Residuals vs Fitted 78 81 36 Figure 15.13: Plot of the fitted values against the residuals for regression.2, w/ a line showing the relationship between the two. If this is horizontal + straight, then we can feel reasonably confident that the average residual for all fitted values is more or less the same. This is one of the standard regression plots produced by the plot() function when the input is a linear regression object. It is obtained by setting which=1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . level commands, but there’s an easier way. Just plot() the regression model, + select which = 1: > plot(x = regression.2, which = 1) # Figure 15.13 The output is shown in Figure 15.13. As you can see, not only does it draw the scatterplot showing the fitted value against the residuals, it also plots a line through the data that shows the relationship between the two. Ideally, this should be a straight, perfectly horizontal line. There’s some hint of curvature here, but it’s not clear whether or not we be concerned. A somewhat more advanced version of the same plot is produced by the residualPlots() function in the car package. This function not only draws plots comparing the fitted values to the residuals, it does so for each individual predictor. The command is > residualPlots( model = regression.2 ) # Figure 15.14 + the resulting plots are shown in Figure 15.14. Note that this function also reports the results of a bunch of curvature tests. For a predictor variable X in some regression model, this test is equivalent to adding a new predictor to the model corresponding to X2 , + running the t-test on the b coefficient associated w/ this new predictor. If it comes up significant, it implies that there is some nonlinear relationship between the variable + the residuals. For what it’s worth, here’s what you get for the regression.2 model: - 485 - 5 6 7 8 9 −10 0 5 10 dan.sleep Pearson residuals 4 6 8 10 12 −10 0 5 10 baby.sleep Pearson residuals 50 60 70 80 −10 0 5 10 Fitted values Pearson residuals Figure 15.14: Plot of the fitted values against the residuals for regression.2, along w/ similar plots for the two predictors individually. This plot is produced by the residualPlots() function in the car package. Note that it refers to the residuals as Pearson residuals, but in this context these are the same as ordinary residuals. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Test stat Pr(>|t|) dan.sleep 2.160 0.033 baby.sleep -0.545 0.587 Tukey test 2.162 0.031 The third line here is the Tukey test, which is basically the same test, except that instead of squaring one of the predictors + adding it to the model, you square the fitted-value. In any case, the fact that the curvature tests have come up significant is hinting that the curvature that we can see in Figures 15.13 + 15.14 is genuine;12 although it still bears remembering that the pattern in Figure 15.12 is pretty damn straight: in other words the deviations from linearity are pretty small, + probably not worth worrying about. 12And, if you take the time to check the residualPlots() for regression.1, it’s pretty clear that this isn’t some wacky distortion being caused by the fact that baby.sleep is a useless predictor variable. It’s an actual nonlinearity in the relationship between dan.sleep + dan.grump. - 486 - In a lot of cases, the solution to this problem (+ many others) is to transform one or more of the variables. We discussed the basics of variable transformation in Sections 7.2 + 7.3, but I do want to make special note of one additional possibility that I didn’t mention earlier: the Box-Cox transform. The Box-Cox function is a fairly simple one, but it’s very widely used fpx, λq x λ ´ 1 λ for all values of λ except λ 0. When λ 0 we just take the natural logarithm (i.e., lnpxq). You can calculate it using the boxCox() function in the car package. Better yet, if what you’re trying to do is convert a data to normal, or as normal as possible, there’s the powerTransform() function in the car package that can estimate the best value of λ. Variable transformation is another topic that deserves a fairly detailed treatment, but (again) due to deadline constraints, it will have to wait until a future version of this book. 15.9.5 Checking the homogeneity of variance The regression models that we’ve talked about all make a homogeneity of variance assumption: the variance of the residuals is assumed to be constant. The default plot that R provides to help w/ doing this (which = 3 when using plot()) shows a plot of the square root of