R for Data Science (IV)

Koji Mizumura

2018/08/03

Table of Contents

# IV Model

# Chapter 22. Introduction

Now that you are equippted with powerful programming tools, we can finally return to modeling. You will use your new tools of data wrangling and programming, to fit many models and understand how they work.

The focus of this book is on exploration, not confimation or formal inference. But you will leanr a few basic tools to help you understand the variation within your models.

The gola of a model is to provide a simple low-dimensional summary of a dataset. Ideally, the model will capture true “signals” (i.e. patterns generated by the phenomenon of interest), and ignore “noise” (i.e. random variation that you’re not interested in). Here we only cover “predictive” models, which, as the name suggests, generate predictions. There is another type of model that we’re not going to discuss: “data discovery” models. These models don’t make predictions, but instead help you discover interesting relationships within your data. (These two categories of models are sometimes called supervised and unsupervised, but I don’t think that terminology is particularly illuminating.)

This book is not going to give you a deep understanding of the mathematical theory that underlies models. It will, however, build your intuition about how statistical models work, and give you a family of useful tools that allow you to use models to better understand your data:

* In [model basics](http://r4ds.had.co.nz/model-basics.html), you’ll learn how models work mechanistically, focussing on the important family of linear models. You’ll learn general tools for gaining insight into what a predictive model tells you about your data, focussing on simple simulated datasets.
* In [model building](http://r4ds.had.co.nz/model-building.html), you’ll learn how to use models to pull out known patterns in real data. Once you have recognised an important pattern it’s useful to make it explicit in a model, because then you can more easily see the subtler signals that remain.
* In [many models](http://r4ds.had.co.nz/many-models.html), you’ll learn how to use many simple models to help understand complex datasets. This is a powerful technique, but to access it you’ll need to combine modeling and progamming tools.

These topics are notable because of what they don’t include: any tools for quantitatively assessing models. This is deliberate : precisely quantifying a model requires a couple of big ideas that we just don’t have the space to cover here. For now, you’ll rely on qualitative assessment and your natural speticism. In [Learning more about models](http://r4ds.had.co.nz/model-building.html#learning-more-about-models), we’ll point you to other resources where you can learn more.

## 22.1 Hypothesis generation vs. Hypothesis confirmation

In this book, we are going to use models as a tool for exploration, completing the trifecta of the tools for EDA that were introduced in Part 1. his is not how models are usually taught, but as you will see, models are an important tool for exploration. Traditionally, the focus of modelling is on inference, or for confirming that an hypothesis is true. Doing this correctly is not complicated, but it is hard. There is a pair of ideas that you must understand in order to do inference correctly:

1. Each observation can either be used for exploration or confimration, not both.
2. You can use an observation as many times as you like for exploration, but you can only use it once for confirmation. As soon as you use an observation twice, you’ve switched from confirmation to exploration.

This is necessary because to confirm a hypothesis you must use data independent of the data you used to generate the hypothesis. Otherwise you will be over optimistic. There is absolutely nothing wrong with exploration, but yo ushould neve sell an exploratory analysis as a confirmatory analysis because it is fundamentally misleading.

If you are serious about doing an confirmation analysis, one approach is to split your data into three pieces before you begin the analysis:

1. 60% of your data goes into a **training** (or exploration) set. You’re allowed to do anything you like with this data: visualise it and fit tons of models to it.
2. 20% goes into a **query** set. You can use this data to compare models or visualisations by hand, but you’re not allowed to use it as part of an automated process.
3. 20% is held back for a **test** set. You can only use this data ONCE, to test your final model.

# Chapter 23: Model basics

## 23.1 Introduction

The goal of a model is to provide a simple low-dimensional summary of a dataset. In the context of this book we’re going to use models to partition data into patterns and residuals. Strong patterns will hide subtler trends, so we’ll use models to help peel back layers of structure as we explore a dataset.

However, before we can start using models on interestin, real, datasets, you need to understand the basics of how model works. For that reason, this chapter of the book is unique because it uses only simulated datasets. These datasets are very simple, and not at all interesting, but they will help you understand the essence of modelling before you apply the same techniques to real data in the next chapter.

There are two parts to a model: 1.First, you define a **family of models** that express a precise, but generic, pattern that you want to capture. For example, the pattern might be a straight line, or a quadratic curve. You will express the model family as an equation like . Here, x and y are known variables from your data, and and are parameters that can vary to capture different patterns.

1. Next, you generate a **fitted model** by finding the model from the family that is the closest to your data. This takes the generic model family and makes it specific, like .

It’s important to understand that a fitter model is just the closest model from afamily of models. That implies that you have the “best” model (according to some criteria); it does not imply that you have a good model and it certainly doesn’t imply that the model is “true”. George Box puts this well in his famous aphrism:

All models are wrong, but some are useful

It’s worth reading the fuller context of the quote:

Now it would be very remarkable if any system existing in the real world could be exactly represented by any simple model. However, cunningly chosen parsimonious models often do provide remarkably useful approximations. For example, the law PV = RT relating pressure P, volume V and temperature T of an “ideal” gas via a constant R is not exactly true for any real gas, but it frequently provides a useful approximation and furthermore its structure is informative since it springs from a physical view of the behavior of gas molecules.

For such a model there is no need to ask the question “Is the model true?”. If “truth” is to be the “whole truth” the answer must be “No”. The only question of interest is “Is the model illuminating and useful?”.

The goal of a model is not to uncover truth, but to discovera simple approaximation that is still useful.

### 23.1.1. Pre-requisites

In this chapter, we will use the modelr package which wraps around base R’s modeling functions to make them work naturally in a pipe.

- echo: show/hide - eval: evaluate/not code - include: evaluate, but include/not include

library(tidyverse)  
library(modelr)  
options(na.action=na.warn)

## 23.2 A simpler model

Lets take a look at the simulated dataset sim1, included with the modelr package. It contains two continuous variables, x, y. Let’s plot them to see how they’re related:

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.5.1

library(modelr)  
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.5.1

## -- Attaching packages --------------------------- tidyverse 1.2.1 --

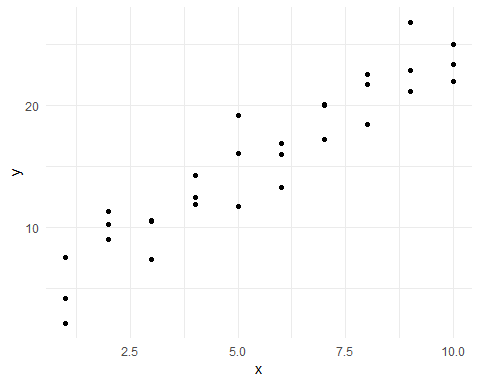
## √ tibble 1.4.2 √ purrr 0.2.5  
## √ tidyr 0.8.1 √ dplyr 0.7.6  
## √ readr 1.1.1 √ stringr 1.3.1  
## √ tibble 1.4.2 √ forcats 0.3.0

## Warning: package 'purrr' was built under R version 3.5.1

## Warning: package 'dplyr' was built under R version 3.5.1

## -- Conflicts ------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

ggplot(sim1,aes(x,y))+  
 geom\_point()+theme\_minimal()



You can see a strong pattern in the data. Let’s use a model to capture that pattern and make it explicit. It’s our job to supply the basic form of the model. In this case, the relationship looks linear, i.e., . Let’s start by getting a fell for what models from that family look like by randomly generating a few and overlaying them on the data.

For this simple case, we can use geom\_abline() which takes a slope and intercept as parameters. Later on we’ll learn more general techniques that work with any model.

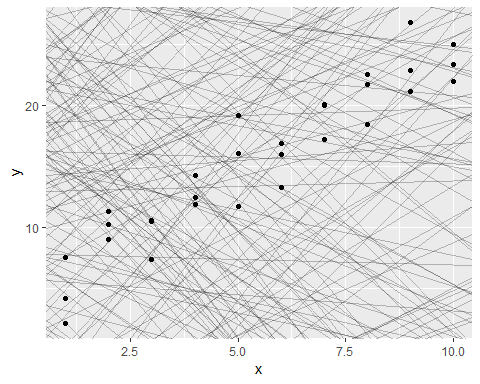
models <- tibble(  
 a1=runif(250,-20,40),  
 a2=runif(250,-5,5)  
)  
# runif(250,min=-20,max=40)  
  
modelsX <- tibble(  
 a1=runif(1,-20,40),  
 a2=runif(1,-5,5)  
) %>% print()

## # A tibble: 1 x 2  
## a1 a2  
## <dbl> <dbl>  
## 1 -10.7 -0.645

head(models)

## # A tibble: 6 x 2  
## a1 a2  
## <dbl> <dbl>  
## 1 5.73 1.98   
## 2 0.768 1.78   
## 3 39.1 4.12   
## 4 8.15 -1.63   
## 5 -11.3 0.0913  
## 6 10.9 0.960

ggplot(data=sim1,aes(x,y))+  
 geom\_abline(aes(intercept=a1,slope=a2),data=models,alpha=1/4)+  
 geom\_point()



There are 250 models on the above plot, but a lot are really bad! We need to find the good models by making precise our intuition that a good model is “close” to the data. We need a way to quantify the distance between the data and a model. Then we can fit the model by finding the value of a\_0 and a\_1 that generate the model with the smallest distance from this data.

One easy place to start is to find the vertical distance between each point and the model, as in the following diagram. (Note that I’ve shifted the x values slightly so you can see the individual distances.)

This distance is just the difference between the y value given by the model (the **prediction**), and the actual y value in the data (the **response**).

To compute this distance, we first turn our model family into an R function. This takes the model parameters and the data as inputs, and gives values predicted by the model as output:

model1 <- function(a,data){  
 a[1]+data$x\*a[2]  
}  
model1(c(7,1,5),sim1)

## [1] 8 8 8 9 9 9 10 10 10 11 11 11 12 12 12 13 13 13 14 14 14 15 15  
## [24] 15 16 16 16 17 17 17

Next, we need some way to compute an overall distance between the predicted and actual values. In other words, the plot above shows 30 distances: how do we collapse that into a single number?

One common way to do this in statistics to use the *“root-mean-squared deviation”*. We compute the difference between actual and predicted, square them, average them, and the take the square root. This distance has lots of appealing mathematical properties, which we’re not going to talk about here. You’ll just have to take my word for it!

measure\_distance <- function(mod,data){  
 diff <- data$y - model1(mod,data)  
 sqrt(mean(diff^2))  
}  
measure\_distance(c(7,1.5),sim1)

## [1] 2.665212

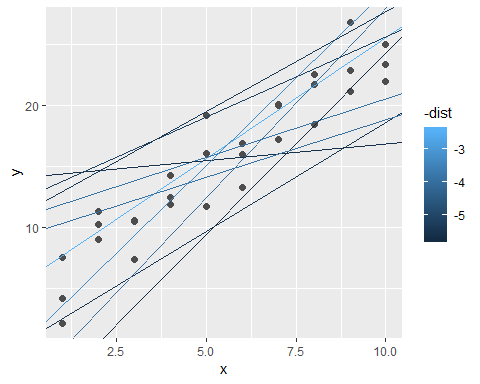
Now we can use purrr to compute the distance for all the models defined above. We need a helper function because our distance function expects the model as a numeric vector of length 2.

sim1\_dist <- function(a1,a2){  
 measure\_distance(c(a1,a2),sim1)  
}  
  
models <- models %>%   
 mutate(dist=purrr::map2\_dbl(a1,a2,sim1\_dist))  
models

## # A tibble: 250 x 3  
## a1 a2 dist  
## <dbl> <dbl> <dbl>  
## 1 5.73 1.98 2.41  
## 2 0.768 1.78 5.45  
## 3 39.1 4.12 46.7   
## 4 8.15 -1.63 19.6   
## 5 -11.3 0.0913 27.0   
## 6 10.9 0.960 3.85  
## 7 -19.5 3.57 16.1   
## 8 -12.0 1.68 18.4   
## 9 28.8 -1.93 11.9   
## 10 17.5 -2.72 19.0   
## # ... with 240 more rows

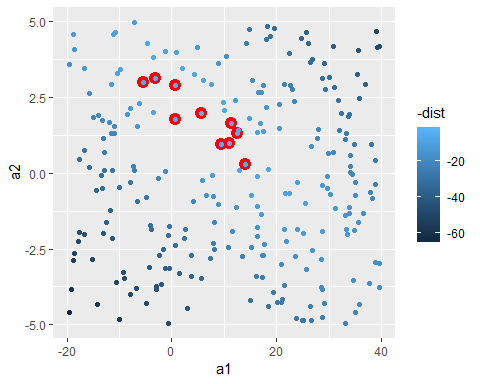
Next, let’s overlay the 10 best models on the data. I’ve coloured the models by -dist: this is an easy way to make sure that the best models (i.e., the ones with the smallest distance) get the highest colours.

ggplot(sim1,aes(x,y))+  
 geom\_point(size=2,colour="grey30")+  
 geom\_abline(  
 aes(intercept=a1,slope=a2,colour=-dist),  
 data=filter(models,rank(dist)<=10)  
 )



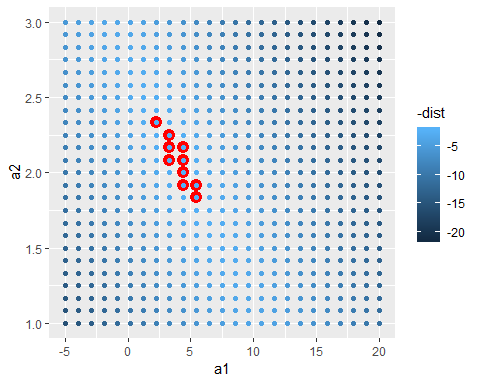
We can also think about these models as observations, and visualising with a scatterplot of a1 vs a2, again coloured by -dist. We can no longer directly see how the model compares to the data, but we can see many models at once. Again, I’ve highlighted the 10 best models, this time by drawing red circles underneath them.

ggplot(models,aes(a1,a2))+  
 geom\_point(data=filter(models,rank(dist)<=10),size=4,colour="red")+  
 geom\_point(aes(colour=-dist))



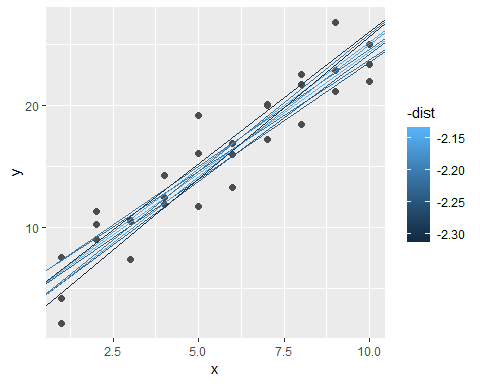
Instead of trying lots of random models, we could be more systematic and generate an evenly spaced grid of points (this is called a grid search). I picked the parameters of the grid roughly by looking at where the best models were in the plot above.

grid <- expand.grid(  
 a1=seq(-5,20,length=25),  
 a2=seq(1,3,length=25)  
) %>%   
 mutate(dist=purrr::map2\_dbl(a1,a2,sim1\_dist))  
  
grid %>%   
 ggplot(aes(a1,a2))+  
 geom\_point(data=filter(grid,rank(dist)<=10),size=4,colour="red")+  
 geom\_point(aes(colour=-dist))



When you overlay the best 10 models back on the original data, they all look pretty good:

ggplot(sim1,aes(x,y))+  
 geom\_point(size=2,colour="grey30")+  
 geom\_abline(  
 aes(intercept = a1, slope = a2, colour=-dist),  
 data=filter(grid,rank(dist)<=10)  
 )

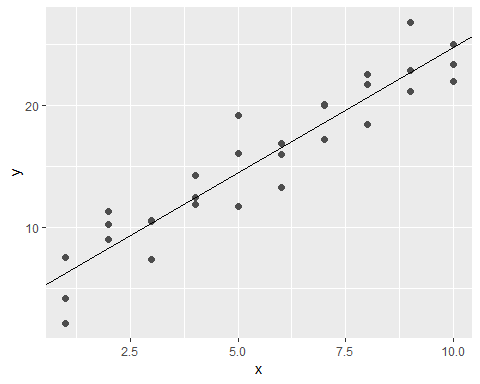


You could imagine iteratively making the grid finer and finer until you narrowed in on the best model. But there’s a better way to tackle that problem: a numerical minimisation tool called Newton-Raphson search. The intuition of Newton-Raphson is pretty simple: you pick a starting point and look around for the steepest slope. You then ski down that slope a little way, and then repeat again and again, until you can’t go any lower. In R, we can do that with optim():

best <- optim(c(0,0),measure\_distance,data=sim1)  
best$par

## [1] 4.222248 2.051204

#> [1] 4.22 2.05  
  
# ggplot(sim1,aes(x,y))+  
# geom\_point(size=2,colour="light grey")+  
# geom\_abline(intercept=c(3,2),slope=c(2,3),colour="blue")  
  
ggplot(sim1,aes(x,y))+  
 geom\_point(size=2,colour="grey30")+  
 geom\_abline(intercept=best$par[1],slope=best$par[2])



Don’t worry too much about the details of how optim() works. It’s the intuition that’s important here. If you have a function that defines the distance between a model and a dataset, an algorithm that can minimise that distance by modifying the parameters of the model, you can find the best model. The neat thing about this approach is that it will work for any family of models that you can write an equation for.

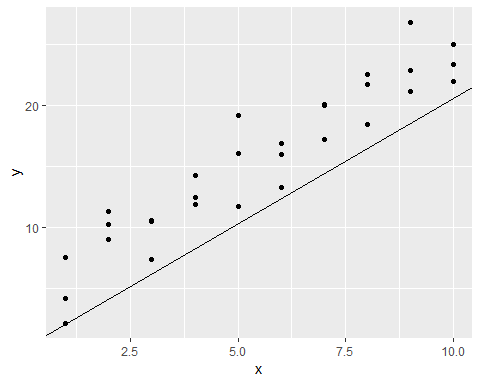
There’s one more approach that we can use for this model, because it’s a special case of a broader family: linear models. A linear model has the general form . So this simple model is equivalent to a general linear model where n is 2 and x\_1 is x. R has a tool specifically designed for fitting linear models called lm(). lm() has a special way to specify the model family: formulas. Formulas look like y ~ x, which lm() will translate to a function like y = a\_1 + a\_2 \* x. We can fit the model and look at the output:

sim1\_mod <- lm(y~x,data=sim1)  
coef(sim1\_mod)

## (Intercept) x   
## 4.220822 2.051533

ggplot(data=sim1,aes(x,y))+  
 geom\_point()+  
 geom\_abline(yintercept=coef(sim1\_mod)[1],slope=coef(sim1\_mod)[2])

## Warning: Ignoring unknown parameters: yintercept



These are exactly the same values we got with optim()! Behind the scenes lm() doesn’t use optim() but instead takes advantage of the mathematical structure of linear models. Using some connections between geometry, calculus, and linear algebra, lm() actually finds the closest model in a single step, using a sophisticated algorithm. This approach is both faster, and guarantees that there is a global minimum.

