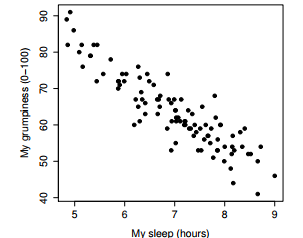
***Learning Statistics with R - University of Adelaide***

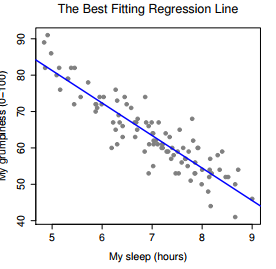
***Part V – Statistical Tools***

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

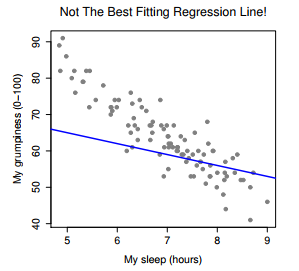
* **Linear regression** = standard tool statisticians rely on when analyzing the relationship between interval scale **predictors** + interval scale **outcomes**.
* Stripped to bare essentials, linear regression models = basically a slightly fancier version of the **Pearson correlation** but are much 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**

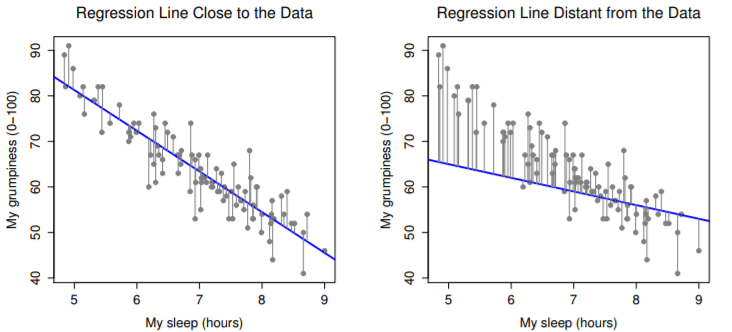


* In turn, we can write down the complete linear regression model as:



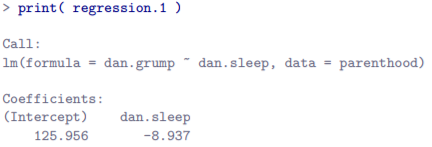
**15.2 Estimating a linear regression model**

* 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

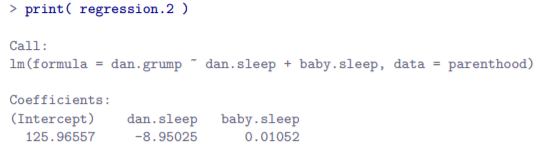
**15.3 Multiple linear regression**

* 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

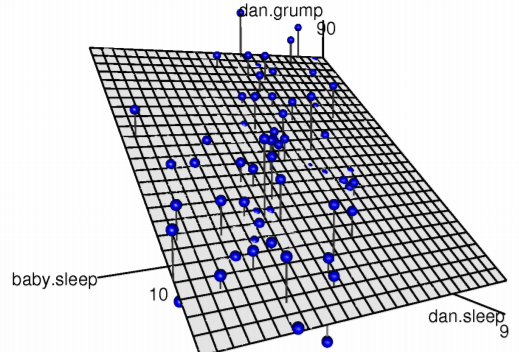


* Now have 3 coefficients to be estimated: b0 = intercept, b1 = coefficient associated w/ my sleep, + b2 = coefficient associated w/ my son’s sleep.
* # of coefficients to be estimated has changed, but basic idea of how the estimation works is unchanged: estimated coefficients ˆb0, ˆb1 + ˆb2 = those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



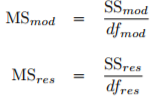
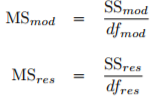
* If our regression model has K predictors, the alternative model is described using the usual formula for a multiple regression model:



* 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**.



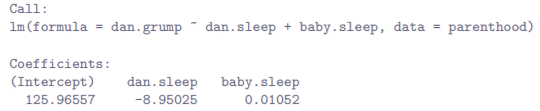
* Just like we w/ ANOVA, convert the sums of squares in to **mean squares** by dividing by dF

* 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).