the size of the residual a |i |, as a function of the fitted value Yˆ i . We can produce the plot using the following command, > plot(x = regression.2, which = 3) + the resulting plot is shown in Figure 15.15. Note that this plot actually uses the standardized residuals (i.e., converted to z scores) rather than the raw ones, but it’s immaterial from our point of view. What we’re looking to see here is a straight, horizontal line running through the middle of the plot. A slightly more formal approach is to run hypothesis tests. The car package provides a function called ncvTest() (non-constant variance test) that can be used for this purpose (Cook & Weisberg, 1983). I won’t explain the details of how it works, other than to say that the idea is that what you do is run a regression to see if there is a relationship between the squared residuals i + the fitted values Yˆ i , or possibly to run a regression using all of the original predictors instead of just Yˆ i . 13 Using the default settings, the ncvTest() looks for a relationship between Yˆ i + the variance of the residuals, making it a straightforward analogue of Figure 15.15. So if we run it for our model, > ncvTest( regression.2 ) Non-constant Variance Score Test Variance formula: ~ fitted.values Chisquare = 0.09317511 Df = 1 p = 0.7601788 We see that our original impression was right: there’s no violations of homogeneity of variance in this data. It’s a bit beyond the scope of this chapter to talk too much about how to deal w/ violations of homogeneity of variance, but I’ll give you a quick sense of what you need to consider. The main thing to worry about, if homogeneity of variance is violated, is that the standard error estimates associated w/ the regression coefficients are no longer entirely reliable, + so your t tests for the coefficients aren’t quite right either. A simple fix to the problem is to make use of a heteroscedasticity corrected covariance matrix when estimating the standard errors. These are often called sandwich estimators, for reasons 13Note that the underlying mechanics of the test aren’t the same as the ones I’ve described for regressions; the goodness of fit is assessed using what’s known as a score-test not an F-test, + the test statistic is (approximately) χ 2 distributed if there’s no relationship - 487 - 50 60 70 80 0.0 0.5 1.0 1.5 Fitted values Standardized residuals lm(dan.grump ~ dan.sleep + baby.sleep) Scale−Location 78 81 36 Figure 15.15: Plot of the fitted values (model predictions) against the square root of the abs standardized residuals. This plot is used to diagnose violations of homogeneity of variance. If the variance is really constant, then the line through the middle should be horizontal + flat. This is one of the standard regression plots produced by the plot() function when the input is a linear regression object. It is obtained by setting which=3. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . that only make sense if you understand the maths at a low level14 You don’t need to understand what this means (not for an introductory class), but it might help to note that there’s a hccm() function in the car() package that does it. Better yet, you don’t even need to use it. You can use the coeftest() function in the lmtest package, but you need the car package loaded: > coeftest( regression.2, vcov= hccm ) t test of coefficients: Estimate Std. Error t value Pr(>|t|) (Intercept) 125.965566 3.247285 38.7910 vif( mod = regression.2 ) dan.sleep baby.sleep 1.651038 1.651038 + since the square root of 1.65 is 1.28, we see that the correlation between our two predictors isn’t causing much of a problem. To give a sense of how we could end up w/ a model that has bigger collinearity problems, suppose I were to run a much less interesting regression model, in which I tried to predict the day on which the data were collected, as a function of all the other variables in the data set. To see why this would be a bit of a problem, let’s have a look at the correlation matrix for all four variables: > cor( parenthood ) dan.sleep baby.sleep dan.grump day dan.sleep 1.00000000 0.62794934 -0.90338404 -0.09840768 baby.sleep 0.62794934 1.00000000 -0.56596373 -0.01043394 dan.grump -0.90338404 -0.56596373 1.00000000 0.07647926 day -0.09840768 -0.01043394 0.07647926 1.00000000 We have some fairly large correlations between some of our predictor variables! When we run the regression model + look at the VIF values, we see that the collinearity is causing a lot of uncertainty about the coefficients. First, run the