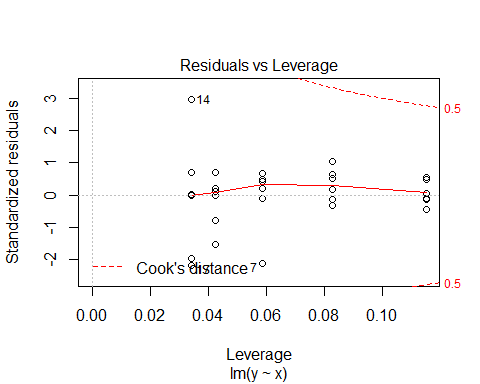
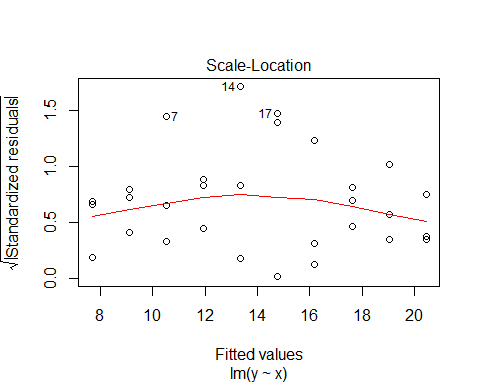
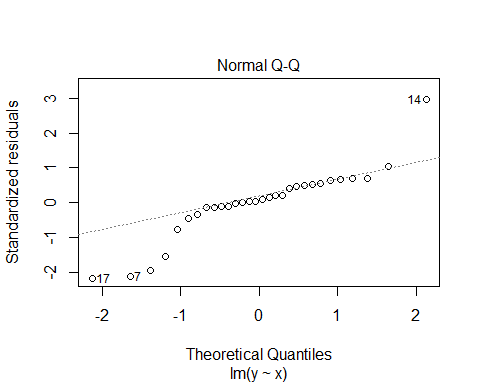
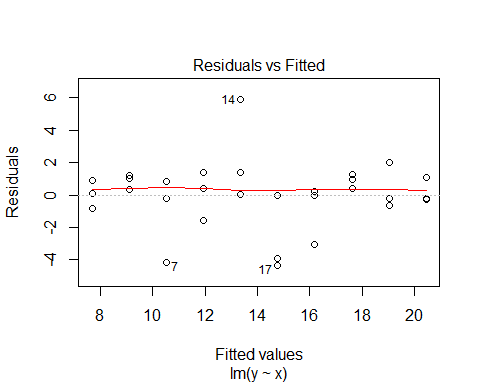
### 23.2.1 Exercises

1. One downside of the linear model is that it is sensitive to unusual values because the distance incorporates a squared term. Fit a linear model to the simulated data below, and visualise the results. Rerun a few times to generate different simulated datasets. What do you notice about the model?

sim1a <- tibble(  
 x = rep(1:10, each = 3),  
 y = x \* 1.5 + 6 + rt(length(x), df = 2)  
)  
sim1a\_mod <- lm(y~x,data=sim1a)  
coef(sim1a\_mod)

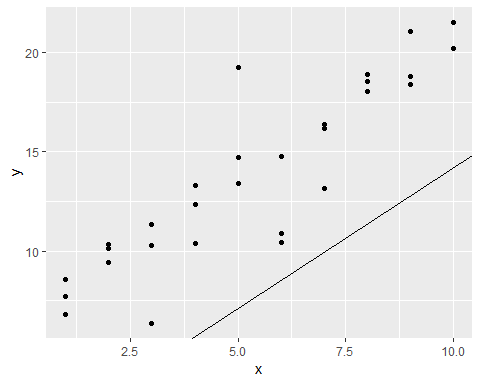
## (Intercept) x   
## 6.254940 1.419778

plot(sim1a\_mod)



ggplot(data=sim1a,aes(x,y))+  
 geom\_point()+  
 geom\_abline(yintercept=coef(sim1a\_mod)[1],slope=coef(sim1a\_mod)[2])

## Warning: Ignoring unknown parameters: yintercept



1. One way to make linear models more robust is to use a different distance measure. For example, instead of root-mean-squared distance, you could use mean-absolute distance:

measure\_distance <- function(mod, data) {  
 diff <- data$y - model1(mod, data)  
 mean(abs(diff))  
}

Use optim() to fit this model to the simulated data above and compare it to the linear model.

1. One challenge with performing numerical optimisation is that it’s only guaranteed to find one local optimum. What’s the problem with optimising a three parameter model like this?

model1 <- function(a, data) {  
 a[1] + data$x \* a[2] + a[3]  
}

## 23.3 Visualizing models

For simple models, ike the one above, you can figure out what pattern the model captures by carefully studying the model family and the fitted coefficients. And if you ever take a statistics course on modelling, you’re likely to spend a lot of time doing just that. Here, however, we’re going to take a different tack. We’re going to focus on understanding a model by looking at its predictions. This has a big advantage: every type of predictive model makes predictions (otherwise what use would it be?) so we can use the same set of techniques to understand any type of predictive model.

It’s also useful to see what the model doesn’t capture, the so-called residuals which are left after subtracting the predictions from the data. Residuals are powerful because they allow us to use models to remove striking patterns so we can study the subtler trends that remain.

### 23.3.1 Predictions

To visualise the predictions from a model, we start by generating an evenly spaced grid of values that covers the region where our data lies. The easiest way to do that is to use modelr::data\_grid(). Its first argument is a data frame, and for each subsequent argument it finds the unique variables and then generates all combinations:

grid <- sim1 %>%   
 data\_grid(x)  
grid

## # A tibble: 10 x 1  
## x  
## <int>  
## 1 1  
## 2 2  
## 3 3  
## 4 4  
## 5 5  
## 6 6  
## 7 7  
## 8 8  
## 9 9  
## 10 10

(This will get more interesting when we start to add more variables to our model.)

Next we add predictions. We’ll use modelr::add\_predictions() which takes a data frame and a model. It adds the predictions from the model to a new column in the data frame:

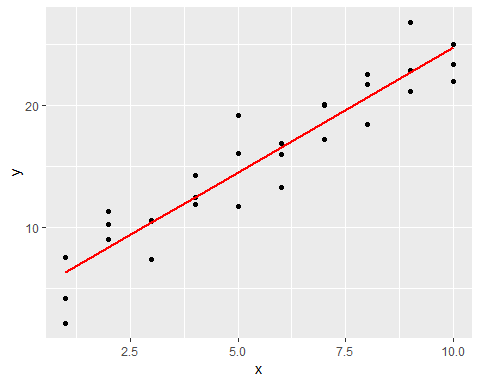
grid <- grid %>%   
 add\_predictions(sim1\_mod)   
grid

## # A tibble: 10 x 2  
## x pred  
## <int> <dbl>  
## 1 1 6.27  
## 2 2 8.32  
## 3 3 10.4   
## 4 4 12.4   
## 5 5 14.5   
## 6 6 16.5   
## 7 7 18.6   
## 8 8 20.6   
## 9 9 22.7   
## 10 10 24.7

You can also use this function to add predictions to your original dataset.)

Next, we plot the predictions. You might wonder about all this extra work compared to just using geom\_abline(). But the advantage of this approach is that it will work with any model in R, from the simplest to the most complex. You’re only limited by your visualisation skills. For more ideas about how to visualise more complex model types, you might try <http://vita.had.co.nz/papers/model-vis.html>.

ggplot(sim1, aes(x))+  
 geom\_point(aes(y=y))+  
 geom\_line(aes(y=pred),data=grid,colour="red",size=1)



### 23.3.2 Residuals

The flip-side of predictions are residuals. The predictions tells you the pattern that the model has captured, and the residuals tell you what the model has missed. The residuals are just the distances between the observed and predicted values that we computed above.

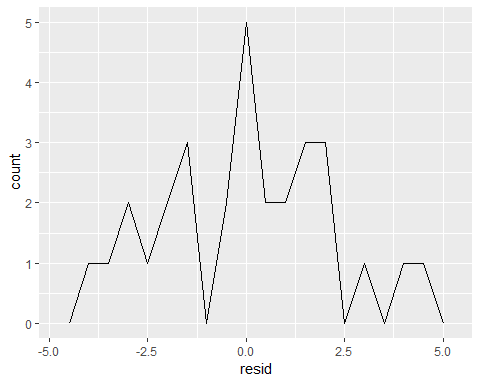
We add residuals to the data with add\_residuals(), which works much likeadd\_predictions(). Note, however, that we use the original dataset, not a manufactured grid. This is because to compute residuals we need actual y values.

sim1 <- sim1 %>%   
 add\_residuals(sim1\_mod)  
sim1

## # A tibble: 30 x 3  
## x y resid  
## <int> <dbl> <dbl>  
## 1 1 4.20 -2.07   
## 2 1 7.51 1.24   
## 3 1 2.13 -4.15   
## 4 2 8.99 0.665   
## 5 2 10.2 1.92   
## 6 2 11.3 2.97   
## 7 3 7.36 -3.02   
## 8 3 10.5 0.130   
## 9 3 10.5 0.136   
## 10 4 12.4 0.00763  
## # ... with 20 more rows

There are a few different ways to understand what the residuals tell us about the model. One way is to simply draw a frequency polygon to help us understand the spread of the residuals:

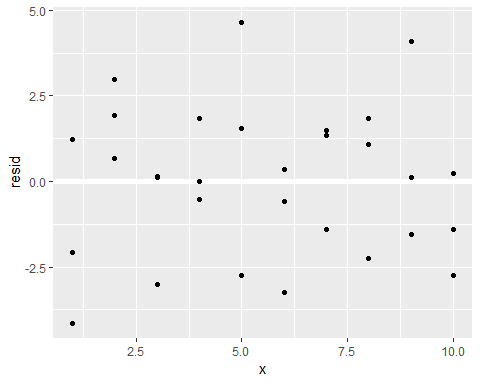
ggplot(sim1,aes(resid))+  
 geom\_freqpoly(binwidth=0.5)



This helps you calibrate the quality of the model: how far away are the predictions from the observed values? Notee average of the residual will always be 0.

You’ll often want to recreate plots using the residuals instead of the original predictor. You’ll see a lot of that in the next chapter.

library(ggplot2)  
library(tidyverse)  
library(modelr)  
  
ggplot(sim1,aes(x,resid))+  
 geom\_ref\_line(h=0)+  
 geom\_point()

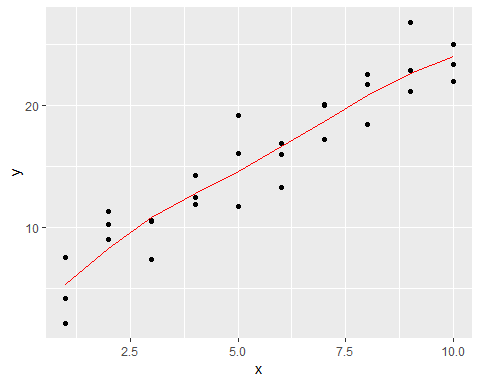


This looks like random noise, suggesting that our model has done a good job of capturing the patterns in the dataset.

### 23.3.3 Exercises

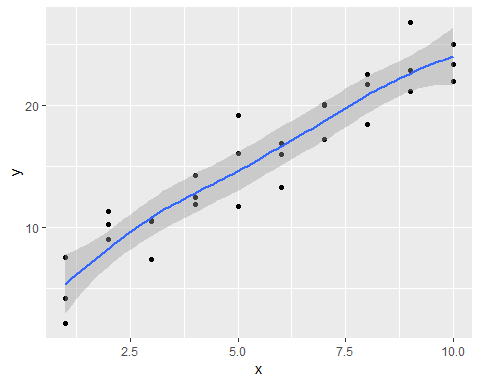
1. Instead of using lm() to fit a straight line, you can use loess() to fit a smooth curve. Repeat the process of model fitting, grid generation, predictions, and visualisation on sim1 using loess() instead of lm(). How does the result compare to geom\_smooth()?

mod1 <- loess(y~x,data=sim1)  
sim1 %>%   
 add\_predictions(mod1) %>%   
 ggplot(aes(x,y))+  
 geom\_point()+  
 geom\_line(aes(y=pred),colour="red")



sim1 %>%   
 ggplot(aes(x,y))+  
 geom\_point()+  
 geom\_smooth()

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'

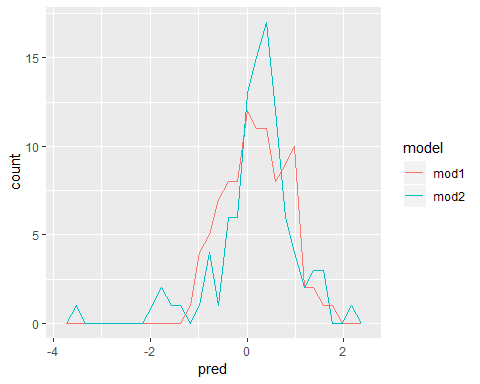


1. add\_predictions() is paired with gather\_predictions() and spread\_predictions(). How do these three functions differ?

gather\_predictions works for gathering serveral models into a tidy data.

new\_data <- tibble(  
 y=rnorm(100),  
 x=y+rnorm(100,mean=5),  
 z=y\*runif(100,max=100)  
)  
  
mod1 <- lm(y~x,data=new\_data)  
mod2 <- lm(y~z,data=new\_data)  
  
final\_data <-   
 new\_data %>%   
 gather\_predictions(mod1,mod2)  
  
final\_data %>%   
 ggplot(aes(pred, colour = model)) +  
 geom\_freqpoly()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



spread\_predictions does the same but adds the predictions as columns rather than as tidy dataset.

new\_data %>%   
 spread\_predictions(mod1,mod2)

## # A tibble: 100 x 5  
## y x z mod1 mod2  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1.07 6.50 19.2 0.754 0.377   
## 2 0.101 4.35 2.85 -0.206 0.138   
## 3 0.808 4.66 12.1 -0.0666 0.274   
## 4 0.283 6.81 6.28 0.893 0.189   
## 5 1.50 5.73 89.3 0.413 1.40   
## 6 -0.516 4.91 -35.9 0.0445 -0.427   
## 7 -0.235 3.52 -6.58 -0.578 0.000935  
## 8 -0.970 4.62 -60.2 -0.0856 -0.782   
## 9 1.55 4.63 99.6 -0.0816 1.55   
## 10 0.521 6.84 37.9 0.908 0.650   
## # ... with 90 more rows

1. What does geom\_ref\_line() do? What package does it come from? Why is displaying a reference line in plots showing residuals useful and important?

geom\_ref\_line is a nice addition to ggplot2 although it comes from modelr. It’s purpose is just adding a reference line in a plot. It’s very practical for analyzing residuals because that way you can figure out if many points are above/below a certain point, and whether the models is worse/better at being overly pessimistic or overly positive.

Why might you want to look at a frequency polygon of absolute residuals? What are the pros and cons compared to looking at the raw residuals?

Looking at absolute residuals would work really well to distinguish the magnitude of bad or good predictions. Moreover, it server well to identiy strong outliers. On the other hand, the con side is that you don’t know whether that strong prediction is either positive or negative. That’s why it’s better to look at raw residuals for that different question.

## 23.4 Formulas and model families

You have seen formulas before when using facet\_wrap() and facet\_grid(). In R, formulas provide a general way of getting “special behaviour”. Rather than evaluating the values of the variables right away, they capture them so they can be interpreted by the function.

The majority of modelling functions in R use a standard conversion from formulas to functions. You’ve seen one simple conversion already: is translated to . If you want to see what R actually does, you can use the model\_matrix() function. It takes a data frame and a formula and returns a tibble that defines the model equation: each column in the output is associated with one coefficient in the model, the function is always . For the simplest case of y ~ x1 this shows us something interesting:

df <- tribble(  
 ~y,~x1,~x2,  
 4,2,5,  
 5,1,6  
)  
  
model\_matrix(df,y~x1)

## # A tibble: 2 x 2  
## `(Intercept)` x1  
## <dbl> <dbl>  
## 1 1 2  
## 2 1 1

The way that R adds the intercept to the model is just by having a column that is full of ones. By default, R will always add this column. If you don’t want, you need to explicitly drop it with -1:

model\_matrix(df,y~x1-1)

## # A tibble: 2 x 1  
## x1  
## <dbl>  
## 1 2  
## 2 1

The model matrix grows in an unsurprising way when you add more variables to the the model:

model\_matrix(df,y~x1+x2)

## # A tibble: 2 x 3  
## `(Intercept)` x1 x2  
## <dbl> <dbl> <dbl>  
## 1 1 2 5  
## 2 1 1 6

This formula notation is sometimes called “Wilkinson-Rogers notation” and was initially described in Symbolic Description of Factorial Models for Analysis of Variance, by G. N. Wilkinson and C. E. Rogers <https://www.jstor.org/stable/2346786>. It’s worth digging up and reading the original paper if you’d like to understand the full details of the modelling algebra.

The following sections expand on how this formula notation works for categorical variables, interactions, and transformation.

### 23.4.1 Categorical variables

Generating a function from a formula is straight forward when the predictor is continuous, but things get a bit more complicated when the predictor is categorical. Imagine you have a formula like y ~ sex, where sex could either be male or female. It doesn’t make sense to convert that to a formula like because sex isn’t a number - you can’t multiply it! Instead what R does is convert it to where sex\_male is one if sex is male and zero otherwise:

df <- tribble(  
 ~sex,~response,  
 "male",1,  
 "female",2,  
 "male",1  
)  
  
model\_matrix(df,response~sex)

## # A tibble: 3 x 2  
## `(Intercept)` sexmale  
## <dbl> <dbl>  
## 1 1 1  
## 2 1 0  
## 3 1 1

You might wonder why R also doesn’t create a sexfemale column. The problem is that would create a column that is perfectly predictable based on the other columns (i.e. sexfemale = 1 - sexmale). Unfortunately the exact details of why this is a problem is beyond the scope of this book, but basically it creates a model family that is too flexible, and will have infinitely many models that are equally close to the data.