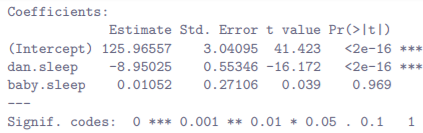
* If the CLT is kind to us, we might be able to guess that the sampling distribution of the *estimated* regression coefficient, **ˆb**, is a normal distribution w/ mean centered on b.
* This would mean that if the null were true, the sampling distribution of ˆb has mean = 0 + unknown SD.
* Assuming we 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.
* 

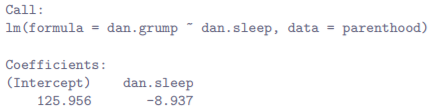








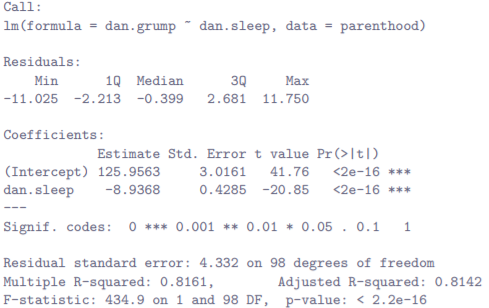
* Reminds what the actual regression model is + provides a quick summary of the residuals (**ε**(i) values), which can be convenient as a quick + dirty check that the model is okay.
* Remember, we *assumed these residuals were normally distributed, w/ mean 0*.
* In particular, it’s worth quickly checking 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



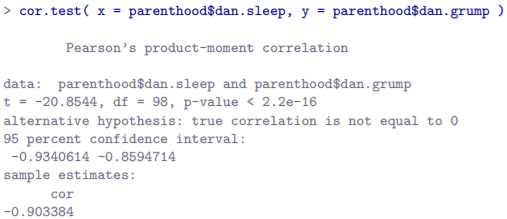
* Note that, although R has done multiple tests here, it hasn’t done a **Bonferroni correction** or anything.
* The above are *standard* one-sample t-tests w/ a 2-sided alternative.
* If you want to make corrections for multiple tests, you need to do that yourself

**15.6 Testing the significance of a correlation**

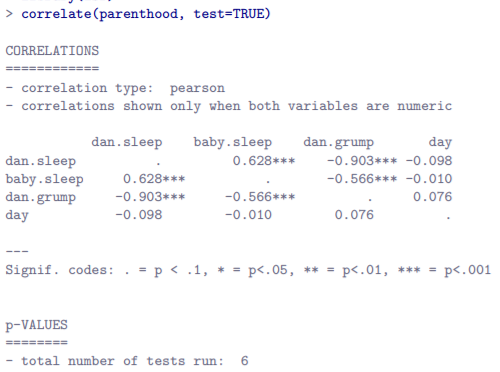
* Hypothesis tests for a single correlation
* Worth very briefly returning to the point made earlier 🡪 **Pearson correlations** = basically the same thing as **linear regressions** *w/ only a* ***single predictor*** *added to the model*.
* This means that the hypothesis tests described in a regression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

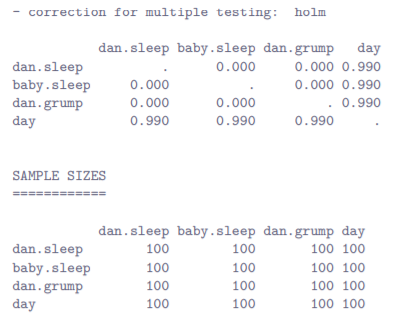


* Now compare to the output of **cor.test() 🡪** runs a hypothesis test to see if an observed correlation between 2 variables is significantly different from 0.



* 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**.





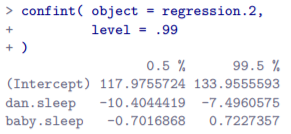
* The output contains 3 matrices.
* 1) **Correlation Matrix**
* 2) Matrix of p-values, using the Holm method to correct for multiple comparisons.
* change correction applied by specifying **p.adjust.method** argument
* 3) Matrix indicating sample size (# of pairwise complete cases) that contributed to 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.

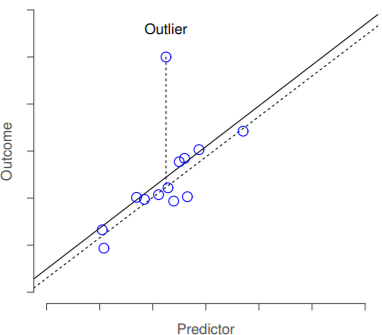
* where ˆσ = estimated population SD of the ordinary residuals, + h(i) = **hat value** of the i-th observation.
* For now, it’s enough to interpret standardized residuals as if we’d converted ordinary residuals to z-scores.
* Note the function uses a different name for the input argument, but it’s still 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