regression... > regression.3 <- lm( day ~ baby.sleep + dan.sleep + dan.grump, parenthood ) + second, look at the VIFs... > vif( regression.3 ) baby.sleep dan.sleep dan.grump 1.651064 6.102337 5.437903 Yep, that’s some mighty fine collinearity you’ve got there. - 489 - 15.10 Model selection One fairly major problem that remains is the problem of model selection. That is, if we have a data set that contains several variables, which ones should we include as predictors, + which ones should we not include? In other words, we have a problem of variable selection. In general, model selection is a complex business, but it’s made somewhat simpler if we restrict ourselves to the problem of choosing a subset of the variables that ought to be included in the model. Nevertheless, I’m not going to try covering even this reduced topic in a lot of detail. Instead, I’ll talk about two broad principles that you need to think about; + then discuss one concrete tool that R provides to help you select a subset of variables to include in your model. Firstly, the two principles: • It’s nice to have an actual substantive basis for your choices. That is, in a lot of situations you the researcher have good reasons to pick out a smallish number of possible regression models that are of theoretical interest; these models will have a sensible interpretation in the context of your field. Never discount the importance of this. Statistics serves the scientific process, not the other way around. • To the extent that your choices rely on statistical inference, there is a trade off between simplicity + goodness of fit. As you add more predictors to the model, you make it more complex; each predictor adds a new free parameter (i.e., a new regression coefficient), + each new parameter increases the model’s capacity to absorb random variations. So the goodness of fit (e.g., R2 ) continues to rise as you add more predictors no matter what. If you want your model to be able to generalize well to new observations, you need to avoid throwing in too many variables. This latter principle is often referred to as Ockham’s razor, + is often summarised in terms of the following pithy saying: do not multiply entities beyond necessity. In this context, it means: don’t chuck in a bunch of largely irrelevant predictors just to boost your R2 . Hm. Yeah, the original was better. In any case, what we need is an actual mathematical criterion that will implement the qualitative principle behind Ockham’s razor in the context of selecting a regression model. As it turns out there are several possibilities. The one that I’ll talk about is the Akaike information criterion (AIC; Akaike, 1974) simply it’s the default one used in the R function step(). In the context of a linear regression model (+ ignoring terms that don’t depend on the model in any way!), the AIC for a model that has K predictor variables plus an intercept is:15 AIC SSres σˆ 2 ` 2K The smaller the AIC value, the better the model performance is. If we ignore the low level details, it’s fairly obvious what the AIC does: on the left we have a term that increases as the model predictions get worse; on the right we have a term that increases as the model complexity increases. The best model is the one that fits the data well (low residuals; left hand side) using as few predictors as possible (low K; right hand side). In short, this is a simple implementation of Ockham’s razor. 15Note, however, that the step() function computes the full version of AIC, including the irrelevant constants that I’ve dropped here. As a consequence this equation won’t correctly describe the AIC values that you see in the outputs here. However, if you calculate the AIC values using my formula for two different regression models + take the difference between them, this will be the same as the differences between AIC values that step() reports. In practice, this is all you care about: the actual value of an AIC statistic isn’t very informative, but the differences between two AIC values are useful, since these provide a measure of the extent to which one model outperforms another. - 490 - 15.10.1 Backward elimination Okay, let’s have a look at the step() function at work. In this example I’ll keep it simple + use only the basic backward elimination approach. That is, start w/ the complete regression model, including all possible predictors. Then, at each step we try all possible ways of removing one of the variables, + whichever of these is best (in terms of lowest AIC value) is accepted. This becomes our new regression model; + we