Fortunately, however, if you focus on visualizing predictions you don’t need to worry about the exact parameterisation. Let’s look at some data and models to make that concrete. Here’s the sim2 dataset from modelr:

library(ggplot2)  
library(modelr)  
  
ggplot(sim2)+  
 geom\_point(aes(x,y))



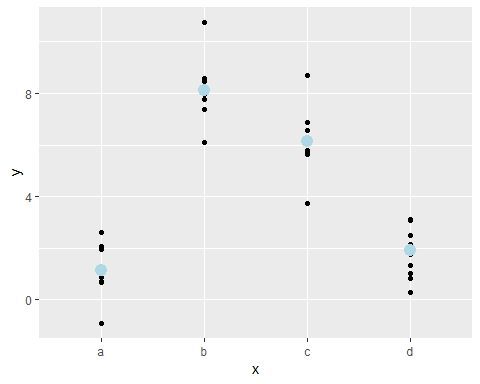
We can fit a model to it, and generate predictions:

mod2 <- lm(y~x,data=sim2)  
  
grid <- sim2 %>%   
 data\_grid(x) %>%   
 add\_predictions(mod2)  
grid

## # A tibble: 4 x 2  
## x pred  
## <chr> <dbl>  
## 1 a 1.15  
## 2 b 8.12  
## 3 c 6.13  
## 4 d 1.91

Effectively, a model with a categorical x will predict the mean value for each category. (Why? Because the mean minimises the root-mean-squared distance.) That’s easy to see if we overlay the predictions on top of the original data:

ggplot(sim2, aes(x))+  
 geom\_point(aes(y=y))+  
 geom\_point(data=grid,aes(y=pred),colour="light blue",size=4)



You can’t make predictions about levels that you didn’t observe. Sometimes you’ll do this by accident so it’s good to recognise this error message:

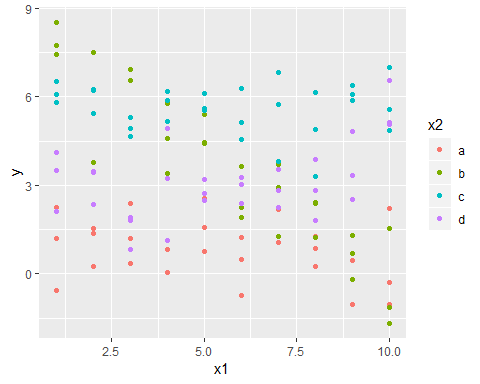
### 23.4.2 Interactions (Continuous and categorical)

What happens when you combine a continuous and categorical variable? sim3 contains a categorical peredictor and continuous predictor. We can visualize it with a simple plot.

head(sim3)

## # A tibble: 6 x 5  
## x1 x2 rep y sd  
## <int> <fct> <int> <dbl> <dbl>  
## 1 1 a 1 -0.571 2  
## 2 1 a 2 1.18 2  
## 3 1 a 3 2.24 2  
## 4 1 b 1 7.44 2  
## 5 1 b 2 8.52 2  
## 6 1 b 3 7.72 2

ggplot(sim3,aes(x1,y))+  
 geom\_point(aes(colour=x2))



There are two possible models you could fit to this data:

mod1 <- lm(y~x1+x2,data=sim3)  
mod2 <- lm(y~x1\*x2,data=sim3)

When you add variables with +, the model will estimate each effect independent of all the others. It’s possible to fit the so-called interaction by using \*. For example, is translated to . Note that whenever you use \*, both the interaction and the individual components are included in the model.

To visalize these models, we need two new tricks: 1. We have two predictors, so we need to give data\_grid() both variables. It finds all the unique values of x1 and x2 and then generates all combinations.

To generate predictions from both models simultaneously, we can use gather\_predictions() which adds each prediction as a row. The complement of gather\_predictions() is spread\_predictions() which adds each prediction to a new column.

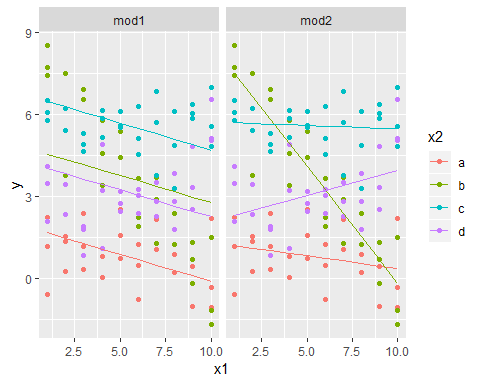
Together this gives us:

grid <- sim3 %>%   
 data\_grid(x1,x2) %>%   
 gather\_predictions(mod1,mod2)  
grid

## # A tibble: 80 x 4  
## model x1 x2 pred  
## <chr> <int> <fct> <dbl>  
## 1 mod1 1 a 1.67  
## 2 mod1 1 b 4.56  
## 3 mod1 1 c 6.48  
## 4 mod1 1 d 4.03  
## 5 mod1 2 a 1.48  
## 6 mod1 2 b 4.37  
## 7 mod1 2 c 6.28  
## 8 mod1 2 d 3.84  
## 9 mod1 3 a 1.28  
## 10 mod1 3 b 4.17  
## # ... with 70 more rows

We can visualize the results for both models on one plot using facetting:

ggplot(sim3, aes(x1,y,colour=x2))+  
 geom\_point()+  
 geom\_line(data=grid,aes(y=pred))+  
 facet\_wrap(~model)



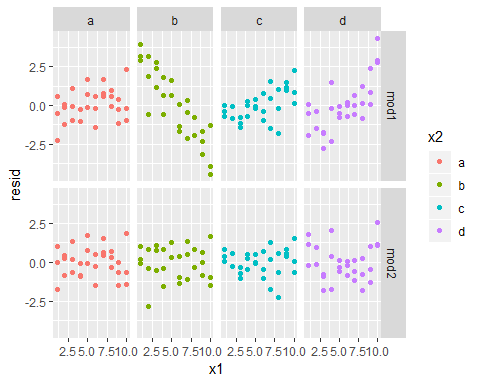
Note that the model that uses + has the same slope for each line, but different intercepts. The model that uses \* has a different slope and intercept for each line.

Which model is better for this data? We can take look at the residuals. Here I’ve facetted by both model and x2 because it makes it easier to see the pattern within each group.

sim3 <- sim3 %>%   
 gather\_residuals(mod1, mod2)  
sim3

## # A tibble: 240 x 7  
## model x1 x2 rep y sd resid  
## <chr> <int> <fct> <int> <dbl> <dbl> <dbl>  
## 1 mod1 1 a 1 -0.571 2 -2.25   
## 2 mod1 1 a 2 1.18 2 -0.491   
## 3 mod1 1 a 3 2.24 2 0.562   
## 4 mod1 1 b 1 7.44 2 2.87   
## 5 mod1 1 b 2 8.52 2 3.96   
## 6 mod1 1 b 3 7.72 2 3.16   
## 7 mod1 1 c 1 6.51 2 0.0261  
## 8 mod1 1 c 2 5.79 2 -0.691   
## 9 mod1 1 c 3 6.07 2 -0.408   
## 10 mod1 1 d 1 2.11 2 -1.92   
## # ... with 230 more rows

ggplot(sim3, aes(x1, resid, colour = x2)) +   
 geom\_point() +   
 facet\_grid(model ~ x2)



There is little obvious pattern in the residuals for mod2. The residuals for mod1 show that the model has clearly missed some pattern in b, and less so, but still present is pattern in c, and d. You might wonder if there’s a precise way to tell which of mod1 or mod2 is better. There is, but it requires a lot of mathematical background, and we don’t really care. Here, we’re interested in a qualitative assessment of whether or not the model has captured the pattern that we’re interested in.

### 23.4.3 Interactions (two continuous)

Let’s take a look at the equvalent model for two continuous variables. Initially things proceed almost identically to the previous example:

mod1 <- lm(y~x1+x2,data=sim4)  
mod2 <- lm(y~x1\*x2,data=sim4)  
  
grid <- sim4 %>%   
 data\_grid(  
 x1=seq\_range(x1,5),  
 x2=seq\_range(x2,5)  
 ) %>%   
 gather\_predictions(mod1,mod2)  
grid

## # A tibble: 50 x 4  
## model x1 x2 pred  
## <chr> <dbl> <dbl> <dbl>  
## 1 mod1 -1 -1 0.996  
## 2 mod1 -1 -0.5 -0.395  
## 3 mod1 -1 0 -1.79   
## 4 mod1 -1 0.5 -3.18   
## 5 mod1 -1 1 -4.57   
## 6 mod1 -0.5 -1 1.91   
## 7 mod1 -0.5 -0.5 0.516  
## 8 mod1 -0.5 0 -0.875  
## 9 mod1 -0.5 0.5 -2.27   
## 10 mod1 -0.5 1 -3.66   
## # ... with 40 more rows

Note my use of seq\_range() inside data\_grid(). Instead of using every unique value of x, I’m going to use a regularly spaced grid of five values between the minimum and maximum numbers. It’s probably not super important here, but it’s a useful technique in general. There are two other useful arguments to seq\_range()]

* pretty = TRUE will generate a “pretty” sequence, i.e. something that looks nice to the human eye. This is useful if you want to produce tables of output:

seq\_range(c(0.0123,0.923423),n=5)

## [1] 0.0123000 0.2400808 0.4678615 0.6956423 0.9234230

seq\_range(c(0.0123,0.923423),n=5,pretty=T)

## [1] 0.0 0.2 0.4 0.6 0.8 1.0

* trim = 0.1 will trim off 10% of the tail values. This is useful if the variables have a long tailed distribution and you want to focus on generating values near the center:

x1 <- rcauchy(100)  
seq\_range(x1, n = 5)

## [1] -20.08731 44.28287 108.65304 173.02322 237.39340

#> [1] -115.9 -83.5 -51.2 -18.8 13.5  
seq\_range(x1, n = 5, trim = 0.10)

## [1] -3.9007857 -2.0507260 -0.2006664 1.6493933 3.4994530

#> [1] -13.84 -8.71 -3.58 1.55 6.68  
seq\_range(x1, n = 5, trim = 0.25)

## [1] -1.4784673 -0.6168007 0.2448659 1.1065324 1.9681990

#> [1] -2.1735 -1.0594 0.0547 1.1687 2.2828  
seq\_range(x1, n = 5, trim = 0.50)

## [1] -0.98228883 -0.52716399 -0.07203916 0.38308568 0.83821051

#> [1] -0.725 -0.268 0.189 0.647 1.104

* expand=0.1 is in some sense the opposite of trim() it expands the range by 10%.

x2 <- c(0, 1)  
seq\_range(x2, n = 5)

## [1] 0.00 0.25 0.50 0.75 1.00

#> [1] 0.00 0.25 0.50 0.75 1.00  
seq\_range(x2, n = 5, expand = 0.10)

## [1] -0.050 0.225 0.500 0.775 1.050

#> [1] -0.050 0.225 0.500 0.775 1.050  
seq\_range(x2, n = 5, expand = 0.25)

## [1] -0.1250 0.1875 0.5000 0.8125 1.1250

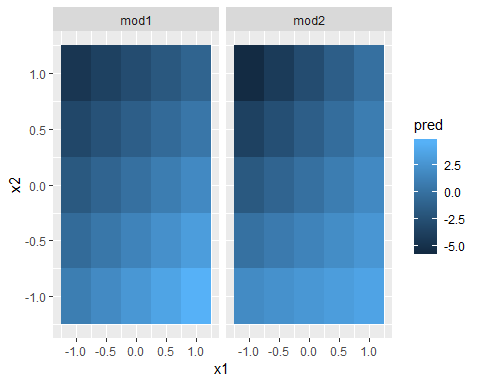
#> [1] -0.125 0.188 0.500 0.812 1.125  
seq\_range(x2, n = 5, expand = 0.50)

## [1] -0.250 0.125 0.500 0.875 1.250

#> [1] -0.250 0.125 0.500 0.875 1.250

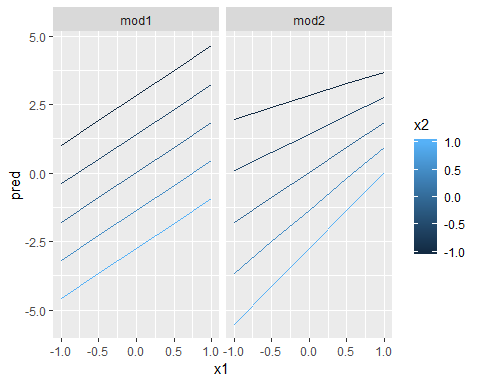
Next let’s try and visualize that moel. We have two continuous predictors, so you can imagine the model like a 3d surface. We could display that using geom\_tile():

ggplot(grid, aes(x1, x2)) +   
 geom\_tile(aes(fill = pred)) +   
 facet\_wrap(~ model)

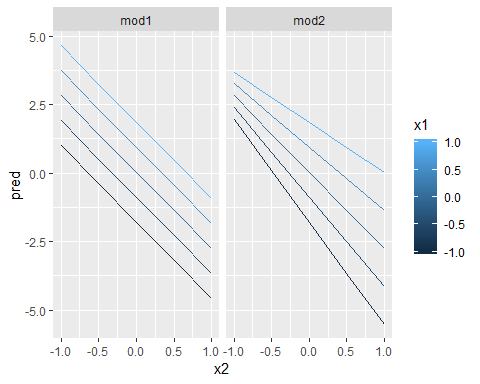


That doesn’t suggest that the models are very different! But that’s partly an illusion: our eyes and brains are not very good at accurately comparing shades of colour. Instead of looking at the surface from the top, we could look at it from either side, showing multiple slices:

ggplot(grid, aes(x1, pred, colour = x2, group = x2)) +   
 geom\_line() +  
 facet\_wrap(~ model)



ggplot(grid, aes(x2, pred, colour = x1, group = x1)) +   
 geom\_line() +  
 facet\_wrap(~ model)



This shows you that interaction between two continuous variables works basically the same way as for a categorical and continuous variable. An interaction says that there’s not a fixed offset: you need to consider both values of x1 and x2 simultaneously in order to predict y.

You can see that even with just two continuous variables, coming up with good visualisations are hard. But that’s reasonable: you shouldn’t expect it will be easy to understand how three or more variables simultaneously interact! But again, we’re saved a little because we’re using models for exploration, and you can gradually build up your model over time. The model doesn’t have to be perfect, it just has to help you reveal a little more about your data.

I spent some time looking at the residuals to see if I could figure if mod2 did better than mod1. I think it does, but it’s pretty subtle. You’ll have a chance to work on it in the exercises.

### 23.4.4 Transformations

You can also perform transformations inside the model formula. For example, is transformed to . If your transformation involves +, \*, ^, or -, you’ll need to wrap it in I() so R doesn’t treat it like part of the model specification. For example, is translated to . If you forget the I() and specify , R will compute . means the interaction of x with itself, which is the same as x. R automatically drops redundant variables so x + x become x, meaning that specifies the function . That’s probably not what you intended!

Again, if you get confused about what your model is doing, you can always use model\_matrix() to see exactly what equation lm() is fitting:

library(tidyverse)  
library(modelr)  
  
df <- tribble(  
 ~y, ~x,  
 1, 1,  
 2, 2,  
 3, 3  
)  
  
model\_matrix(df,y~x^2+x)

## # A tibble: 3 x 2  
## `(Intercept)` x  
## <dbl> <dbl>  
## 1 1 1  
## 2 1 2  
## 3 1 3

model\_matrix(df,y~I(x^2)+x)

## # A tibble: 3 x 3  
## `(Intercept)` `I(x^2)` x  
## <dbl> <dbl> <dbl>  
## 1 1 1 1  
## 2 1 4 2  
## 3 1 9 3

Transformations are useful because you can use them to approximate non-linear functions. If you’ve taken a calculus class, you may have heard of Taylor’s theorem which says you can approximate any smooth function with an infinite sum of polynomials. That means you can use a polynomial function to get arbitrarily close to a smooth function by fitting an equation like . Typing that sequence by hand is tedious, so R provides a helper function: poly():

model\_matrix(df,y~poly(x,2))

## # A tibble: 3 x 3  
## `(Intercept)` `poly(x, 2)1` `poly(x, 2)2`  
## <dbl> <dbl> <dbl>  
## 1 1 -7.07e- 1 0.408  
## 2 1 -7.85e-17 -0.816  
## 3 1 7.07e- 1 0.408

However there’s one major problem with using poly(): outside the range of the data, polynomials rapidly shoot off to positive or negative infinity. One safer alternative is to use the natural spline, splines::ns().

library(splines)  
head(df)

## # A tibble: 3 x 2  
## y x  
## <dbl> <dbl>  
## 1 1 1  
## 2 2 2  
## 3 3 3

model\_matrix(df,y~ns(x,2))

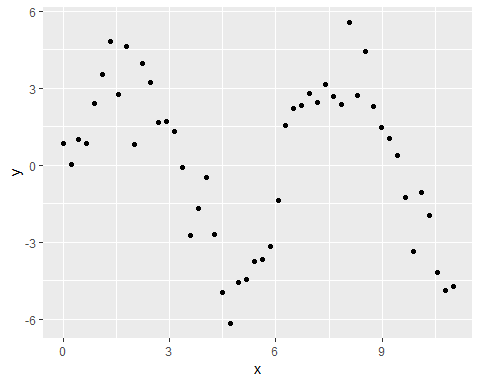
## # A tibble: 3 x 3  
## `(Intercept)` `ns(x, 2)1` `ns(x, 2)2`  
## <dbl> <dbl> <dbl>  
## 1 1 0 0   
## 2 1 0.566 -0.211  
## 3 1 0.344 0.771

Let’s see what that looks like when we try and approximate a non-linear function:

sim5 <- tibble(  
 x=seq(0,3.5\*pi,length=50),  
 y=4\*sin(x)+rnorm(length(x))  
)  
  
sim5

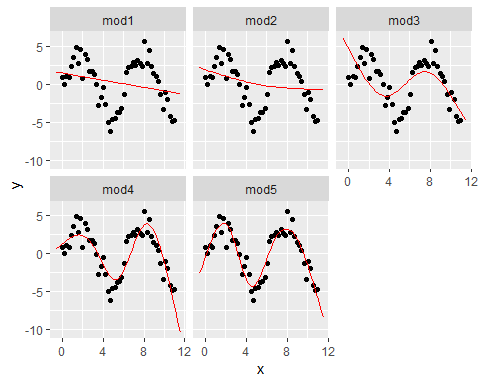
## # A tibble: 50 x 2  
## x y  
## <dbl> <dbl>  
## 1 0 0.846   
## 2 0.224 0.0396  
## 3 0.449 1.02   
## 4 0.673 0.850   
## 5 0.898 2.40   
## 6 1.12 3.55   
## 7 1.35 4.82   
## 8 1.57 2.78   
## 9 1.80 4.62   
## 10 2.02 0.831   
## # ... with 40 more rows

# seq(stats::rnorm(20)) # effectively 'along'  
# seq(1, 9, by = 2) # matches 'end'  
# seq(1, 9, by = pi) # stays below 'end'  
# seq(1, 6, by = 3)  
# seq(1.575, 5.125, by = 0.05)  
# seq(17) # same as 1:17, or even better seq\_len(17)  
  
ggplot(sim5,aes(x,y))+  
 geom\_point()



I’m going to fit five models to this data:

mod1 <- lm(y~ns(x,1),data=sim5)  
mod2 <- lm(y~ns(x,2),data=sim5)  
mod3 <- lm(y~ns(x,3),data=sim5)  
mod4 <- lm(y~ns(x,4),data=sim5)  
mod5 <- lm(y~ns(x,5),data=sim5)  
  
grid <- sim5 %>%   
 data\_grid(x=seq\_range(x,n=50,expand=0.1)) %>%   
 gather\_predictions(mod1,mod2,mod3,mod4,mod5,.pred="y")  
  
ggplot(sim5, aes(x, y)) +   
 geom\_point() +  
 geom\_line(data = grid, colour = "red") +  
 facet\_wrap(~ model)



Notice that the extrapolation outside the range of the data is clearly bad. This is the downside to approximating a function with a polynomial. But this is a very real problem with every model: the model can never tell you if the behaviour is true when you start extrapolating outside the range of the data that you have seen. You must rely on theory and science

### 23.4.5 Exercises

1. What happens if you repeat the analysis of sim2 using a model without an intercept. What happens to the model equation? What happens to the predictions?

head(sim2)