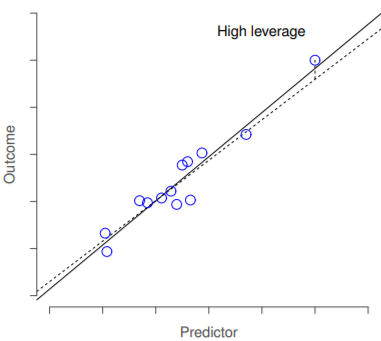


* Notice our estimate of the SD = **ˆσ(-i)** + this corresponds to the estimate of the residual SD that you’d have obtained if you just deleted the i-th observation from the data set.
* Sounds like the sort of thing that would be a nightmare to calculate, since it seems to be saying you must run N new 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

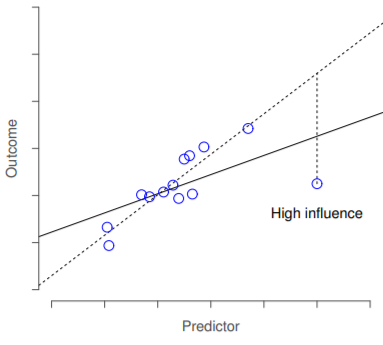
* You don’t often need to manually extract these residuals yourself, even though they’re at the heart of almost all regression diagnostics
* residuals(), rstandard() + rstudent() are all useful to know about, but most of the time various functions that run diagnostics will take care of these calculations 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**

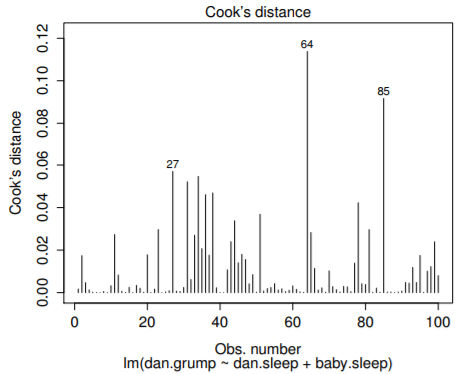
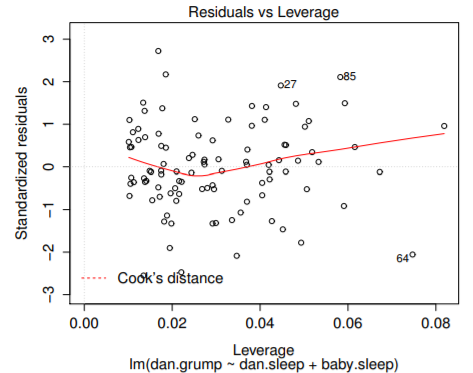


* Notice this = a multiplication of something measuring *outlier-ness* of an observation (left) + something measuring the *leverage* of the observation (right).
* To have a large Cook’s distance, an observation must be a fairly substantial outlier AND have high leverage.

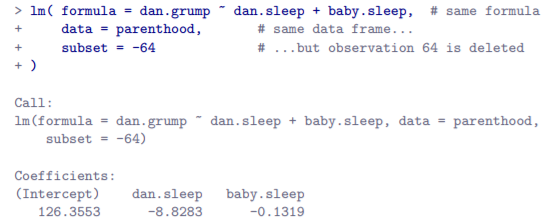


* Rough guide 🡺 **Cook’s distance > 1** = often considered a large value (quick + dirty rule), though 4/N has also been suggested as a possible rule of thumb.
* \*\*\*Don’t usually need to make use of these functions, since you can have R automatically draw the **critical plots**

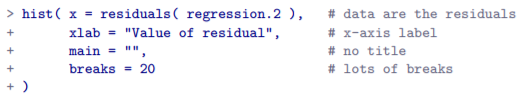


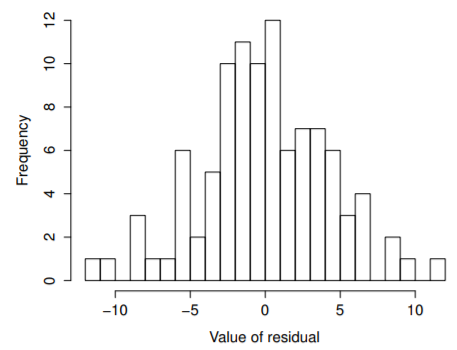
 

* Left = Cook’s distance for every observation, one of the standard regression plots produced by **plot()** when input = a linear regression object + by setting **which = 4**
* **Right =** Residuals vs. leverage, another standard regression plot produced by **plot**() when input = a linear regression 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

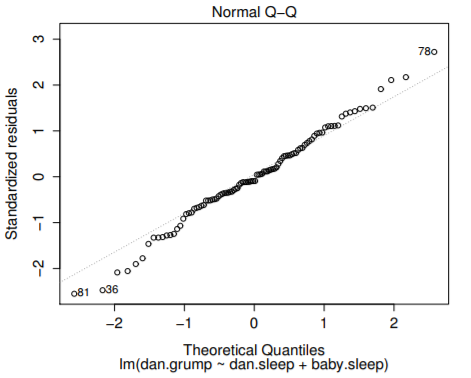


* Can see the regression coefficients have barely changed in comparison to earlier
* i.e. really don’t have any problem as far as anomalous data are concerned
* Checking normality of the residuals
* Like many statistical tools, regression models rely on a normality assumption.
* In this case, we assume the residuals are normally distributed.
* The tools for testing this aren’t fundamentally different to those in Section 13.9.
* Firstly, it never hurts to draw a histogram.