then try all possible deletions from the new model, again choosing the option w/ lowest AIC. This process continues until we end up w/ a model that has a lower AIC value than any of the other possible models that you could produce by deleting one of its predictors. Let’s see this in action. First, I need to define the model from which the process starts. > full.model <- lm( formula = dan.grump ~ dan.sleep + baby.sleep + day, + data = parenthood + ) That’s nothing terribly new: yet another regression. Booooring. Still, we do need to do it: the object argument to the step() function will be this regression model. W/ this in mind, I would call the step() function using the following command: > step( object = full.model, # start at the full model + direction = backward # allow it remove predictors but not add them + ) although in practice I didn’t need to specify direction backward is the default. The output is somewhat lengthy, so I’ll go through it slowly. Firstly, the output reports the AIC value for the current best model: Start: AIC=299.08 dan.grump ~ dan.sleep + baby.sleep + day That’s our starting point. Since small AIC values are good, we want to see if we can get a value smaller than 299.08 by deleting one of those three predictors. So what R does is try all three possibilities, calculate the AIC values for each one, + then print out a short table w/ the results: Df Sum of Sq RSS AIC - baby.sleep 1 0.1 1837.2 297.08 - day 1 1.6 1838.7 297.16 1837.1 299.08 - dan.sleep 1 4909.0 6746.1 427.15 To read this table, it helps to note that the text in the left hand column is telling you what change R made to the regression model. So the line that reads is the actual model we started w/, + you can see on the right hand side that this still corresponds to an AIC value of 299.08 (obviously). The other three rows in the table correspond to the other three models that it looked at: it tried removing the baby.sleep variable, which is indicated by - baby.sleep, + this produced an AIC value of 297.08. That was the best of the three moves, so it’s at the top of the table. So, this move is accepted, + now we start again. There are two predictors left in the model, dan.sleep + day, so it tries deleting those: Step: AIC=297.08 dan.grump ~ dan.sleep + day Df Sum of Sq RSS AIC - day 1 1.6 1838.7 295.17 1837.2 297.08 - dan.sleep 1 8103.0 9940.1 463.92 - 491 - Okay, so what we can see is that removing the day variable lowers the AIC value from 297.08 to 295.17. So R decides to keep that change too, + moves on: Step: AIC=295.17 dan.grump ~ dan.sleep Df Sum of Sq RSS AIC 1838.7 295.17 - dan.sleep 1 8159.9 9998.6 462.50 This time around, there’s no further deletions that can actually improve the AIC value. So the step() function stops, + prints out the result of the best regression model it could find: Call: lm(formula = dan.grump ~ dan.sleep, data = parenthood) Coefficients: (Intercept) dan.sleep 125.956 -8.937 which is (perhaps not all that surprisingly) the regression.1 model that we started w/ at the beginning of the chapter. 15.10.2 Forward selection As an alternative, you can also try forward selection. This time around we start w/ the smallest possible model as our start point, + only consider the possible additions to the model. However, there’s one complication: you also need to tell step() what the largest possible model you’re willing to entertain is, using the scope argument. The simplest usage is like this: > null.model <- lm( dan.grump ~ 1, parenthood ) # intercept only. > step( object = null.model, # start w/ null.model + direction = forward, # only consider addition moves + scope = dan.grump ~ dan.sleep + baby.sleep + day # largest model allowed + ) If I do this, the output takes on a similar form, but now it only considers addition (+) moves rather than deletion (-) moves: Start: AIC=462.5 dan.grump ~ 1 Df Sum of Sq RSS AIC + dan.sleep 1 8159.9 1838.7 295.17 + baby.sleep 1 3202.7 6795.9 425.89 9998.6 462.50 + day 1 58.5 9940.1 463.92 Step: AIC=295.17 dan.grump ~ dan.sleep Df Sum of Sq RSS AIC 1838.7 295.17 - 492 - + day 1 1.55760 1837.2 297.08 + baby.sleep 1 0.02858 1838.7 297.16 Call: lm(formula = dan.grump ~ dan.sleep, data = parenthood) Coefficients: (Intercept) dan.sleep 125.956 -8.937 As you can see, it’s found the same model. In general though, forward + backward selection don’t always have to end up in the same place. 