## # A tibble: 6 x 2  
## x y  
## <chr> <dbl>  
## 1 a 1.94   
## 2 a 1.18   
## 3 a 1.24   
## 4 a 2.62   
## 5 a 1.11   
## 6 a 0.866

mod1 <- lm(y~x,data=sim2)  
mod2 <- lm(y~x+-1,data=sim2)  
  
model\_matrix(sim2,y~x)

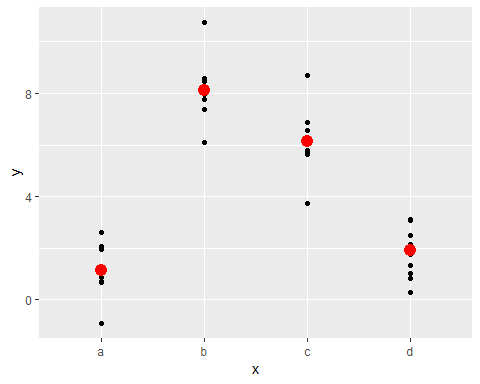
## # A tibble: 40 x 4  
## `(Intercept)` xb xc xd  
## <dbl> <dbl> <dbl> <dbl>  
## 1 1 0 0 0  
## 2 1 0 0 0  
## 3 1 0 0 0  
## 4 1 0 0 0  
## 5 1 0 0 0  
## 6 1 0 0 0  
## 7 1 0 0 0  
## 8 1 0 0 0  
## 9 1 0 0 0  
## 10 1 0 0 0  
## # ... with 30 more rows

model\_matrix(sim2,y~x+-1)

## # A tibble: 40 x 4  
## xa xb xc xd  
## <dbl> <dbl> <dbl> <dbl>  
## 1 1 0 0 0  
## 2 1 0 0 0  
## 3 1 0 0 0  
## 4 1 0 0 0  
## 5 1 0 0 0  
## 6 1 0 0 0  
## 7 1 0 0 0  
## 8 1 0 0 0  
## 9 1 0 0 0  
## 10 1 0 0 0  
## # ... with 30 more rows

Nothing happens because the reference group was previously the intercept but it is now an extra category. Consequently, predictions and residuals will be the same.

grid <-   
 sim2 %>%   
 data\_grid(x) %>%   
 add\_predictions(mod2)  
  
ggplot(sim2,aes(x))+  
 geom\_point(aes(y=y))+  
 geom\_point(data=grid,aes(y=pred),colour="red",size=4)



1. Use model\_matrix() to explore the equations generated for the models I fit to sim3 and sim4. Why is \* a good shorthand for interaction?

mod1 <- lm(y ~ x1 + x2, data = sim3)  
mod2 <- lm(y ~ x1 \* x2, data = sim3)  
  
mod1 <- lm(y ~ x1 + x2, data = sim4)  
mod2 <- lm(y ~ x1 \* x2, data = sim4)  
  
model\_matrix(sim3, y ~ x1 + x2)

## # A tibble: 240 x 5  
## `(Intercept)` x1 x2b x2c x2d  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 1 0 0 0  
## 2 1 1 0 0 0  
## 3 1 1 0 0 0  
## 4 1 1 1 0 0  
## 5 1 1 1 0 0  
## 6 1 1 1 0 0  
## 7 1 1 0 1 0  
## 8 1 1 0 1 0  
## 9 1 1 0 1 0  
## 10 1 1 0 0 1  
## # ... with 230 more rows

model\_matrix(sim3, y ~ x1 \* x2)

## # A tibble: 240 x 8  
## `(Intercept)` x1 x2b x2c x2d `x1:x2b` `x1:x2c` `x1:x2d`  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 1 0 0 0 0 0 0  
## 2 1 1 0 0 0 0 0 0  
## 3 1 1 0 0 0 0 0 0  
## 4 1 1 1 0 0 1 0 0  
## 5 1 1 1 0 0 1 0 0  
## 6 1 1 1 0 0 1 0 0  
## 7 1 1 0 1 0 0 1 0  
## 8 1 1 0 1 0 0 1 0  
## 9 1 1 0 1 0 0 1 0  
## 10 1 1 0 0 1 0 0 1  
## # ... with 230 more rows

model\_matrix(sim4,y~x1+x2)

## # A tibble: 300 x 3  
## `(Intercept)` x1 x2  
## <dbl> <dbl> <dbl>  
## 1 1 -1 -1   
## 2 1 -1 -1   
## 3 1 -1 -1   
## 4 1 -1 -0.778  
## 5 1 -1 -0.778  
## 6 1 -1 -0.778  
## 7 1 -1 -0.556  
## 8 1 -1 -0.556  
## 9 1 -1 -0.556  
## 10 1 -1 -0.333  
## # ... with 290 more rows

model\_matrix(sim4,y~x1\*x2)

## # A tibble: 300 x 4  
## `(Intercept)` x1 x2 `x1:x2`  
## <dbl> <dbl> <dbl> <dbl>  
## 1 1 -1 -1 1   
## 2 1 -1 -1 1   
## 3 1 -1 -1 1   
## 4 1 -1 -0.778 0.778  
## 5 1 -1 -0.778 0.778  
## 6 1 -1 -0.778 0.778  
## 7 1 -1 -0.556 0.556  
## 8 1 -1 -0.556 0.556  
## 9 1 -1 -0.556 0.556  
## 10 1 -1 -0.333 0.333  
## # ... with 290 more rows

The \* operator in a formula expands it to include both the main effecnt and the interaction between the two variables.

1. Using the basic principles, convert the formulas in the following two models into functions. (Hint: start by converting the categorical variable into 0-1 variables.)

mod1 <- lm(y~x1+x2,data=sim3)  
mod2 <- lm(y~x1\*x2,data=sim3)

Please refer: <https://jrnold.github.io/r4ds-exercise-solutions/model-basics.html#formulas-and-model-families>

model\_matrix\_mod1 <- function(.data) {  
 mutate(.data,  
 `x2b` = as.numeric(x2 == "b"),  
 `x2c` = as.numeric(x2 == "c"),  
 `x2d` = as.numeric(x2 == "d"),  
 `x1:x2b` = x1 \* x2b,  
 `x1:x2c` = x1 \* x2c,  
 `x1:x2d` = x1 \* x2d) %>%  
 select(x1, x2b, x2c, x2d, `x1:x2b`, `x1:x2c`, `x1:x2d`)  
}  
  
model\_matrix\_mod2 <- function(.data){  
 mutate(.data,`x1:x2`=x1\*x2) %>%   
 select(x1,x2,`x1:x2`)  
}  
  
# model\_matrix\_mod1(sim3)  
model\_matrix\_mod2(sim4)

## # A tibble: 300 x 3  
## x1 x2 `x1:x2`  
## <dbl> <dbl> <dbl>  
## 1 -1 -1 1   
## 2 -1 -1 1   
## 3 -1 -1 1   
## 4 -1 -0.778 0.778  
## 5 -1 -0.778 0.778  
## 6 -1 -0.778 0.778  
## 7 -1 -0.556 0.556  
## 8 -1 -0.556 0.556  
## 9 -1 -0.556 0.556  
## 10 -1 -0.333 0.333  
## # ... with 290 more rows

1. For sim4, which of mod1 and mod2 is better? I think mod2 does a slightly better job at removing patterns, but it’s pretty subtle. Can you come up with a plot to support my claim?

mod1 <- lm(y ~ x1 + x2, data = sim4)  
mod2 <- lm(y ~ x1 \* x2, data = sim4)  
  
resid\_res <-  
 sim4 %>%   
 gather\_residuals(mod1, mod2)  
  
resid\_res %>%   
 ggplot(aes(x = resid, colour = model)) +  
 geom\_freqpoly(binwidth = 0.5)



The distribution shows that the right-most side of the residuals are much smoother for the second model and closer to the central distribution.

## 23.5 Missing values

Missing values obviously can not convey any information about the relationship between the variables, so modelling functions will drop any rows that contain missing values. R’s default behaviour is to silently drop them, but options(na.action = na.warn) (run in the prerequisites), makes sure you get a warning.

library(tidyverse)  
df <- tribble(  
 ~x,~y,  
 1,2.2,  
 2,NA,  
 3,3.5,  
 4,8.3,  
 NA,10  
)  
  
mod <- lm(y~x,data=df)  
mod

##   
## Call:  
## lm(formula = y ~ x, data = df)  
##   
## Coefficients:  
## (Intercept) x   
## -0.2286 1.8357

To suppores the warining, set set.action=na.exclude you can always see exactly how many observations were used with nobs().

modl <- lm(y~x,data=df,na.action=na.exclude)  
modl

##   
## Call:  
## lm(formula = y ~ x, data = df, na.action = na.exclude)  
##   
## Coefficients:  
## (Intercept) x   
## -0.2286 1.8357

nobs(mod)

## [1] 3

## 23.6 Other model families

This chapter has focussed exclusively on the class of linear models, which assume a relationship of the form . Linear models additionally assume that the residuals have a normal distribution, which we haven’t talked about. There are a large set of model classes that extend the linear model in various interesting ways. Some of them are:

1. Generalised linear models, e.g. stats::glm(). Linear models assume that the response is continuous and the error has a normal distribution. Generalised linear models extend linear models to include non-continuous responses (e.g. binary data or counts). They work by defining a distance metric based on the statistical idea of likelihood.
2. Generalised additive models, e.g. mgcv::gam(), extend generalised linear models to incorporate arbitrary smooth functions. That means you can write a formula like y ~ s(x) which becomes an equation like y = f(x) and let gam() estimate what that function is (subject to some smoothness constraints to make the problem tractable).
3. Penalised linear models, e.g. glmnet::glmnet(), add a penalty term to the distance that penalises complex models (as defined by the distance between the parameter vector and the origin). This tends to make models that generalise better to new datasets from the same population.
4. Robust linear models, e.g. MASS:rlm(), tweak the distance to downweight points that are very far away. This makes them less sensitive to the presence of outliers, at the cost of being not quite as good when there are no outliers.
5. Trees, e.g. rpart::rpart(), attack the problem in a completely different way than linear models. They fit a piece-wise constant model, splitting the data into progressively smaller and smaller pieces. Trees aren’t terribly effective by themselves, but they are very powerful when used in aggregate by models like random forests (e.g. randomForest::randomForest()) or gradient boosting machines (e.g. xgboost::xgboost.)

These models all work similarly from a programming perspective. Once you’ve mastered linear models, you should find it easy to master the mechanics of these other model classes. Being a skilled modeller is a mixture of some good general principles and having a big toolbox of techniques. Now that you’ve learned some general tools and one useful class of models, you can go on and learn more classes from other sources.

# Chapter 24: Model building

## 24.1 Introduction

In the previous chapter you learned how linear models work, and learned some basic tools for understanding what a model is telling you about your data. The previous chapter focussed on simulated datasets. This chapter will focus on real data, showing you how you can progressively build up a model to aid your understanding of the data.

We will take advantage of the fact that you can think about a model partitioning your data into pattern and residuals. We’ll find patterns with visualisation, then make them concrete and precise with a model. We’ll then repeat the process, but replace the old response variable with the residuals from the model. The goal is to transition from implicit knowledge in the data and your head to explicit knowledge in a quantitative model. This makes it easier to apply to new domains, and easier for others to use.

For very large and complex datasets this will be a lot of work. There are certainly alternative approaches - a more machine learning approach is simply to focus on the predictive ability of the model. These approaches tend to produce black boxes: the model does a really good job at generating predictions, but you don’t know why. This is a totally reasonable approach, but it does make it hard to apply your real world knowledge to the model. That, in turn, makes it difficult to assess whether or not the model will continue to work in the long-term, as fundamentals change. For most real models, I’d expect you to use some combination of this approach and a more classic automated approach.

t’s a challenge to know when to stop. You need to figure out when your model is good enough, and when additional investment is unlikely to pay off. I particularly like this quote from reddit user Broseidon241:

A long time ago in art class, my teacher told me “An artist needs to know when a piece is done. You can’t tweak something into perfection - wrap it up. If you don’t like it, do it over again. Otherwise begin something new”. Later in life, I heard “A poor seamstress makes many mistakes. A good seamstress works hard to correct those mistakes. A great seamstress isn’t afraid to throw out the garment and start over.” - Broseidon241, <https://www.reddit.com/r/datascience/comments/4irajq>

### 24.1.1 Prerequisites

We’ll use the same tools as in the previous chapter, but add in some real datasets: diamonds from ggplot2, and flights from nycflights13. We’ll also need lubridate in order to work with the date/times in flights.

library(tidyverse)  
library(modelr)  
options(na.action=na.warn)  
  
library(nycflights13)

## Warning: package 'nycflights13' was built under R version 3.5.1

library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## date

library(magrittr)

##   
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':  
##   
## set\_names

## The following object is masked from 'package:tidyr':  
##   
## extract

## 24.2 Why are low quality diamonds more expensive

In previous chapters, we have seen a surprising relationship between the quality of diamonds and their price: low quality diamonds (poor cuts, bad colours, and inferior clarity) have higher prices.

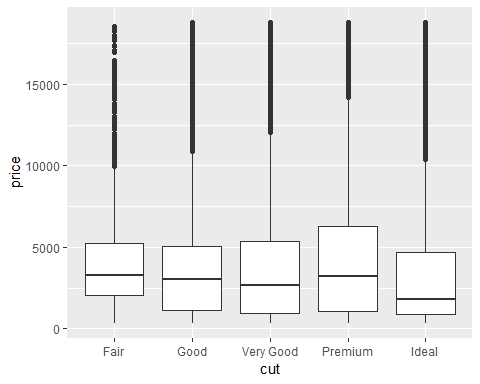
# data overview  
head(diamonds)

## # A tibble: 6 x 10  
## carat cut color clarity depth table price x y z  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43  
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31  
## 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31  
## 4 0.290 Premium I VS2 62.4 58 334 4.2 4.23 2.63  
## 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75  
## 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48

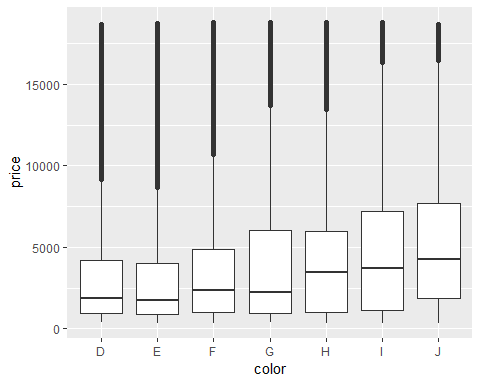
diamonds %>%   
 group\_by(clarity) %>%   
 summarise(mean=mean(price))

## # A tibble: 8 x 2  
## clarity mean  
## <ord> <dbl>  
## 1 I1 3924.  
## 2 SI2 5063.  
## 3 SI1 3996.  
## 4 VS2 3925.  
## 5 VS1 3839.  
## 6 VVS2 3284.  
## 7 VVS1 2523.  
## 8 IF 2865.

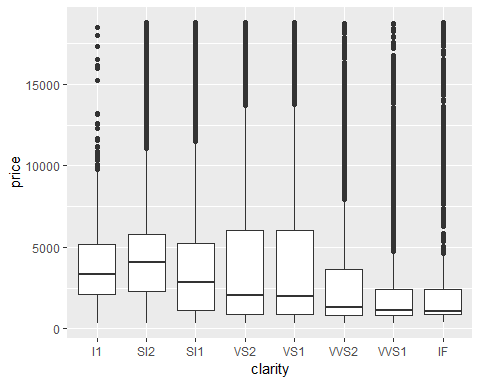
ggplot(diamonds,aes(cut,price))+geom\_boxplot()



ggplot(diamonds,aes(color,price))+geom\_boxplot()



ggplot(diamonds,aes(clarity,price))+geom\_boxplot()

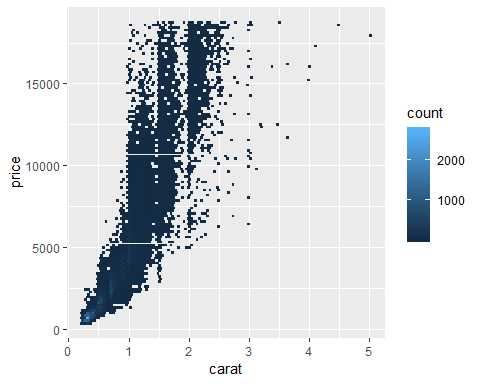


### 24.2.1 Price and carat

It looks like lower quality diamonds have higher prices because there is an important confounding variable: the weight(carat) of the diamond. The weight of the diamond. The weight of the diamond is the single most important factor for determining thde price of the diamond, and lower quality diamonds tend to be larger.

ggplot(diamonds,aes(carat,price))+  
 geom\_hex(bins=100)

## Warning: package 'hexbin' was built under R version 3.5.1



We can make it easier to see how the other attributes of a diamond affect its relative price by fitting a model to separate out the effect of carat. But first, lets make a couple of tweaks to the diamonds dataset to make it easier to work with:

1. Focus on diamonds smaller than 2.5 carat(99.7% of the data)

# data count  
n <- diamonds %>%   
 ungroup() %>%   
 count()  
  
n\_1 <- diamonds %>%   
 filter(carat<=2.5) %>%   
 count()  
n\_1/n

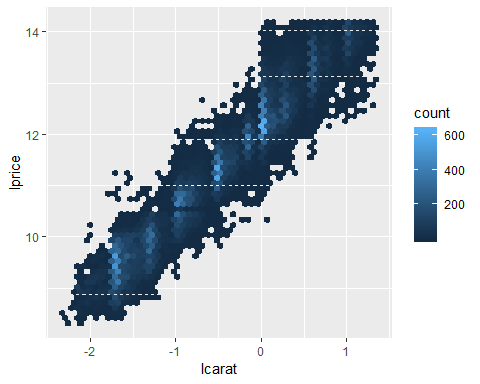
## n  
## 1 0.9976641

1. Log-transform the carat and price variables

diamonds2 <- diamonds %>%   
 filter(carat<=2.5) %>%   
 mutate(lprice=log2(price),  
 lcarat=log2(carat))

Together, these changes make it easier to see the relationship between carat and price.

ggplot(diamonds2, aes(lcarat,lprice))+  
 geom\_hex(bins=50)

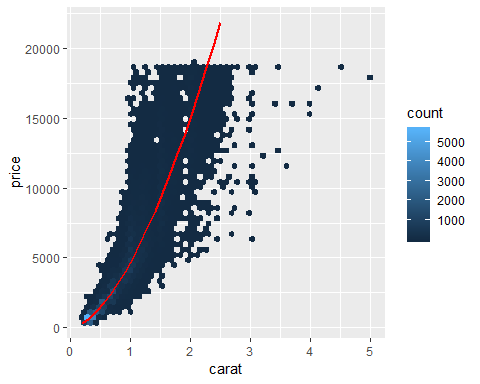


The log-transformation is particularly useful here because it makes the pattern linear, and linear patterns are the easiest to work with. Let’s take the next step and remove that strong linear pattern. We first make the pattern explicit by fitting a model:

mod\_diamonds <- lm(lprice~lcarat,data=diamonds2)

Then, we look at what the model tells us about the data. Note that I back transform the predictions, undoing the log transformation, so I can overlay the predictions on the raw data:

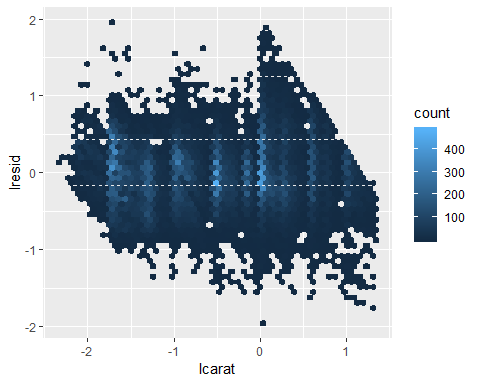
grid <- diamonds2 %>%  
 data\_grid(carat=seq\_range(carat,20)) %>%   
 mutate(lcarat=log2(carat)) %>%   
 add\_predictions(mod\_diamonds,"lprice") %>%   
 mutate(price=2^lprice)  
  
ggplot(diamonds,aes(carat,price))+  
 geom\_hex(bins=50)+  
 geom\_line(data=grid,colour="red",size=1)



That tells us something interesting about our data. If we believe our model, then the large diamonds are much cheaper than expected. This is probably because no diamond in this dataset costs more than $19,000.