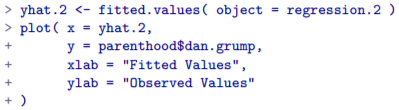


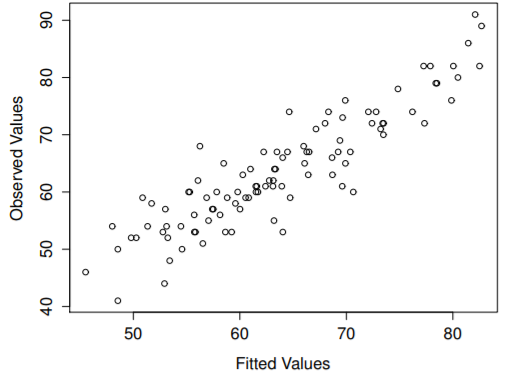


* Looks pretty damn close to normal, almost unnaturally so.
* Could also run a **Shapiro-Wilk test** to check w/ shapiro.test() 🡪 W= .99, which, at this sample size, is NON-significant (p = .84), again suggesting the normality assumption is met
* 3rd measure = also draw a QQ-plot w/ qqnorm()
* 



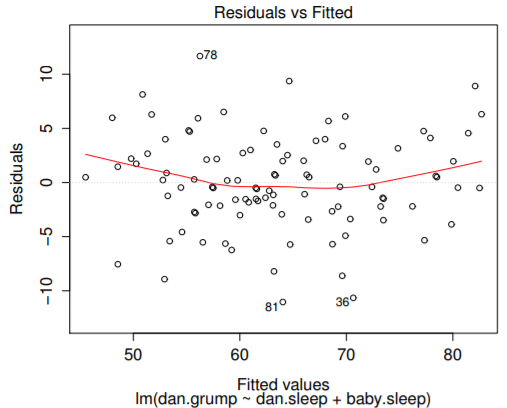
* This shows the standardized residuals plotted as a function of their theoretical quantiles according to the regression model.
* Checking linearity of the relationship
* The 3rd thing to test = **linearity of the relationships** between predictors + outcomes.
* There’s a few different things you might want to do in order to check this.
* 1st = never hurts to just plot the relationship between fitted values Yˆi + observed values Yi
* Could use **fitted.values()** to extract the Yˆ(i) values in much the same way we used **residuals**()to extract the i values