15.10.3 A caveat Automated variable selection methods are seductive things, especially when they’re bundled up in (fairly) simple functions like step(). They provide an element of objectivity to your model selection, + that’s kind of nice. Unfortunately, they’re sometimes used as an excuse for thoughtlessness. No longer do you have to think carefully about which predictors to add to the model + what the theoretical basis for their inclusion might be... everything is solved by the magic of AIC. + if we start throwing around phrases like Ockham’s razor, well, it sounds like everything is wrapped up in a nice neat little package that no-one can argue w/. Or, perhaps not. Firstly, there’s very little agreement on what counts as an appropriate model selection criterion. When I was taught backward elimination as an undergraduate, we used F-tests to do it, that was the default method used by the software. The default in the step() function is AIC, + since this is an introductory text that’s the only method I’ve described, but the AIC is hardly the Word of the Gods of Statistics. It’s an approximation, derived under certain assumptions, + it’s guaranteed to work only for large samples when those assumptions are met. Alter those assumptions + you get a different criterion, like the BIC for instance. Take a different approach again + you get the NML criterion. Decide that you’re a Bayesian + you get model selection based on posterior odds ratios. Then there are a bunch of regression specific tools that I haven’t mentioned. + so on. All of these different methods have strengths + weaknesses, + some are easier to calculate than others (AIC is probably the easiest of the lot, which might account for its popularity). Almost all of them produce the same answers when the answer is obvious but there’s a fair amount of disagreement when the model selection problem becomes hard. What does this mean in practice? Well, you could go + spend several years teaching yourself the theory of model selection, learning all the ins + outs of it; so that you could finally decide on what you personally think the right thing to do is. Speaking as someone who actually did that, I wouldn’t recommend it: you’ll probably come out the other side even more confused than when you started. A better strategy is to show a bit of common sense... if you’re staring at the results of a step() procedure, + the model that makes sense is close to having the smallest AIC, but is narrowly defeated by a model that doesn’t make any sense... trust your instincts. Statistical model selection is an inexact tool, + as I said at the beginning, interpretability matters. 15.10.4 Comparing two regression models An alternative to using automated model selection procedures is for the researcher to explicitly select two or more regression models to compare to each other. You can do this in a few different ways, depending on what research question you’re trying to answer. Suppose we want to know whether or not the amount of sleep that my son got has any relationship to my grumpiness, over + above what we - 493 - might expect from the amount of sleep that I got. We also want to make sure that the day on which we took the measurement has no influence on the relationship. That is, we’re interested in the relationship between baby.sleep + dan.grump, + from that perspective dan.sleep + day are nuisance variable or covariates that we want to control for. In this situation, what we would like to know is whether dan.grump ~ dan.sleep + day + baby.sleep (which I’ll call Model 1, or M1) is a better regression model for these data than dan.grump ~ dan.sleep + day (which I’ll call Model 0, or M0). There are two different ways we can compare these two models, one based on a model selection criterion like AIC, + the other based on an explicit hypothesis test. I’ll show you the AIC based approach first it’s simpler, + follows naturally from the step() function that we saw in the last section. The first thing I need to do is actually run the regressions: > M0 <- lm( dan.grump ~ dan.sleep + day, parenthood ) > M1 <- lm( dan.grump ~ dan.sleep + day + baby.sleep, parenthood ) Now that I have my regression models, I could use the summary() function to run various hypothesis tests + other useful statistics, just as we have discussed throughout this chapter. However, since the current focus on model comparison, I’ll skip this step + go straight to the AIC calculations. Conveniently, the AIC() function in R lets you input several regression models, + it will spit out the AIC values for each of them:16 > AIC( M0, M1 ) df AIC M0 4 582.8681 M1 5 584.8646 Since Model 0 has the smaller AIC value, it is judged to be the better model for these data. A somewhat different