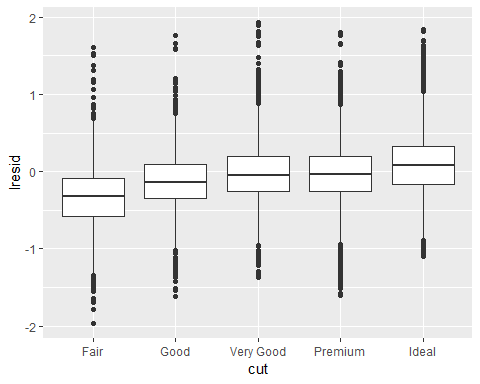
Now we can look at the residuals, which verifies that we have successfully removed the strong linear pattern:

diamonds2 <- diamonds2 %>%   
 add\_residuals(mod\_diamonds,"lresid")  
  
ggplot(diamonds2,aes(lcarat,lresid))+  
 geom\_hex(bins=50)

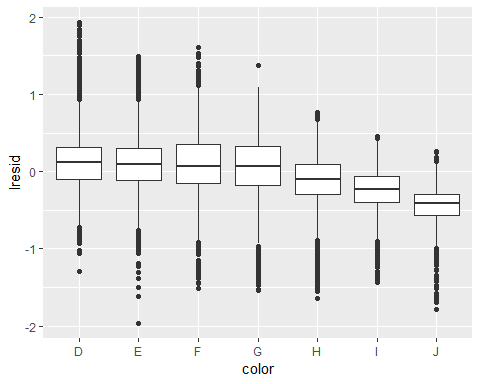


Impoortantly, we can now re-do motivating plots using those residuals instead of price.

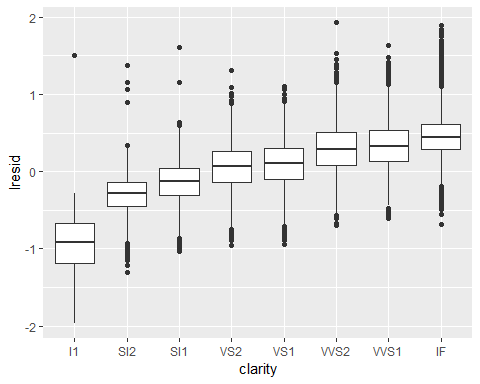
ggplot(diamonds2, aes(cut,lresid))+geom\_boxplot()



ggplot(diamonds2, aes(color,lresid))+geom\_boxplot()



ggplot(diamonds2, aes(clarity,lresid))+geom\_boxplot()



Now we see the relationship we expect: as the quality of the diamond increases, so too does its relative price. To interpret the y axis, we need to think about what the residuals are telling us, and what scale they are on. A residual of -1 indicates that lprice was 1 unit lower than a prediction based solely on its weight. is 1/2, points with a value of -1 are half the expected price, and residuals with value 1 are twice the predicted price.

### 24.2.2 A more complicated model

If we wanted to, we could continue to build up our model, moving the effects we’ve observed into the model to make them explicit. For example, we could include color, cut, and clarity into the model so that we also make explicit the effect of these three categorical variables:

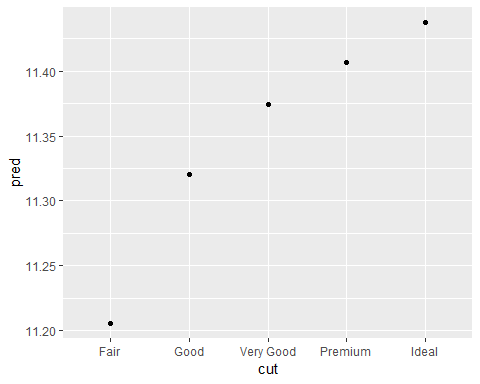
mod\_diamonds2 <- lm(lprice~lcarat+color+cut+clarity,data=diamonds2)

This model now includes four predictors, so it’s getting hader to visualize. Fortunately, they’re currentl all independent which means that we can plot them individually in four plots. To make the process a little easier, we’re going to use the .model argument to data\_grid:

grid <- diamonds2 %>%  
 data\_grid(cut,.model=mod\_diamonds2) %>%   
 add\_predictions(mod\_diamonds2)  
grid

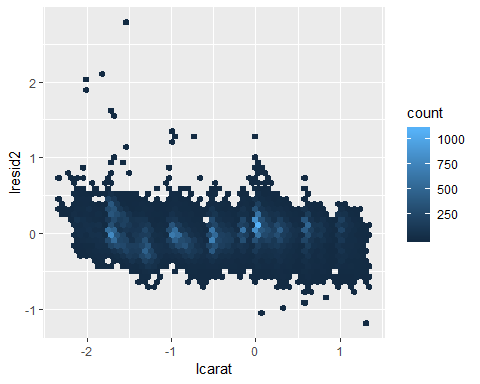
## # A tibble: 5 x 5  
## cut lcarat color clarity pred  
## <ord> <dbl> <chr> <chr> <dbl>  
## 1 Fair -0.515 G VS2 11.2  
## 2 Good -0.515 G VS2 11.3  
## 3 Very Good -0.515 G VS2 11.4  
## 4 Premium -0.515 G VS2 11.4  
## 5 Ideal -0.515 G VS2 11.4

ggplot(grid,aes(cut,pred))+  
 geom\_point()



If the model needs variables that you haven’t explicitly supplied, data\_grid() will automatically fill them in with “typical” value. For continuous variables, it uses the median, and categorical variables it uses the most common value (or values, if there is a tie).

diamonds2 <- diamonds2 %>%   
 add\_residuals(mod\_diamonds2,"lresid2")  
  
ggplot(diamonds2,aes(lcarat,lresid2))+  
 geom\_hex(bins=50)



This plot indicates that there are some diamonds with quite large residuals - remember a residual of 2 indicates that the diamond is 4x the price that we expected. It’s often useful to look at unusual values individually:

diamonds2 %>%   
 filter(abs(lresid2) > 1) %>%   
 add\_predictions(mod\_diamond2) %>%   
 mutate(pred = round(2 ^ pred)) %>%   
 select(price, pred, carat:table, x:z) %>%   
 arrange(price)

Nothing really jumps out at me here, but it’s probably worth spending time considering if this indicates a problem with our model, or if there are errors in the data. If there are mistakes in the data, this could be an opportunity to buy diamonds that have been priced low incorrectly.

### 24.2.3 Exercises

1. In the plot of lcarat vs. lprice, there are some bright vertical strips. What do they represent?

Those represents the categories of carat which is in fact an integer variable. But because we logged the initial variables we also get results different from integers.

1. If , what does that say about the relationship between price and carat?

That the price of a diamond is completely dependent on the carat size but only when the relationship is in a multiplicative or linear fashion. A 1% increase in carat is associated with a 1% increase in price.

1. Extract the diamonds that have very high and very low residuals. Is there anything unusual about these diamonds? Are they particularly bad or good, or do you think these are pricing errors?

library(tidyverse)  
  
diamonds2 <-   
 diamonds %>%   
 mutate(lprice=log2(price),  
 lcarat=log2(carat))  
  
mod1 <- lm(lprice~lcarat+color+clarity+cut,data=diamonds2)  
  
bottom <-  
 diamonds2 %>%   
 add\_residuals(mod1) %>%   
 arrange(resid) %>%   
 slice(1:10)  
  
top <-  
 diamonds2 %>%   
 add\_residuals(mod1) %>%   
 arrange(-resid) %>%   
 slice(1:10)  
  
bind\_rows(bottom,top) %>%   
 select(price,carat,resid)

## # A tibble: 20 x 3  
## price carat resid  
## <int> <dbl> <dbl>  
## 1 6512 3 -1.46   
## 2 10470 2.46 -1.17   
## 3 10453 3.05 -1.14   
## 4 14220 3.01 -1.12   
## 5 9925 3.01 -1.12   
## 6 18701 3.51 -1.09   
## 7 1262 1.03 -1.04   
## 8 8040 3.01 -1.02   
## 9 12587 3.5 -0.990  
## 10 8044 3 -0.985  
## 11 2160 0.34 2.81   
## 12 1776 0.290 2.10   
## 13 1186 0.25 2.06   
## 14 1186 0.25 2.06   
## 15 1013 0.25 1.94   
## 16 2366 0.3 1.61   
## 17 1715 0.32 1.57   
## 18 4368 0.51 1.36   
## 19 10011 1.01 1.31   
## 20 3807 0.61 1.31

1. Does the final model, mod\_diamond2, do a good job of predicting diamond prices? Would you trust it to tell you how much to spend if you were buying a diamond?

diamonds2 %>%  
 add\_predictions(mod1) %>%  
 mutate(pred=2^pred) %>%  
 select(price,pred) %>%   
 mutate(se=predict(mod1,se.fit=T)$se.fit,  
 low\_ci=pred-se\*2,  
 upper\_ci=pred+se\*2,  
 correct=if\_else(price>=low\_ci&price<=upper\_ci,TRUE,FALSE)) %>%  
 summarise(prop\_correct=mean(correct))

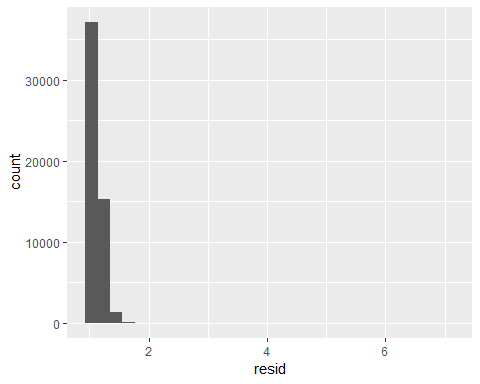
## # A tibble: 1 x 1  
## prop\_correct  
## <dbl>  
## 1 0

It doesn’t look like very good model at predicting because 0% of the predictions were close to the actual price. This is based on the 95% interval.

We could do it separately and check the magniturde of the residuals

diamonds2 %>%   
 add\_residuals(mod1) %>%   
 mutate(resid=2^abs(resid)) %>%   
 ggplot(aes(resid))+  
 geom\_histogram()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



## 24.3 What affects the number of daily flights?

Let’s work through a similar process for a dataset that seems even simpler at first glance: the number of flights that leave NYC per day. This is a really small dataset — only 365 rows and 2 columns — and we’re not going to end up with a fully realised model, but as you’ll see, the steps along the way will help us better understand the data. Let’s get started by counting the number of flights per day and visualising it with ggplot2.

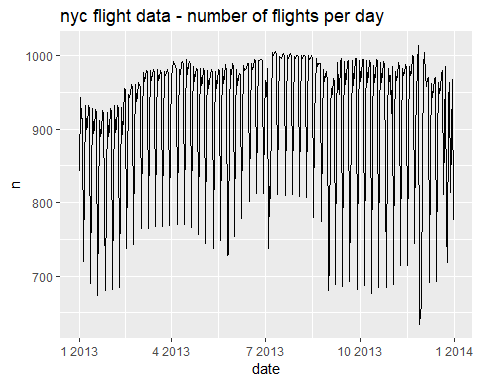
library(tidyverse)  
library(nycflights13)  
library(lubridate)  
library(ggplot2)  
library(modelr)  
  
# data overview  
head(flights)

## # A tibble: 6 x 19  
## year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
## <int> <int> <int> <int> <int> <dbl> <int>  
## 1 2013 1 1 517 515 2 830  
## 2 2013 1 1 533 529 4 850  
## 3 2013 1 1 542 540 2 923  
## 4 2013 1 1 544 545 -1 1004  
## 5 2013 1 1 554 600 -6 812  
## 6 2013 1 1 554 558 -4 740  
## # ... with 12 more variables: sched\_arr\_time <int>, arr\_delay <dbl>,  
## # carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
## # air\_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,  
## # time\_hour <dttm>

daily <- flights %>%   
 mutate(date=make\_date(year,month,day)) %>%   
 group\_by(date) %>%   
 summarise(n=n())  
head(daily)

## # A tibble: 6 x 2  
## date n  
## <date> <int>  
## 1 2013-01-01 842  
## 2 2013-01-02 943  
## 3 2013-01-03 914  
## 4 2013-01-04 915  
## 5 2013-01-05 720  
## 6 2013-01-06 832

# data plot  
ggplot(data=daily)+  
 geom\_line(aes(x=date,y=n))+  
 ggtitle("nyc flight data - number of flights per day")



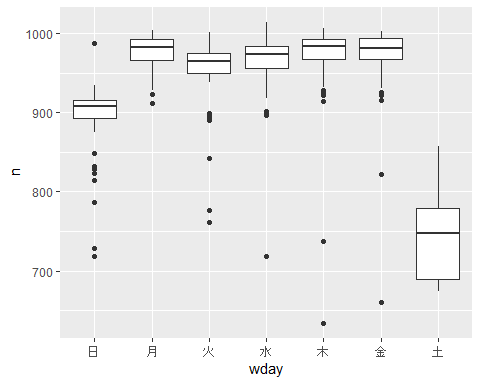
ylab("number of flights per day")

## $y  
## [1] "number of flights per day"  
##   
## attr(,"class")  
## [1] "labels"

### 24.3.1 Day of Work

Understanding the long-term trend is challenging because there’s a very strong day-of-week effect that dominates the subtler patterns. Let’s start by looking at the distribution of flight numbers by day-of-week:

daily <- daily %>%   
 mutate(wday = wday(date, label = TRUE))  
ggplot(daily, aes(wday, n)) +   
 geom\_boxplot()



There are fewer flights on weekends because most travel is for business. The effect is particularly pronounced on Saturday: you might sometimes leave on Sunday for a Monday morning meeting, but it’s very rare that you’d leave on Saturday as you’d much rather be at home with your family.

One way to remove this strong pattern is to use a model. First, we fit the model, and display its predictions overlaid on the original data:

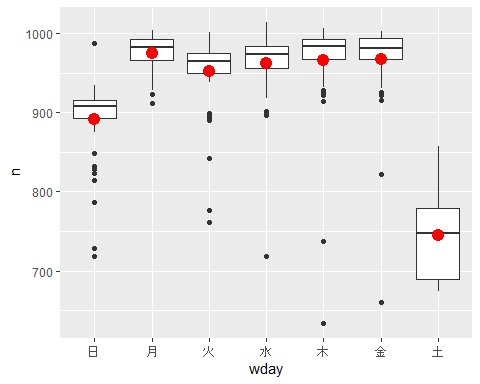
mod <- lm(n ~ wday, data = daily)  
summary(mod)

##   
## Call:  
## lm(formula = n ~ wday, data = daily)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -331.75 -8.69 12.31 25.19 112.38   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 922.595 2.554 361.209 <2e-16 \*\*\*  
## wday.L -83.322 6.765 -12.317 <2e-16 \*\*\*  
## wday.Q -155.111 6.760 -22.945 <2e-16 \*\*\*  
## wday.C -62.834 6.756 -9.300 <2e-16 \*\*\*  
## wday^4 -80.126 6.766 -11.842 <2e-16 \*\*\*  
## wday^5 -4.967 6.748 -0.736 0.4622   
## wday^6 -16.934 6.751 -2.508 0.0126 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 48.8 on 358 degrees of freedom  
## Multiple R-squared: 0.7178, Adjusted R-squared: 0.7131   
## F-statistic: 151.8 on 6 and 358 DF, p-value: < 2.2e-16

grid <- daily %>%   
 data\_grid(wday) %>%   
 add\_predictions(mod,"n")  
head(grid)

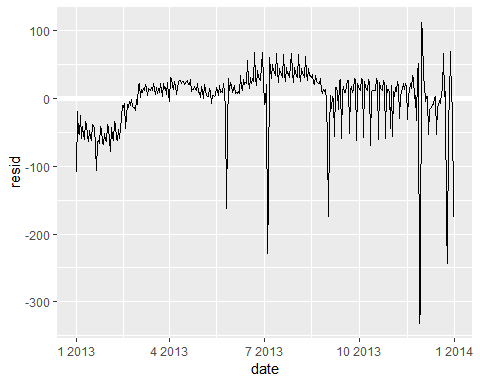
## # A tibble: 6 x 2  
## wday n  
## <ord> <dbl>  
## 1 日 891.  
## 2 月 975.  
## 3 火 951.  
## 4 水 963.  
## 5 木 966.  
## 6 金 967.

ggplot(daily,aes(wday,n))+  
 geom\_boxplot()+  
 geom\_point(data=grid,colour="red",size=4)



Next we compute and visualise the residuals:

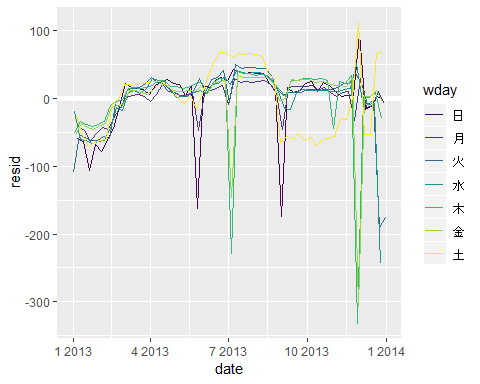
daily <- daily %>%   
 add\_residuals(mod)  
  
daily %>%   
 ggplot(aes(date,resid))+  
 geom\_ref\_line(h=0)+  
 geom\_line()



Note the change in the y-axis: now we are seeing the deviation from the expected number of flights, given the day of week. This plot is useful because now that we’ve removed much of the large day-of-week effect, we can see some of the subtler patterns that remain:

1. Our model seems to fail starting in June: you can still see a strong regular pattern that our model hasn’t captured. Drawing a plot with one line for each day of the week makes the cause easier to see:

ggplot(daily,aes(date,resid,colour=wday))+  
 geom\_ref\_line(h=0)+  
 geom\_line()



Our model fails to accurately predict the number of flights on Saturday: during summer there are more flights than we expect, and during Fall there are fewer. We’ll see how we can do better to capture this pattern in the next section.