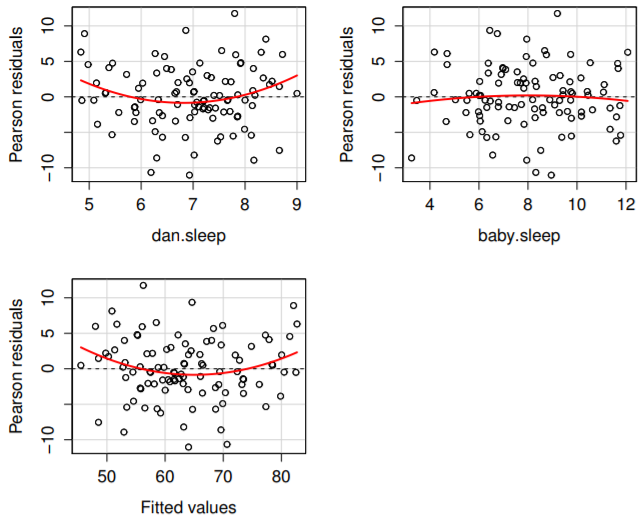
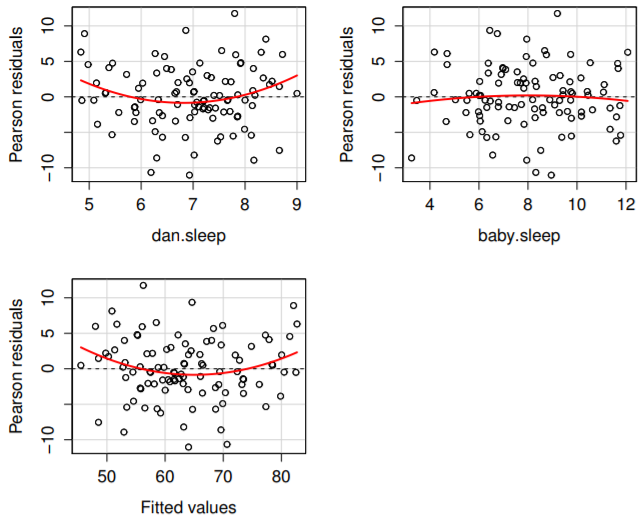
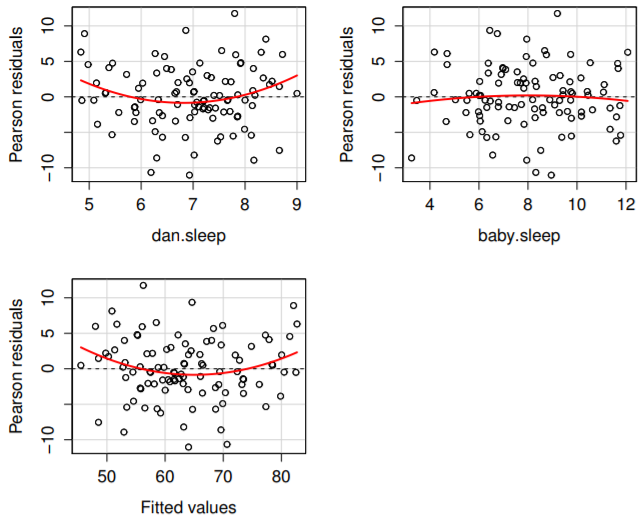
* These give a kind of big picture view.
* If this plot looks approximately linear, we’re probably not doing too badly (that’s not to say there aren’t any problems).
* However, big departures from linearity = strongly suggests you need to make some changes
* In any case, to get a more detailed picture, it’s often more informative to look at *the relationship between the fitted values + the residuals themselves*

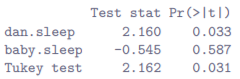




* Plot of fitted values against residuals for the model, w/ a line showing the relationship between the 2.
* If this line is horizontal + straight = can feel reasonably confident the average residual for all fitted values = more or less the same.
* Ideally, this should be a straight, perfectly horizontal line.
* We have some hint of curvature, but it’s not clear if we should be concerned.
* A somewhat more advanced version of this is produced by **residualPlots**() in **car** package.



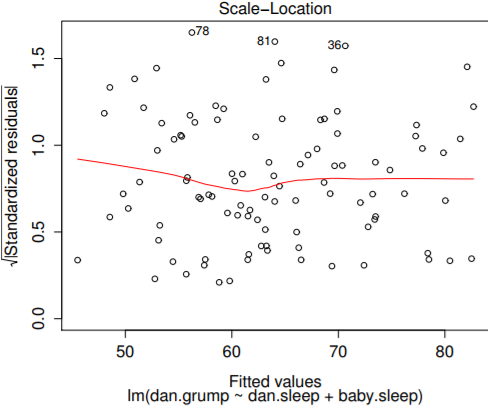


* This function not only draws plots comparing fitted values to residuals, but does so for each individual predictor.
* Note this function also reports the results of a bunch of curvature tests.
* *For a predictor variable X in some regression model, this test is equivalent to adding a new predictor to the model corresponding to X^2 + running a t-test on the b coefficient associated w/ this new predictor.*
* If it comes up significant, it implies there is some nonlinear relationship between the variable + the residuals.
* **Tukey test**, basically the same test as described above, except instead of squaring 1 of the predictors + adding it to the model, it *squares the fitted-value*.
* In any case, the fact that the *curvature tests = significant is hinting the curvature we can see in plots is genuine*, although it still bears remembering that the pattern in the relationship between fitted values Yˆi + observed values Yi is pretty damn straight
* i.e. the deviations from linearity are pretty small, + probably not worth worrying about.
* If you take the time to check residualPlots() for our 1st regression model, it’s pretty clear 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
* In a lot of cases, the solution to this problem (+ many others) is to **transform** 1 or more variables
* Ex: **Box-Cox transform** = a fairly simple one, but it’s very widely used via **boxCox**() in car package

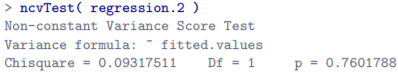


* for all values of λ except λ = 0.
* When λ = 0, we just take the natural logarithm, ln(x). You can calculate it using the.
* Better yet, if you’re trying to convert a data to normal/as normal as possible, there’s **powerTransform**() in car package that *can estimate the best value of λ*.
* **Variable transformation** = another topic that deserves detailed treatment (future version)
* Checking the homogeneity of variance/homoscedasticity
* Regression models we’ve talked about all make a homoscedasticity assumption = variance of the residuals is assumed to be constant.
* Default plot R provides to help w/ doing this (**which = 3** w/ **plot()**) shows a plot of the square root of the size of the residual  as a function of the fitted value Yˆi .