approach to the problem comes out of the hypothesis testing framework. Suppose you have two regression models, where one of them (Model 0) contains a subset of the predictors from the other one (Model 1). That is, Model 1 contains all of the predictors included in Model 0, plus one or more additional predictors. When this happens we say that Model 0 is nested w/in Model 1, or possibly that Model 0 is a submodel of Model 1. Regardless of the terminology what this means is that we can think of Model 0 as a null hypothesis + Model 1 as an alternative hypothesis. + in fact we can construct an F test for this in a fairly straightforward fashion. We can fit both models to the data + obtain a residual sum of squares for both models. I’ll denote these as SSp0q res + SSp1q res respectively. The superscripting here just indicates which model we’re talking about. Then our F statistic is F pSSp0q res ´ SSp1q resq{k pSSp1q resq{pN ´ p ´ 1q where N is the number of observations, p is the number of predictors in the full model (not including the intercept), + k is the difference in the number of parameters between the two models.17 The degrees of freedom here are k + N ´ p ´ 1. Note that it’s often more convenient to think about the difference 16While I’m on this topic I should point out that there is also a function called BIC() which computes the Bayesian information criterion (BIC) for the models. So you could type BIC(M0,M1) + get a very similar output. In fact, while I’m not particularly impressed w/ either AIC or BIC as model selection methods, if you do find yourself using one of these two, the empirical evidence suggests that BIC is the better criterion of the two. In most simulation studies that I’ve seen, BIC does a much better job of selecting the correct model. 17It’s worth noting in passing that this same F statistic can be used to test a much broader range of hypotheses than those that I’m mentioning here. Very briefly: notice that the nested model M0 corresponds to the full model M1 when we constrain some of the regression coefficients to zero. It is sometimes useful to construct submodels by placing other kinds of constraints on the regression coefficients. For instance, maybe two different coefficients might have to sum to zero, or something like that. You can construct hypothesis tests for those kind of constraints too, but it is somewhat more complicated + the sampling distribution for F can end up being something known as the non-central F distribution, which is waaaaay beyond the scope of this book! All I want to do is alert you to this possibility. - 494 - between those two SS values as a sum of squares in its own right. That is: SS∆ SSp0q res ´ SSp1q res The reason why this his helpful is that we can express SS∆ a measure of the extent to which the two models make different predictions about the the outcome variable. Specifically: SS∆ ÿ i ´ yˆ p1q i ´ yˆ p0q i ¯2 where ˆy p0q i is the fitted value for yi according to model M0 + ˆy p1q i is the is the fitted value for yi according to model M1. Okay, so that’s the hypothesis test that we use to compare two regression models to one another. Now, how do we do it in R? The answer is to use the anova() function. All we have to do is input the two models that we want to compare (null model first): > anova( M0, M1 ) Analysis of Variance Table Model 1: dan.grump ~ dan.sleep + day Model 2: dan.grump ~ dan.sleep + day + baby.sleep Res.Df RSS Df Sum of Sq F Pr(>F) 1 97 1837.2 2 96 1837.1 1 0.063688 0.0033 0.9541 Note that, just like we saw w/ the output from the step() function, R has used the acronym RSS to refer to the residual sum of squares from each model. That is, RSS in this output corresponds to SSres in the formula above. Since we have p ą .05 we retain the null hypothesis (M0). This approach to regression, in which we add all of our covariates into a null model, + then add the variables of interest into an alternative model, + then compare the two models in hypothesis testing framework, is often referred to as hierarchical regression. 15.11 Summary • Basic ideas in linear regression + how regression models are estimated (Sections 15.1 + 15.2). • Multiple linear regression (Section 15.3). • Measuring the overall performance of a regression model using R2 (Section 15.4) • Hypothesis tests for regression models (Section 15.5) • Calculating CIs for regression coefficients, + standardized coefficients (Section 15.7) • The assumptions of regression (Section 15.8) + how to check them (Section 15.9) • Selecting a regression model (Section 15.10)