1. There are some days with far fewer flights than expected:

daily %>%   
 filter(resid < -100)

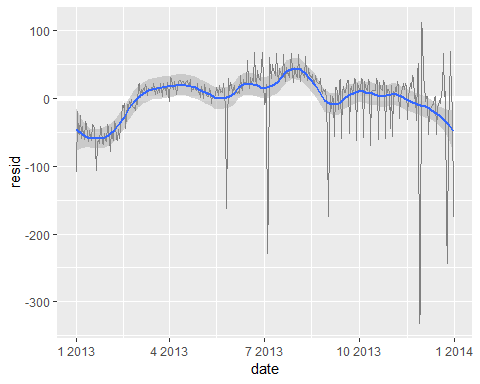
## # A tibble: 11 x 4  
## date n wday resid  
## <date> <int> <ord> <dbl>  
## 1 2013-01-01 842 火 -109.  
## 2 2013-01-20 786 日 -105.  
## 3 2013-05-26 729 日 -162.  
## 4 2013-07-04 737 木 -229.  
## 5 2013-07-05 822 金 -145.  
## 6 2013-09-01 718 日 -173.  
## 7 2013-11-28 634 木 -332.  
## 8 2013-11-29 661 金 -306.  
## 9 2013-12-24 761 火 -190.  
## 10 2013-12-25 719 水 -244.  
## 11 2013-12-31 776 火 -175.

If you’re familiar with American public holidays, you might spot New Year’s day, July 4th, Thanksgiving and Christmas. There are some others that don’t seem to correspond to public holidays. You’ll work on those in one of the exercises.

1. There seems to be some smoother long term trend over the course of the year. We can highlight that trend with geom\_smooth():

daily %>%   
 ggplot(aes(date,resid))+  
 geom\_ref\_line(h=0)+  
 geom\_line(colour="grey50")+  
 geom\_smooth(se=TRUE,span=0.20)

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'

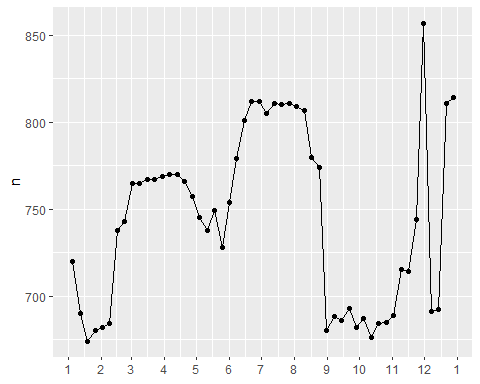


There are fewer flights in January (and December), and more in summer (May-Sep). We can’t do much with this pattern quantitatively, because we only have a single year of data. But we can use our domain knowledge to brainstorm potential explanations.

### 24.3.2 Seasonal Saturday effect

Let’s first tackle our failure to accurately predict the number of flights on Saturday. A good place to start is to go back to the raw numbers, focussing on Saturdays:

daily %>%   
 filter(wday=="土") %>%   
 ggplot(aes(date,n))+  
 geom\_point()+  
 geom\_line()+  
 scale\_x\_date(NULL,date\_breaks = "1 month",date\_labels="%b")



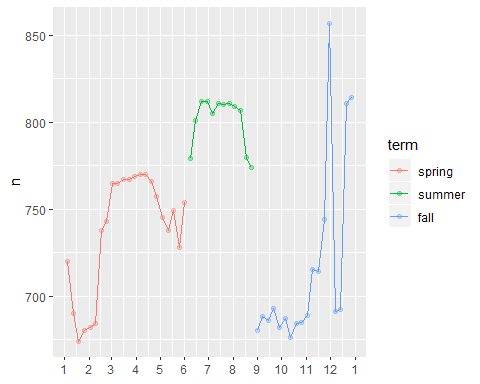
I have both points and lines to make it more clear what is data and what is interpolation.

I suspect this pattern is caused by summer holidays: many people go on holiday in the summer, and people don’t mind traveliing on Saturdays for vacation. . Looking at this plot, we might guess that summer holidays are from early June to late August. That seems to line up fairly well with the state’s school terms: summer break in 2013 was Jun 26–Sep 9.

Why are there more Saturday flights in the Spring than the Fall? I asked some American friends and they suggested that it’s less common to plan family vacations during the Fall because of the big Thanksgiving and Christmas holidays. We don’t have the data to know for sure, but it seems like a plausible working hypothesis.

Lets create a “term” variable that roughly captures the three school terms, and check our work with a plot:

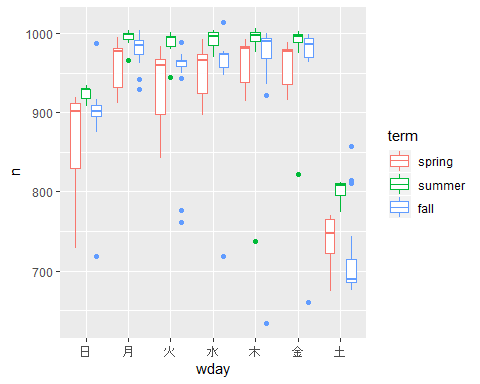
library(lubridate)  
  
term <- function(date){  
 cut(date,  
 breaks = ymd(20130101,20130605,20130825,20140101),  
 labels=c("spring","summer","fall")  
 )  
}  
  
daily <- daily %>%   
 mutate(term = term(date))   
  
daily %>%   
 filter(wday=="土") %>%   
 ggplot(aes(date,n,colour=term))+  
 geom\_point(alpha=1/3)+  
 geom\_line()+  
 scale\_x\_date(NULL,date\_breaks = "1 month",date\_labels="%b")



(I manually tweaked the dates to get nice breaks in the plot. Using a visualisation to help you understand what your function is doing is a really powerful and general technique.)

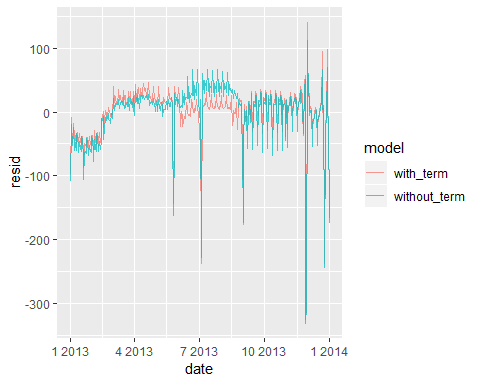
It’s useful to see how this new variable affects the other days of the week:

daily %>%   
 ggplot(aes(wday,n,colour=term))+  
 geom\_boxplot()



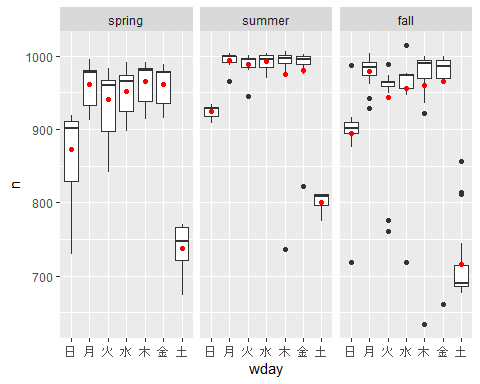
It looks like there is significant variation across the terms, so fitting a separate day of week effect for each term is reasonable. This improves our model, but not as much as we might hope:

mod1 <- lm(n ~ wday, data = daily)  
mod2 <- lm(n ~ wday \* term, data = daily)  
  
daily %>%   
 gather\_residuals(without\_term = mod1, with\_term = mod2) %>%   
 ggplot(aes(date, resid, colour = model)) +  
 geom\_line(alpha = 0.75)



We can see the problem by overlaying the predictions from the model on to the raw data:

grid <- daily %>%  
 data\_grid(wday,term) %>%   
 add\_predictions(mod2,"n")  
  
ggplot(daily,aes(wday,n))+  
 geom\_boxplot()+  
 geom\_point(data=grid,colour="red")+  
 facet\_wrap(~term)

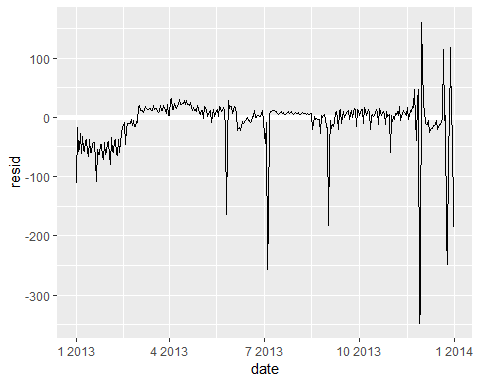


Our model is finding the mean effect, but we have a lot of big outliers, so mean tends to be far away from the typical value. We can alleviate this problem by using a model that is robust to the effect of outliers: MASS::rlm(). This greatly reduces the impact of the outliers on our estimates, and gives a model that does a good job of removing the day of week pattern:

head(daily)

## # A tibble: 6 x 5  
## date n wday resid term   
## <date> <int> <ord> <dbl> <fct>   
## 1 2013-01-01 842 火 -109. spring  
## 2 2013-01-02 943 水 -19.7 spring  
## 3 2013-01-03 914 木 -51.8 spring  
## 4 2013-01-04 915 金 -52.5 spring  
## 5 2013-01-05 720 土 -24.6 spring  
## 6 2013-01-06 832 日 -59.5 spring

mod3 <- MASS::rlm(n~wday\*term,data=daily)  
  
daily %>%   
 add\_residuals(mod3,"resid") %>%   
 ggplot(aes(date,resid))+  
 geom\_hline(yintercept=0,size=2,colour="white")+  
 geom\_line()



It’S now much easier to see the long-term trend, and the positive and negative outliers.

### 24.3.3 Computed variables

If you are experimenting with many models and many visualizations, It’s a good idea to bundle the creation of variables up into a function so there is no chance of accidentally applying a different transformation in different places.

For example, we could write

compute\_vars <- function(data) {  
 data %>%   
 mutate(  
 term = term(date),   
 wday = wday(date, label = TRUE)  
 )  
}

Another option is to put the transformations directly in the model formula:

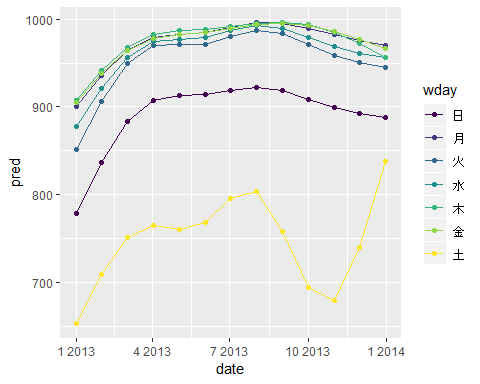
wday2 <- function(x) wday(x, label = TRUE)  
mod3 <- lm(n ~ wday2(date) \* term(date), data = daily)

Either approach is reasonable. Making the transformed variable explicit is useful if you want to check your work, or use them in a visualisation. But you can’t easily use transformations (like splines) that return multiple columns. Including the transformations in the model function makes life a little easier when you’re working with many different datasets because the model is self contained.

### 24.3.4 Time of year: an alternative approach

In the previous section we used our domain knowledge (how the US school term affects travel) to improve the model. An alternative to using our knowledge explicitly in the model is to give the data more room to speak. We could use a more flexible model and allow that to capture the pattern we’re interested in. A simple linear trend isn’t adequate, so we could try using a natural spline to fit a smooth curve across the year:

library(splines)  
library(magrittr)  
library(modelr)  
library(tidyverse)  
  
mod <- MASS::rlm(n ~ wday \* ns(date, 5), data = daily)  
  
daily %>%   
 data\_grid(wday, date = seq\_range(date, n = 13)) %>%   
 add\_predictions(mod) %>%   
 ggplot(aes(date, pred, colour = wday)) +   
 geom\_line() +  
 geom\_point()



We see a strong pattern in the numbers of Saturday flights. This is reassuring, because we also saw that pattern in the raw data. It’s a good sign when you get the same signal from different approaches.

## 24.4 Learning more about models

We have only scratched the absolute surface of modelling, but you have hopefully gained some simple, but general-purpose tools that you can use to improve your own data analyses. It’s OK to start simple! As you’ve seen, even very simple models can make a dramatic difference in your ability to tease out interactions between variables.

These modelling chapters are even more opinionated than the rest of the book. I approach modelling from a somewhat different perspective to most others, and there is relatively little space devoted to it. Modelling really deserves a book on its own, so I’d highly recommend that you read at least one of these three books:

* Statistical Modeling: A Fresh Approach by Danny Kaplan, <http://www.mosaic-web.org/go/StatisticalModeling/>. This book provides a gentle introduction to modelling, where you build your intuition, mathematical tools, and R skills in parallel. The book replaces a traditional “introduction to statistics” course, providing a curriculum that is up-to-date and relevant to data science.
* An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, <http://www-bcf.usc.edu/~gareth/ISL/> (available online for free). This book presents a family of modern modelling techniques collectively known as statistical learning. For an even deeper understanding of the math behind the models, read the classic Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani, and Jerome Friedman, <http://statweb.stanford.edu/~tibs/ElemStatLearn/> (also available online for free).
* Applied Predictive Modeling by Max Kuhn and Kjell Johnson, <http://appliedpredictivemodeling.com>. This book is a companion to the caret package and provides practical tools for dealing with real-life predictive modelling challenges.

# Chapter 25: Many models

## 25.1 Introduction

In this chapter, you are going to learn thre powerful ideas that help you to work with large numbers of models with ease:

1. Using many simple models to better understand complex datasets.
2. Using list-columns to store arbitrary data structures in a data frame. For example, this will allow you to have a column that contains linear models.
3. Using the **broom** package, by David Robinson, to turn models into tidy data. This is a powerful technique for working with large numbers of models because once you have tidy data, you can apply all of the techniques that you’ve learned about earlier in the book.

We’ll start by diving into a motivating example using data about life expectancy around the world. It’s a small dataset but it illustrates how important modelling can be for improving your visualisations. We’ll use a large number of simple models to partition out some of the strongest signals so we can see the subtler signals that remain. We’ll also see how model summaries can help us pick out outliers and unusual trends.

The following sectins will dive into more details about the individual techniques:

1. In [list-columns](http://r4ds.had.co.nz/many-models.html#list-columns-1), you’ll learn more about the list-column data structure, and why it’s valid to put lists in data frames.
2. In [creating list-columns](http://r4ds.had.co.nz/many-models.html#creating-list-columns), you’ll learn the three main ways in which you’ll create list-columns.
3. In [simplifying list-columns](http://r4ds.had.co.nz/many-models.html#simplifying-list-columns) you’ll learn how to convert list-columns back to regular atomic vectors (or sets of atomic vectors) so you can work with them more easily.
4. In [making tidy data with broom](http://r4ds.had.co.nz/many-models.html#making-tidy-data-with-broom), you’ll learn about the full set of tools provided by broom, and see how they can be applied to other types of data structure.

### 25.1.1 Prerequisites

Working with many models requires many of the packages of the tidyverse (for data exploration, wrangling, and programming) and modelr to facilitate modelling.

library(modelr)  
library(tidyverse)

## 25.2 gapminder

To motivate the power of many simple models, we’re going to look into the “gapminder” data. This data was popularised by Hans Rosling, a Swedish doctor and statistician. If you’ve never heard of him, stop reading this chapter right now and go watch one of his videos! He is a fantastic data presenter and illustrates how you can use data to present a compelling story. A good place to start is this short video filmed in conjunction with the BBC: <https://www.youtube.com/watch?v=jbkSRLYSojo>.

The gapminder data summarises the progression of countries over time, looking at statistics like life expectancy and GDP. The data is easy to access in R, thanks to Jenny Bryan who created the gapminder package:

# install.packages("gapminder")  
library(gapminder)

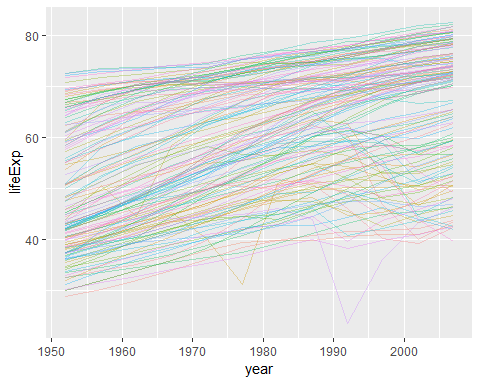
## Warning: package 'gapminder' was built under R version 3.5.1

gapminder

## # A tibble: 1,704 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.  
## 7 Afghanistan Asia 1982 39.9 12881816 978.  
## 8 Afghanistan Asia 1987 40.8 13867957 852.  
## 9 Afghanistan Asia 1992 41.7 16317921 649.  
## 10 Afghanistan Asia 1997 41.8 22227415 635.  
## # ... with 1,694 more rows

In this case study, we’re going to focus on just three variables to answer the questions “How does life expentancy (lifeExp) change over time(year) for each country(country)?” A good place to start is with a plot:

library(magrittr)  
gapminder %>%   
 ggplot(aes(x=year,y=lifeExp,group=country,colour=country))+  
 geom\_line(alpha=1/3,show.legend = F)

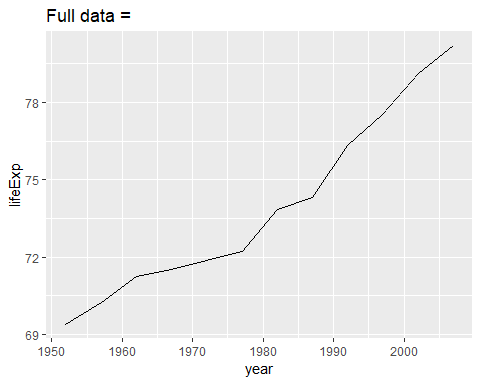


This is a small dataset: it only has ~1,700 observations and 3 variables. But it’s still hard to see what’s going on! Overall, it looks like life expectancy has been steadily improving. However, if you look closely, you might notice some countries that don’t follow this pattern. How can we make those countries easier to see?

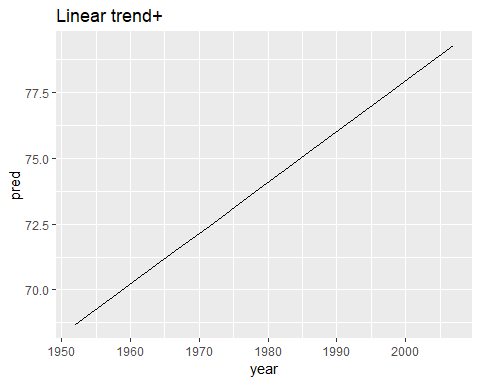
One way is to use the same approach as in the last chapter: there’s a strong signal (overall linear growth) that makes it hard to see subtler trends. We’ll tease these factors apart by fitting a model with a linear trend. The model captures steady growth over time, and the residuals will show what’s left.