* Plot of fitted values (model predictions) against the square root of the standardized residuals.
* This is used to diagnose violations of homogeneity of variance.
* If variance is really constant, the line through the middle should be horizontal + flat.
* Note this plot actually uses the **standardized residuals** (converted to z scores) rather than raw,
* Looking for a straight, horizontal line running through the middle of the plot.
* A slightly more formal approach = run hypothesis tests.
* car package provides **ncvTest**() (**non-constant variance test**) that can be used for this purpose
* the idea = run a regression to see if there is a relationship between the squared residuals + the fitted values Yˆi
* or possibly to run a regression using all original predictors instead of just Yˆi
* The underlying mechanics of this test aren’t the same as the ones for regressions;
* The goodness of fit is assessed using **a score-test**, NOT an F-test, + the test statistic is (approximately) χ2 distributed if there’s no relationship
* Using default settings, ncvTest() looks for a relationship between Yˆi + the variance of the residuals, making it a straightforward analogue of the above plot



* We see our original impression was right = there’s no violations of homoscedasticity in this data
* Quick sense of what you need to consider for how to deal w/ violations of homoscedasticity
* Main thing to worry about = if homoscedasticity is violated, the standard error estimates associated w/ the regression coefficients are no longer entirely reliable, + so t-tests for the coefficients aren’t quite right either.
* A simple fix = make use of a **heteroscedasticity-corrected covariance matrix** when estimating the standard errors.
* These are often called **sandwich estimators**, for reasons that only make sense if you understand the math at a low level
* In these estimators, the covariance matrix for **b** is given by:



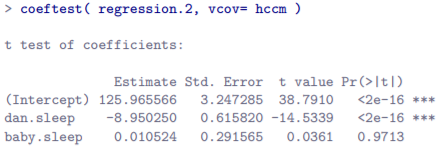
* See, it’s a “sandwich”, assuming you think:



* In any case, the usual estimator is what you get when you set:



* The original corrected version = 
* However, the version implemented as the default in **hccm**() is a tweak on this, proposed by Long and Ervin
* This version uses , where h(i) - the i-th hat value.
* You don’t need to understand what this means, but it might help to note hccm() in car() does it.
* Better yet, you can use **coeftest**() in **lmtest** package (but you need the car package loaded)



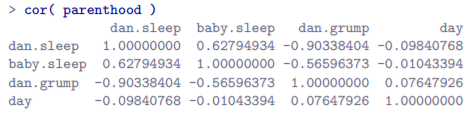
* Not surprisingly, these t tests are pretty much identical to the ones we saw w/ **summary(regression.2)** earlier
* Checking for collinearity
* Last regression diagnostic = the use of **variance inflation factors (VIFs) =** useful for determining whether or not predictors in a regression model = too highly correlated w/ each other
* There is a **VIF** associated w/ *each* predictor X(k) in the model, + the formula for the k-th VIF is:



* where R2(-k) = R-squared value you’d get if you ran a regression using X(k) as the outcome variable + all other X variables as the predictors.
* The idea here = R2(-k) is a *very good measure* of the extent to which X(k) is correlated w/ all other variables in a model.
* Better yet, the square root of VIF is pretty interpretable = tells you how much wider the CI for the corresponding coefficient **b**(k) is, relative to what you’d have expected if the predictors are all nice + uncorrelated w/ one another.
* W/ only 2 predictors, VIF values are always going to be the same



* Since the square root of 1.65 = 1.28, we see the correlation between our 2 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 another regression model to predict the day on which data were collected as a function of all the other variables in the data set.
* Look at the correlation matrix for all 4 variables:



* We have some fairly large correlations between some predictor variables





* When we run the regression model + look at the VIF values, we see the collinearity is causing a lot of uncertainty about the coefficients.

**15.10 Model selection**

* 1 fairly major problem that remains = **model selection**.
* For some data set w/ several variables, which do we include as predictors, + which should we not?
* In other words, we have a problem of **variable selection**.
* In general, model selection = a complex business, but made somewhat simpler if we restrict ourselves to the problem of choosing a *subset* of variables that ought to be included in the model.
* 2 broad principles:
* It’s nice to have an actual *substantive* basis for choices.
* In a lot of situation, a researcher has 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 their field.
* Never discount the importance of this.
* **Statistics *serves* the scientific process**, not the other way around.
* To the extent your choices rely on statistical inference, there is a trade-off between simplicity + goodness of fit.
* More predictors to a model = more complex
* Each predictor adds a new parameter (**regression coefficient**) + each new parameter *increases a model’s capacity to absorb random variations.*
* *Therefore, goodness of fit, R^2, continues to rise as you add more predictors no matter what.*
* For a model to be able to **generalize** well to *new* observations, avoid throwing in too many variables.
* **Occam’s razor** = do not multiply entities beyond necessity.
* In this context 🡪*Don’t chuck in largely irrelevant predictors just to boost R2*
* Need an actual mathematical criterion to implement the qualitative principle behind Occam’s razor in the context of selecting a regression model.
* There are several possibilities, such as the **Akaike information criterion (AIC)**, the default used in **step**()
* In the context of a linear regression model, the AIC for a model that has K predictor variables plus an intercept is:



* Note, step() computes the *full* version of AIC, including irrelevant constants that’ve been dropped above
* As a consequence, this equation won’t correctly describe the AIC values you see in the outputs.
* However, if you calculate AIC values using this formula for 2 different regression models + take the difference between them, this will be the same as the differences between AIC values 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 2 AIC values are useful, as these provide a measure of the extent to which one model outperforms another.
* Smaller AIC value = better model performance
* Ignoring low-level details, it’s fairly obvious what AIC does
* On the left is a term that *increases as the model predictions get worse* + on the right is a term that *increases as the model complexity increases*.
* **The best model** = one that fits the data well (low residuals = left hand side) using as few predictors as possible (low K = right hand side).
* **Backward elimination =** start w/ a complete regression model, including all possible predictors.
* Then, at each step, try all possible ways of removing 1 of the variables, + whichever of these is best (in terms of lowest AIC value) is accepted.
* This becomes our *new* regression model + then *try all possible deletions from the new model*, AGAIN choosing the option w/ lowest AIC.
* This continues until we end up w/ a model w/ a lower AIC value than any other possible models you could produce by deleting 1 of its predictors.
* 1st define the model from which the process starts

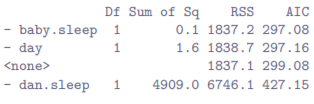


* The object argument to step() will be this regression model.

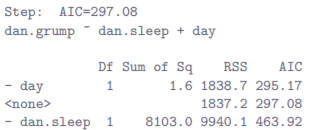




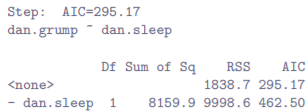
* *\*didn’t need to specify direction, as backward is the default.*
* 1st, the output reports the **AIC value** for the *current* best model, 299.08 = our starting point.
* Since small AIC values are good, we want to see if we can get a value smaller than 299.08 by deleting 1 of those 3 predictors.
* R tries all 3 possibilities, calculating AIC for each one, + prints out a short table w/ the results:



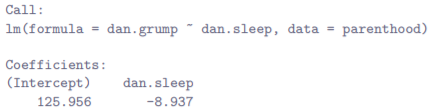
* Text on the is telling you what change R made to the regression model + <none> = the full model
* The other 3 rows correspond to the other 3 models it looked at 🡪 tried removing baby.sleep, which is indicated by -baby.sleep, + produced an AIC value = 297.08.
* That was the best of the 3 moves, so it’s at the top of the table.
* Now we start again w/ just 2 predictors left in the model + it tries deleting those



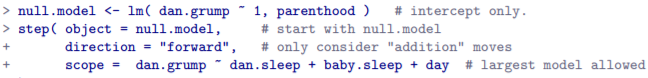
* See that removing day lowers AIC value to 295.17 so R keeps that change + moves on



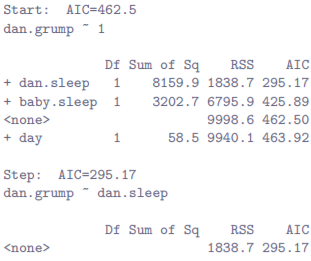
* No there’s no further deletions that can improve AIC so step() stops + prints out the result of the best regression model it could find

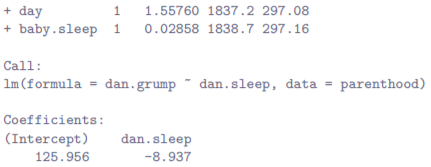


* This is (perhaps not surprisingly) 1st regression we started w/ at the beginning of the chapter.
* **Forward selection** = this time start w/ the *smallest* possible model as a start point + only consider possible additions to the model.
* 1 complication: tell step() the largest possible model you’re willing to entertain using **scope** arg.