You always know how to do that if we had a single country:

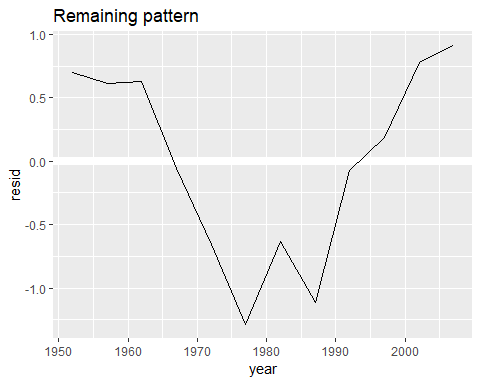
par(mfrow=c(1,3))   
  
nz <- filter(gapminder,country=="New Zealand")  
nz %>%   
 ggplot(aes(year,lifeExp))+  
 geom\_line()+  
 ggtitle("Full data =")



nz\_mod <- lm(lifeExp~year,data=nz)  
nz %>%  
 add\_predictions(nz\_mod) %>%   
 ggplot(aes(year,pred))+  
 geom\_line()+  
 ggtitle("Linear trend+")



nz %>%   
 add\_residuals(nz\_mod) %>%   
 ggplot(aes(year, resid)) +   
 geom\_hline(yintercept = 0, colour = "white", size = 3) +   
 geom\_line() +   
 ggtitle("Remaining pattern")



How can we easily fit that model to every country?

### 25.2.1 Nested data

You could imagine copy and pasting that code multiple times; but you’ve already learnt a better way! Extract out the common code with a function and repeat using a map function from purrr. This problem is structured a little differently to what you’ve seen before. Instead of repeating an action for each variable, we want to repeat an action for each country, a subset of rows. To do that, we need a new data structure: the **nested data frame**. To create a nested data frame we start with a grouped data frame, and “nest” it:

head(gapminder)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.

by\_country <- gapminder %>%   
 group\_by(country,continent) %>%   
 nest()  
  
by\_country

## # A tibble: 142 x 3  
## country continent data   
## <fct> <fct> <list>   
## 1 Afghanistan Asia <tibble [12 x 4]>  
## 2 Albania Europe <tibble [12 x 4]>  
## 3 Algeria Africa <tibble [12 x 4]>  
## 4 Angola Africa <tibble [12 x 4]>  
## 5 Argentina Americas <tibble [12 x 4]>  
## 6 Australia Oceania <tibble [12 x 4]>  
## 7 Austria Europe <tibble [12 x 4]>  
## 8 Bahrain Asia <tibble [12 x 4]>  
## 9 Bangladesh Asia <tibble [12 x 4]>  
## 10 Belgium Europe <tibble [12 x 4]>  
## # ... with 132 more rows

(I’m cheating a little by grouping on both continent and country. Given country, continent is fixed, so this doesn’t add any more groups, but it’s an easy way to carry an extra variable along for the ride.)

This creates a data frame that has one row per group (per country), and a rather unusual column: data. data is a list of data frames (or tibbles, to be precise). This seems like a crazy idea: we have a data frame with a column that is a list of other data frames! I’ll explain shortly why I think this is a good idea.

The data column is a little tricky to look at because it’s a moderately complicated list, and we’re still working on good tools to explore these objects. Unfortunately using str() is not recommended as it will often produce very long output. But if you pluck out a single element from the data column you’ll see that it contains all the data for that country (in this case, Afghanistan).

by\_country$data[[1]]

## # A tibble: 12 x 4  
## year lifeExp pop gdpPercap  
## <int> <dbl> <int> <dbl>  
## 1 1952 28.8 8425333 779.  
## 2 1957 30.3 9240934 821.  
## 3 1962 32.0 10267083 853.  
## 4 1967 34.0 11537966 836.  
## 5 1972 36.1 13079460 740.  
## 6 1977 38.4 14880372 786.  
## 7 1982 39.9 12881816 978.  
## 8 1987 40.8 13867957 852.  
## 9 1992 41.7 16317921 649.  
## 10 1997 41.8 22227415 635.  
## 11 2002 42.1 25268405 727.  
## 12 2007 43.8 31889923 975.

Note the difference between a standard grouped data frame and a neted data frame: in a grouped data frame, each row is an observation; in a nested data frame, each row is a group. Another way to think about a nested dataset is we now have a meta-observation: a row that represents the complete time course for a country, rather than a single point in time.

### 25.2.2 List-columns

Now that we have our nested data frame, we’re in a good position to fit some models. We have a model-fitting function:

country\_model <- function(df){  
 lm(lifeExp ~ year, data=df)  
}

And we want to apply it to every data frame. The data frames are in a list, so we can use purrr:map() to apply country\_model to each element:

models <- map(by\_country$data,country\_model)

However, rather than leaving the list of models as a free-floating object, I think it’s better to store it as a column in the by\_country data frame. Storing related objects in columns is a key part of the value of data frames, and why I think list-columns are such a good idea. In the course of working with these countries, we are going to have lots of lists where we have one element per country. So why not store them all together in one data frame?

In other words, instead of creating a new object in the global environment, we’re going to create a new variable in the by\_country data frame. That’s a job for dplyr::mutate():

by\_country <- by\_country %>%   
 mutate(model=map(data,country\_model))  
by\_country

## # A tibble: 142 x 4  
## country continent data model   
## <fct> <fct> <list> <list>   
## 1 Afghanistan Asia <tibble [12 x 4]> <S3: lm>  
## 2 Albania Europe <tibble [12 x 4]> <S3: lm>  
## 3 Algeria Africa <tibble [12 x 4]> <S3: lm>  
## 4 Angola Africa <tibble [12 x 4]> <S3: lm>  
## 5 Argentina Americas <tibble [12 x 4]> <S3: lm>  
## 6 Australia Oceania <tibble [12 x 4]> <S3: lm>  
## 7 Austria Europe <tibble [12 x 4]> <S3: lm>  
## 8 Bahrain Asia <tibble [12 x 4]> <S3: lm>  
## 9 Bangladesh Asia <tibble [12 x 4]> <S3: lm>  
## 10 Belgium Europe <tibble [12 x 4]> <S3: lm>  
## # ... with 132 more rows

# by\_country$data[[1]]  
by\_country$model[[1]]

##   
## Call:  
## lm(formula = lifeExp ~ year, data = df)  
##   
## Coefficients:  
## (Intercept) year   
## -507.5343 0.2753

This has a big advantage: because all the related objects are stored together, you don’t need to manually keep them in sync when you filter or arrange. The semantics of the data frame takes care of that for you:

by\_country %>%   
 filter(continent=="Europe")

## # A tibble: 30 x 4  
## country continent data model   
## <fct> <fct> <list> <list>   
## 1 Albania Europe <tibble [12 x 4]> <S3: lm>  
## 2 Austria Europe <tibble [12 x 4]> <S3: lm>  
## 3 Belgium Europe <tibble [12 x 4]> <S3: lm>  
## 4 Bosnia and Herzegovina Europe <tibble [12 x 4]> <S3: lm>  
## 5 Bulgaria Europe <tibble [12 x 4]> <S3: lm>  
## 6 Croatia Europe <tibble [12 x 4]> <S3: lm>  
## 7 Czech Republic Europe <tibble [12 x 4]> <S3: lm>  
## 8 Denmark Europe <tibble [12 x 4]> <S3: lm>  
## 9 Finland Europe <tibble [12 x 4]> <S3: lm>  
## 10 France Europe <tibble [12 x 4]> <S3: lm>  
## # ... with 20 more rows

by\_country %>%   
 arrange(continent,country)

## # A tibble: 142 x 4  
## country continent data model   
## <fct> <fct> <list> <list>   
## 1 Algeria Africa <tibble [12 x 4]> <S3: lm>  
## 2 Angola Africa <tibble [12 x 4]> <S3: lm>  
## 3 Benin Africa <tibble [12 x 4]> <S3: lm>  
## 4 Botswana Africa <tibble [12 x 4]> <S3: lm>  
## 5 Burkina Faso Africa <tibble [12 x 4]> <S3: lm>  
## 6 Burundi Africa <tibble [12 x 4]> <S3: lm>  
## 7 Cameroon Africa <tibble [12 x 4]> <S3: lm>  
## 8 Central African Republic Africa <tibble [12 x 4]> <S3: lm>  
## 9 Chad Africa <tibble [12 x 4]> <S3: lm>  
## 10 Comoros Africa <tibble [12 x 4]> <S3: lm>  
## # ... with 132 more rows

If your list of data frames and list of models were separate objects, you have to remember that whenever you re-order or subset one vector, you need to re-order or subset all the others in order to keep them in sync. If you forget, your code will continue to work, but it will give the wrong answer!

### 25.2.3 Unnesting

Previously we computed the residuals of a single model with a single dataset. Now we have 142 data frames and 142 models. To compute the residuals, we need to call add\_residuals() with each model-data pair:

library(modelr)  
  
by\_country <- by\_country %>%   
 mutate(  
 resids = map2(data, model, add\_residuals)  
 )  
by\_country

## # A tibble: 142 x 5  
## country continent data model resids   
## <fct> <fct> <list> <list> <list>   
## 1 Afghanistan Asia <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 2 Albania Europe <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 3 Algeria Africa <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 4 Angola Africa <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 5 Argentina Americas <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 6 Australia Oceania <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 7 Austria Europe <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 8 Bahrain Asia <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 9 Bangladesh Asia <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## 10 Belgium Europe <tibble [12 x 4]> <S3: lm> <tibble [12 x 5]>  
## # ... with 132 more rows

But how you can plot a list of data frames? Instead of struggling to answer that question, let’s turn the list of data frames back into a regular data frame. Previously we used nest() to turn a regular data frame into an nested data frame, and now we do the opposite with unnest():

resids <- unnest(by\_country,resids)  
resids

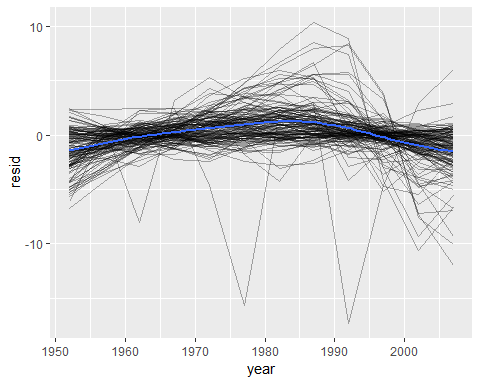
## # A tibble: 1,704 x 7  
## country continent year lifeExp pop gdpPercap resid  
## <fct> <fct> <int> <dbl> <int> <dbl> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779. -1.11   
## 2 Afghanistan Asia 1957 30.3 9240934 821. -0.952   
## 3 Afghanistan Asia 1962 32.0 10267083 853. -0.664   
## 4 Afghanistan Asia 1967 34.0 11537966 836. -0.0172  
## 5 Afghanistan Asia 1972 36.1 13079460 740. 0.674   
## 6 Afghanistan Asia 1977 38.4 14880372 786. 1.65   
## 7 Afghanistan Asia 1982 39.9 12881816 978. 1.69   
## 8 Afghanistan Asia 1987 40.8 13867957 852. 1.28   
## 9 Afghanistan Asia 1992 41.7 16317921 649. 0.754   
## 10 Afghanistan Asia 1997 41.8 22227415 635. -0.534   
## # ... with 1,694 more rows

Note that each regular column is repeated one for each row in the nested column.

Now we have regular data frame, we can plot the residuals:

resids %>%   
 ggplot(aes(year,resid))+  
 geom\_line(aes(group=country),alpha=1/3)+  
 geom\_smooth(se=FALSE)

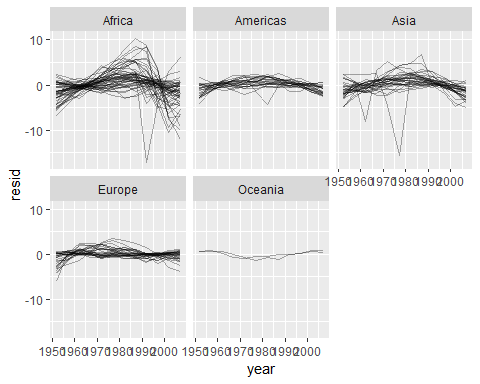
## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



# > `geom\_smooth()` using method=`gam` and formula `y~s(x,bs="cs")`

Facetting by countinent is particularly revealing:

resids %>%  
 ggplot(aes(year,resid,group=country))+  
 geom\_line(alpha=1/3)+  
 facet\_wrap(~continent)



It looks like we’ve missed some patterns. There’s also something intersting going on in Africa: we see some very large residuals which suggests our model isn’t fitting so well there. We’ll explore that more in the next section, attacking it from a slightly different angle.

### 25.2.4 Model quality

Instead of looking at the residuals from the model, we could look at some general measurements of model quality. You learned how to compute some specific measures in the previous chapter. Here we’ll show a different approach using the broom package. The broom package provides a general set of functions to turn models into tidy data. Here we’ll use broom::glance() to extract some model quality metrics. If we apply it to a model, we get a data frame with a single row:

broom::glance(nz\_mod)

## r.squared adj.r.squared sigma statistic p.value df logLik  
## 1 0.9535846 0.9489431 0.8043472 205.4459 5.407324e-08 2 -13.32064  
## AIC BIC deviance df.residual  
## 1 32.64128 34.096 6.469743 10

We can use mutate() and unnest() to create a data frame with a row for each country:

by\_country %>%   
 mutate(glance=map(model,broom::glance)) %>%   
 unnest(glance)

## # A tibble: 142 x 16  
## country continent data model resids r.squared adj.r.squared sigma  
## <fct> <fct> <lis> <lis> <list> <dbl> <dbl> <dbl>  
## 1 Afghan~ Asia <tib~ <S3:~ <tibb~ 0.948 0.942 1.22   
## 2 Albania Europe <tib~ <S3:~ <tibb~ 0.911 0.902 1.98   
## 3 Algeria Africa <tib~ <S3:~ <tibb~ 0.985 0.984 1.32   
## 4 Angola Africa <tib~ <S3:~ <tibb~ 0.888 0.877 1.41   
## 5 Argent~ Americas <tib~ <S3:~ <tibb~ 0.996 0.995 0.292  
## 6 Austra~ Oceania <tib~ <S3:~ <tibb~ 0.980 0.978 0.621  
## 7 Austria Europe <tib~ <S3:~ <tibb~ 0.992 0.991 0.407  
## 8 Bahrain Asia <tib~ <S3:~ <tibb~ 0.967 0.963 1.64   
## 9 Bangla~ Asia <tib~ <S3:~ <tibb~ 0.989 0.988 0.977  
## 10 Belgium Europe <tib~ <S3:~ <tibb~ 0.995 0.994 0.293  
## # ... with 132 more rows, and 8 more variables: statistic <dbl>,  
## # p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>, BIC <dbl>,  
## # deviance <dbl>, df.residual <int>

#> # A tibble: 142 x 16  
#> country continent data model resids r.squared adj.r.squared sigma  
#> <fct> <fct> <list> <lis> <list> <dbl> <dbl> <dbl>  
#> 1 Afghani… Asia <tibble… <S3:… <tibble… 0.948 0.942 1.22   
#> 2 Albania Europe <tibble… <S3:… <tibble… 0.911 0.902 1.98   
#> 3 Algeria Africa <tibble… <S3:… <tibble… 0.985 0.984 1.32   
#> 4 Angola Africa <tibble… <S3:… <tibble… 0.888 0.877 1.41   
#> 5 Argenti… Americas <tibble… <S3:… <tibble… 0.996 0.995 0.292  
#> 6 Austral… Oceania <tibble… <S3:… <tibble… 0.980 0.978 0.621  
#> # ... with 136 more rows, and 8 more variables: statistic <dbl>,  
#> # p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>, BIC <dbl>,  
#> # deviance <dbl>, df.residual <int>

This isn’t quite the output we want, because it still includes all the list columns. This is default behaviour when unnest() works on single row data frames. To suppress these columns we use .drop = TRUE:

glance <- by\_country %>%   
 mutate(glance=map(model,broom::glance)) %>%   
 unnest(glance,drop=T)  
glance

## # A tibble: 142 x 17  
## country continent data model resids drop r.squared adj.r.squared sigma  
## <fct> <fct> <lis> <lis> <list> <lgl> <dbl> <dbl> <dbl>  
## 1 Afghan~ Asia <tib~ <S3:~ <tibb~ TRUE 0.948 0.942 1.22   
## 2 Albania Europe <tib~ <S3:~ <tibb~ TRUE 0.911 0.902 1.98   
## 3 Algeria Africa <tib~ <S3:~ <tibb~ TRUE 0.985 0.984 1.32   
## 4 Angola Africa <tib~ <S3:~ <tibb~ TRUE 0.888 0.877 1.41   
## 5 Argent~ Americas <tib~ <S3:~ <tibb~ TRUE 0.996 0.995 0.292  
## 6 Austra~ Oceania <tib~ <S3:~ <tibb~ TRUE 0.980 0.978 0.621  
## 7 Austria Europe <tib~ <S3:~ <tibb~ TRUE 0.992 0.991 0.407  
## 8 Bahrain Asia <tib~ <S3:~ <tibb~ TRUE 0.967 0.963 1.64   
## 9 Bangla~ Asia <tib~ <S3:~ <tibb~ TRUE 0.989 0.988 0.977  
## 10 Belgium Europe <tib~ <S3:~ <tibb~ TRUE 0.995 0.994 0.293  
## # ... with 132 more rows, and 8 more variables: statistic <dbl>,  
## # p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>, BIC <dbl>,  
## # deviance <dbl>, df.residual <int>

(Pay attention to the variables that aren’t printed: there’s a lot of useful stuff there.)