* The output takes on a similar form, but now only considers addition (+) moves rather than deletion (-)

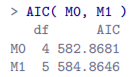




* As you can see, it’s found the same model.
* *Though, in general, forward + backward selection don’t always end up in the same place.*
* A caveat: Automated variable selection methods are seductive things, especially when bundled up in (fairly) simple functions like step()
* They provide an element of objectivity to 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 a model + what the theoretical basis for their inclusion might be as everything is solved by the magic of AIC
* There’s very little agreement on what counts as appropriate model selection criterion.
* 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 **BIC** for instance.
* Take a different approach again + get the **NML** criterion.
* Say you’re a Bayesian + you get model selection based on **posterior odds ratios**.
* All these different methods have strengths + weaknesses, + some are easier to calculate than others (AIC = probably easiest, 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.
* Show a bit of common sense: if staring at 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* + interpretability matters.
* An alternative to using automated model selection procedures is to explicitly select 2+ regression models to compare to each other.
* You can do this in a few different ways, depending on the research question you’re trying to answer.
* Ex: Want to know whether the amount of sleep a baby son got has any relationship to my grumpiness, over + above what we might expect from the amount of sleep that I got.
* Also want to make sure 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 variables** or **covariates** we want to control for.
* Here, we’d like to know whether **dan.grump ~ dan.sleep + day + baby.sleep** (Model 1, or M1) is a better regression model for these data than **dan.grump ~ dan.sleep + day** (Model 0, or M0).
* There are 2 different ways we can compare these 2 models: 1 based on a model selection criterion like AIC, + the other based on an explicit hypothesis test.
* AIC-based approach = simpler + follows naturally from step()
* 1st: Run the regressions



* Now could use summary() to run various hypothesis tests + other useful statistics
* However, since the current focus is on model comparison, go straight to AIC calculations.
* Conveniently, AIC() lets you input several regression models + it spits out AIC values for each



* Since Model 0 has the smaller AIC value, it is judged to be the better model for these data.
* There is also **BIC**() which computes the **Bayesian information criterion (BIC)** for models.
* Could type **BIC(M0,M1)** + get a very similar output.
* If using 1 of these 2, the empirical evidence suggests BIC = the better criterion of the 2.
* In most simulation studies, BIC does a much better job of selecting the correct model
* Hypothesis-testing framework approach
* Suppose you have 2 regression models, where 1 (Model 0) *contains a subset of the predictors from the other* (Model 1).
* i.e. Model 1 contains all predictors included in Model 0, plus 1+ additional predictors
* When this happens, we say *Model 0 is* ***nested w/in*** *Model 1*, or *Model 0 is a* ***submodel*** *of Model 1*.
* Regardless, what this means is to think of Model 0 as the null + Model 1 as the alternative
* Can construct an F-test for this in a fairly straightforward fashion 🡪 fit both models to the data + obtain a **residual sum of squares, SSres**, for both models



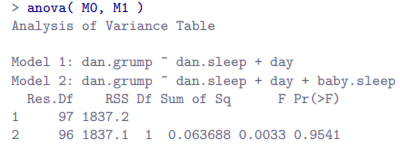
* where **N** = # of observations, **p** = # of predictors in the full model (not including intercept), + **k** = difference in # of parameters between the 2 models, + the **dF** = k + N - p - 1
* Worth noting: this same F statistic can be used to test a much broader range of hypotheses than those mentioned here.
* Notice the nested model M0 corresponds to the full model M1 when we constrain some regression coefficients to 0.
* Its sometimes useful to construct submodels by placing other kinds of constraints on the regression coefficients.
* For instance, maybe 2 different coefficients might have to sum to 0, or something like that.
* Can construct hypothesis tests for those kind of constraints as well, but it’s more complicated + the sampling distribution for F can end up being something known as the **non-central F distribution**, way beyond the scope of this book
* Note it’s often more convenient to think about the difference between those 2 SSres values as a sum of squares *in its own right.* That is:



* This his helpful b/c now we can express SS∆ as a measure of the extent to which the 2 models make different predictions about the outcome. Specifically:



* where ˆy(0)i = the fitted value for y(i) according to model M0 + ˆy(1)i = fitted value for y(i) according to model M1.
* That’s the hypothesis test we use to compare 2 regression models to one another.
* Can do it in R w/ anova() 🡪 input the 2 models we want to compare (null model first):



* R used the acronym **RSS** to refer to the **residual sum of squares** from each model.
* RSS in this output corresponds to **SSres** in the formula above.
* Since we have p > .05 we retain the null (M0).
* This approach to regression, adding all covariates into a null model + then add variables of interest into an alternative model + compare the 2 in hypothesis testing framework is often referred to as **hierarchical regression**