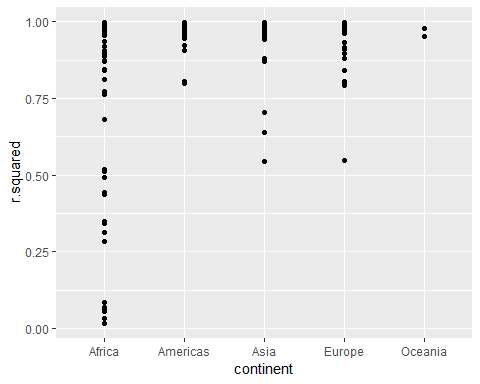
With this data frame in hand, we can start to look for models that don’t fit well:

glance %>%   
 arrange(r.squared)

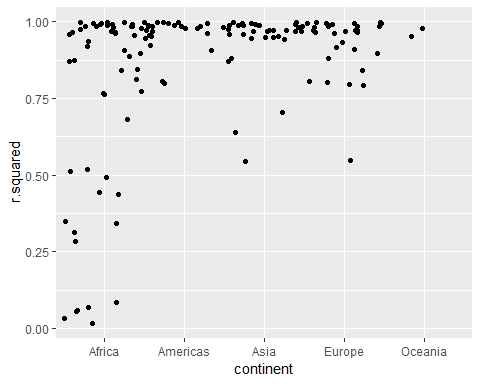
## # A tibble: 142 x 17  
## country continent data model resids drop r.squared adj.r.squared sigma  
## <fct> <fct> <lis> <lis> <list> <lgl> <dbl> <dbl> <dbl>  
## 1 Rwanda Africa <tib~ <S3:~ <tibb~ TRUE 0.0172 -0.0811 6.56  
## 2 Botswa~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0340 -0.0626 6.11  
## 3 Zimbab~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0562 -0.0381 7.21  
## 4 Zambia Africa <tib~ <S3:~ <tibb~ TRUE 0.0598 -0.0342 4.53  
## 5 Swazil~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0682 -0.0250 6.64  
## 6 Lesotho Africa <tib~ <S3:~ <tibb~ TRUE 0.0849 -0.00666 5.93  
## 7 Cote d~ Africa <tib~ <S3:~ <tibb~ TRUE 0.283 0.212 3.93  
## 8 South ~ Africa <tib~ <S3:~ <tibb~ TRUE 0.312 0.244 4.74  
## 9 Uganda Africa <tib~ <S3:~ <tibb~ TRUE 0.342 0.276 3.19  
## 10 Congo,~ Africa <tib~ <S3:~ <tibb~ TRUE 0.348 0.283 2.43  
## # ... with 132 more rows, and 8 more variables: statistic <dbl>,  
## # p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>, BIC <dbl>,  
## # deviance <dbl>, df.residual <int>

The worst models all appear to be in Africa. Let’s double check that with a plot. Here we have a relatively small number of observations and a discrete variable, so geom\_jitter() is effective:

glance %>%   
 ggplot(aes(continent,r.squared))+  
 geom\_point()



glance %>%   
 ggplot(aes(continent,r.squared))+  
 geom\_jitter(width=0.5)



We could pull out the countries with particularly bad and plot the data:

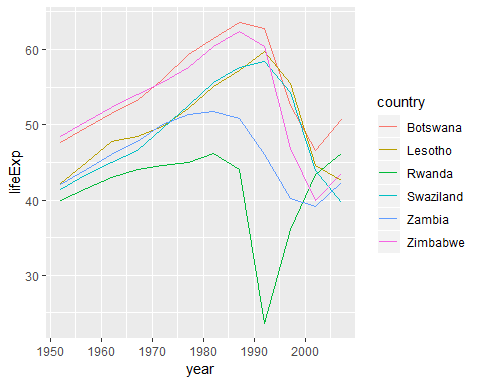
library(gapminder)  
bad\_fit <- filter(glance,r.squared<0.25)  
bad\_fit

## # A tibble: 6 x 17  
## country continent data model resids drop r.squared adj.r.squared sigma  
## <fct> <fct> <lis> <lis> <list> <lgl> <dbl> <dbl> <dbl>  
## 1 Botswa~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0340 -0.0626 6.11  
## 2 Lesotho Africa <tib~ <S3:~ <tibb~ TRUE 0.0849 -0.00666 5.93  
## 3 Rwanda Africa <tib~ <S3:~ <tibb~ TRUE 0.0172 -0.0811 6.56  
## 4 Swazil~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0682 -0.0250 6.64  
## 5 Zambia Africa <tib~ <S3:~ <tibb~ TRUE 0.0598 -0.0342 4.53  
## 6 Zimbab~ Africa <tib~ <S3:~ <tibb~ TRUE 0.0562 -0.0381 7.21  
## # ... with 8 more variables: statistic <dbl>, p.value <dbl>, df <int>,  
## # logLik <dbl>, AIC <dbl>, BIC <dbl>, deviance <dbl>, df.residual <int>

head(gapminder)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.

gapminder %>%   
 semi\_join(bad\_fit,by="country") %>%   
 ggplot(aes(year,lifeExp,colour=country))+  
 geom\_line()



We see two main effects here: the tragedies of the HIV/AIDS epidemic and the Rwandan genocide.

### 25.2.5 Exercises

1. A linear trend seems to be slightly too simple for the overall trend. Can you do better with a quadratic polynomial? How can you interpret the coefficients of the quadratic? (Hint you might want to transform year so that it has mean zero.)
2. Explore other methods for visualising the distribution of  
    per continent. You might want to try the ggbeeswarm package, which provides similar methods for avoiding overlaps as jitter, but uses deterministic methods.
3. To create the last plot (showing the data for the countries with the worst model fits), we needed two steps: we created a data frame with one row per country and then semi-joined it to the original dataset. It’s possible to avoid this join if we use unnest() instead of unnest(.drop = TRUE). How?

## 25.3 List-columns

Now that you’ve seen a basic workflow for managing many models, let’s dive back into some of the details. In this section, we’ll explore the list-column data structure in a little more detail. It’s only recently that I’ve really appreciated the idea of the list-column. List-columns are implicit in the definition of the data frame: a data frame is a named list of equal length vectors. A list is a vector, so it’s always been legitimate to use a list as a column of a data frame. However, base R doesn’t make it easy to create list-columns, and data.frame() treats a list as a list of columns:.

data.frame(x=list(1:3,3:5))

## x.1.3 x.3.5  
## 1 1 3  
## 2 2 4  
## 3 3 5

You can prevent data.frame() from doing this with I(), but the results doesn’t print particulrly well:

data.frame(  
 x = I(list(1:3, 3:5)),   
 y = c("1, 2", "3, 4, 5")  
)

## x y  
## 1 1, 2, 3 1, 2  
## 2 3, 4, 5 3, 4, 5

#> x y  
#> 1 1, 2, 3 1, 2  
#> 2 3, 4, 5 3, 4, 5

Tibble alleviates this problem by being lazier (tibble()doesn’t modify its inpus) and by providing a better print method:

tibble(  
 x=list(1:3,3:5),  
 y=c("1,2","3,4,5")  
)

## # A tibble: 2 x 2  
## x y   
## <list> <chr>  
## 1 <int [3]> 1,2   
## 2 <int [3]> 3,4,5

#> # A tibble: 2 x 2  
#> x y   
#> <list> <chr>   
#> 1 <int [3]> 1, 2   
#> 2 <int [3]> 3, 4, 5

It’s even easier with tribble() as it can automatically work out that you need a list:

trible <- tribble(  
 ~x,~y,  
 1:3,"1.2",  
 3:5,"3,4,5"  
)  
trible$x[[1]]

## [1] 1 2 3

List-columns are often most useful as intermediate data structure. They’re hard to work with directly, because most R functions work with atomic vectors or data frames, but the advantage of keeping related items together in a data frame is worth a little hassle.

Generally, there are three parts of an effective list-column pipeline: 1. You create the list-column using one of nest(), summarise()+list() or mutate() + a map function, as described in Creating list-columns. 2. You create other intermediate list-columns by transforming existing list columns with map(), map2() or pmap(). For example, in the case study above, we created a list-column of models by transforming a list-column of data frames. 3. You simplify the list-column back down to a data frame or atomic vector, as described in Simplifying list-columns.

## 25.4 Creating list-columns

Typically, you won’t create list-columns with tibble(). Instead, you’ll create them from regular columns, using one of three methods.

1. With tidyr::nest() to convert a grouped data frame into a nested data frame where you have list-column of data frames.
2. With mutate() and vectorised functions that return a list.
3. With summarise() and summary functions that return multiple results.

Alternatively, you might create them from a named list, using tibble::enframe().

Generally, when creating list-columns, you should make sure they’re homogeneous: each element should contain the same type of thing. There are no checks to make sure this is true, but if you use purrr and remember what you’ve learned about type-stable functions, you should find it happens naturally.

### 25.4.1 With nesting

nest() creates a nested data frame, which is a data frame with a list-column of data frames.In a nested data frame each row is a meta-observation: the other columns give variables that define the observation (like country and continent above), and the list-column of data frames gives the individual observations that make up the meta-observation.

gapminder %>%   
 group\_by(country,continent) %>%   
 nest()

## # A tibble: 142 x 3  
## country continent data   
## <fct> <fct> <list>   
## 1 Afghanistan Asia <tibble [12 x 4]>  
## 2 Albania Europe <tibble [12 x 4]>  
## 3 Algeria Africa <tibble [12 x 4]>  
## 4 Angola Africa <tibble [12 x 4]>  
## 5 Argentina Americas <tibble [12 x 4]>  
## 6 Australia Oceania <tibble [12 x 4]>  
## 7 Austria Europe <tibble [12 x 4]>  
## 8 Bahrain Asia <tibble [12 x 4]>  
## 9 Bangladesh Asia <tibble [12 x 4]>  
## 10 Belgium Europe <tibble [12 x 4]>  
## # ... with 132 more rows

You can also use it on an ungrouped data frame, specifying which columns you want to nest:

gapminder %>%   
 names()

## [1] "country" "continent" "year" "lifeExp" "pop" "gdpPercap"

gapminder %>%   
 nest(year:gdpPercap)

## # A tibble: 142 x 3  
## country continent data   
## <fct> <fct> <list>   
## 1 Afghanistan Asia <tibble [12 x 4]>  
## 2 Albania Europe <tibble [12 x 4]>  
## 3 Algeria Africa <tibble [12 x 4]>  
## 4 Angola Africa <tibble [12 x 4]>  
## 5 Argentina Americas <tibble [12 x 4]>  
## 6 Australia Oceania <tibble [12 x 4]>  
## 7 Austria Europe <tibble [12 x 4]>  
## 8 Bahrain Asia <tibble [12 x 4]>  
## 9 Bangladesh Asia <tibble [12 x 4]>  
## 10 Belgium Europe <tibble [12 x 4]>  
## # ... with 132 more rows

### 25.4.2 From vectorised functions

Some useful functions take an atomic vector and return a list. For example, in strings you learned about stringr::str\_split() which takes a character vector and returns a list of character vectors. If you use that inside mutate, you’ll get a list-column:

df <- tribble(  
 ~x1,  
 "a,b,c",  
 "d,e,f,g"  
)  
  
df %>%   
 mutate(x2=stringr::str\_split(x1,","))

## # A tibble: 2 x 2  
## x1 x2   
## <chr> <list>   
## 1 a,b,c <chr [3]>  
## 2 d,e,f,g <chr [4]>

unnest() knows how to handle these lists of vectors:

df %>%   
 mutate(x2=stringr::str\_split(x1,",")) %>%   
 unnest()

## # A tibble: 7 x 2  
## x1 x2   
## <chr> <chr>  
## 1 a,b,c a   
## 2 a,b,c b   
## 3 a,b,c c   
## 4 d,e,f,g d   
## 5 d,e,f,g e   
## 6 d,e,f,g f   
## 7 d,e,f,g g

(If you find yourself using this pattern a lot, make sure to check out tidyr::separate\_rows() which is a wrapper around this common pattern).

Another example of this pattern is using the map(), map2(), pmap() from purrr. For example, we could take the final example from Invoking different functions and rewrite it to use mutate():

sim <- tribble(  
 ~f, ~params,  
 "runif", list(min = -1, max = 1),  
 "rnorm", list(sd = 5),  
 "rpois", list(lambda = 10)  
)  
  
sim %>%  
 mutate(sims = invoke\_map(f, params, n = 10))

## # A tibble: 3 x 3  
## f params sims   
## <chr> <list> <list>   
## 1 runif <list [2]> <dbl [10]>  
## 2 rnorm <list [1]> <dbl [10]>  
## 3 rpois <list [1]> <int [10]>

Note that technically sim isn’t homogeneous because it contains both double and integer vectors. However, this is unlikely to cause many problems since integers and doubles are both numeric vectors.

### 25.4.3 From multivalued summaries

One restriction of summarise() is that it only works with summary functions that return a single value. That means that you can’t use it with functions like quantile() that return a vector of arbitrary length:

mtcars %>%   
 group\_by(cyl) %>%   
 summarise(q=quantile(mpg))  
#> Error in summarise\_impl(.data, dots): Column `q` must be length 1 (a summary value), not 5

You can, however, wrap the results in a list! This obeys the contract of summarise(), because each summary is now a list (a vector) of length 1.

mtcars1 <- mtcars %>%   
 group\_by(cyl) %>%   
 summarise(q=list(quantile(mpg)))  
  
mtcars1

## # A tibble: 3 x 2  
## cyl q   
## <dbl> <list>   
## 1 4 <dbl [5]>  
## 2 6 <dbl [5]>  
## 3 8 <dbl [5]>

mtcars1$q[[1]]

## 0% 25% 50% 75% 100%   
## 21.4 22.8 26.0 30.4 33.9

To make useful results with **unnest**, you’ll also need to capture the probabilities:

probs <- c(0.01,0.25,0.5,0.75,0.99)  
mtcars %>%   
 group\_by(cyl) %>%   
 summarise(p=list(probs),q=list(quantile(mpg,probs))) %>%   
 unnest()

## # A tibble: 15 x 3  
## cyl p q  
## <dbl> <dbl> <dbl>  
## 1 4 0.01 21.4  
## 2 4 0.25 22.8  
## 3 4 0.5 26   
## 4 4 0.75 30.4  
## 5 4 0.99 33.8  
## 6 6 0.01 17.8  
## 7 6 0.25 18.6  
## 8 6 0.5 19.7  
## 9 6 0.75 21   
## 10 6 0.99 21.4  
## 11 8 0.01 10.4  
## 12 8 0.25 14.4  
## 13 8 0.5 15.2  
## 14 8 0.75 16.2  
## 15 8 0.99 19.1

# filter(cyl==4)

### 25.4.4 From a named list

Data frames with list-columns provide a solution to a common problem: what do you do if you want to iterate over both the contents of a list and its elements? Instead of trying to jam everything into one object, it’s often easier to make a data frame: one column can contain the elements, and one column can contain the list. An easy way to create such a data frame from a list is tibble::enframe().

x <- list(  
 a=1:5,  
 b=3:4,  
 c=5:6  
)  
  
df <- enframe(x)  
df

## # A tibble: 3 x 2  
## name value   
## <chr> <list>   
## 1 a <int [5]>  
## 2 b <int [2]>  
## 3 c <int [2]>

The advantage of this structure is that it generalises in a straightforward way - names are useful if you have character vector of metadata, but don’t help if you have other types of data, or multiple vectors.

Now if you want to iterate over names and values in parallel, you can use map2():

df %>%   
 mutate(  
 smry = map2\_chr(name, value, ~ stringr::str\_c(.x, ": ", .y[1]))  
 )

## # A tibble: 3 x 3  
## name value smry   
## <chr> <list> <chr>  
## 1 a <int [5]> a: 1   
## 2 b <int [2]> b: 3   
## 3 c <int [2]> c: 5

### 25.4.5 Exercises

1. List all the functions that you can think of that take a atomic vector and return a list.
2. Brainstorm useful summary functions that, like quantile(), return multiple values.
3. What’s missing in the following data frame? How does quantile() return that missing piece? Why isn’t that helpful here?
4. What does this code do? Why might might it be useful?

mtcars %>%   
 group\_by(cyl) %>%   
 summarise\_each(funs(list))

## `summarise\_each()` is deprecated.  
## Use `summarise\_all()`, `summarise\_at()` or `summarise\_if()` instead.  
## To map `funs` over all variables, use `summarise\_all()`

## # A tibble: 3 x 11  
## cyl mpg disp hp drat wt qsec vs am gear carb   
## <dbl> <list> <list> <list> <list> <list> <list> <list> <lis> <lis> <lis>  
## 1 4 <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl~ <dbl~ <dbl~  
## 2 6 <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl~ <dbl~ <dbl~  
## 3 8 <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl ~ <dbl~ <dbl~ <dbl~

## 25.5 Simplifying list-columns

To apply the techniques of data manipulation and visualisation you’ve learned in this book, you’ll need to simplify the list-column back to a regular column (an atomic vector), or set of columns. The technique you’ll use to collapse back down to a simpler structure depends on whether you want a single value per element, or multiple values:

1. If you want a single value, use mutate() with map\_lgl(), map\_int(), map\_dbl(), and map\_chr() to create an atomic vector.
2. If you want many values, use unnest() to convert list-columns back to regular columns, repeating the rows as many times as necessary.

These are described in more detail below.

### 25.5.1 List to vector

If you can reduce your list column to an atomic vector then it will be a regular column. For example, you can always summarise an object with its type and length, so this code will work regardless of what sort of list-column you have:

df <- tribble(  
 ~x,  
 letters[1:5],  
 1:3,  
 runif(5)  
)  
  
df %>%   
 mutate(  
 type=map\_chr(x,typeof),  
 length=map\_int(x,length)  
 )

## # A tibble: 3 x 3  
## x type length  
## <list> <chr> <int>  
## 1 <chr [5]> character 5  
## 2 <int [3]> integer 3  
## 3 <dbl [5]> double 5

This is the same basic information that you get from the default tbl print method, but now you can use it for filtering. This is a useful technique if you have a heterogeneous list, and want to filter out the parts aren’t working for you.

Don’t forget about the map\_\*() shortcuts - you can use map\_chr(x, "apple") to extract the string stored in apple for each element of x. This is useful for pulling apart nested lists into regular columns. Use the .null argument to provide a value to use if the element is missing (instead of returning NULL):

library(tidyverse)  
  
df <- tribble(  
 ~x,  
 list(a=1,b=2),  
 list(a=2,c=4)  
)  
df

## # A tibble: 2 x 1  
## x   
## <list>   
## 1 <list [2]>  
## 2 <list [2]>

df %>%   
 mutate(  
 a=map\_dbl(x,"a"),  
 b=map\_dbl(x,"b",.null=NA\_real\_)  
 )

## # A tibble: 2 x 3  
## x a b  
## <list> <dbl> <dbl>  
## 1 <list [2]> 1 2  
## 2 <list [2]> 2 NA

### 25.5.2 Unnesting

unnest() works by repeating the regular columns once for each element of the list-column. For example, in the following very simple example, we repeat the first row 4 times (because there the first element of y has length four). and the second row once:

tibble(x=1:2,y=list(1:4,1))

## # A tibble: 2 x 2  
## x y   
## <int> <list>   
## 1 1 <int [4]>  
## 2 2 <dbl [1]>

tibble(x=1:2,y=list(1:4,1)) %>%   
 unnest(y)

## # A tibble: 5 x 2  
## x y  
## <int> <dbl>  
## 1 1 1  
## 2 1 2  
## 3 1 3  
## 4 1 4  
## 5 2 1

This means you can’t simutaneously unnest two columns that contain different number of elements:

df1 <- tribble(  
 ~x,~y,~z,  
 1,c("a","b"),1:2,  
 2,"c",3  
)  
  
df1

## # A tibble: 2 x 3  
## x y z   
## <dbl> <list> <list>   
## 1 1 <chr [2]> <int [2]>  
## 2 2 <chr [1]> <dbl [1]>

df1 %>% unnest(y,z)

## # A tibble: 3 x 3  
## x y z  
## <dbl> <chr> <dbl>  
## 1 1 a 1  
## 2 1 b 2  
## 3 2 c 3

# doesn't work because y and z have different number of elements  
df2 <- tribble(  
 ~x,~y,~z,  
 1,"a",1:2,  
 2,c("b","c"),3  
)  
  
df2  
df2 %>% unnest(y,z)  
#> Error: All nested columns must have the same number of elements.

The same principle applies when unnesting list-columns of data frames. You can unnest multiple list-cols as long as all the data frames in each row have the same number of rows.

## 25.6 Making tidy data with broom

The broom package provides three general tools for turning models into tidy data frames:

1. broom::glance(model) returns a row for each model. Each column gives a model summary: either a measure of model quality, or complexity, or a combination of the two.
2. broom::tidy(model) returns a row for each coefficient in the model. Each column gives information about the estimate or its variability.
3. broom::augment(model, data) returns a row for each row in data, adding extra values like residuals, and influence statistics.