PairProgramming Week 7

HDS

2024-09-03

checked 01/03/2025

Pair Programming, Module 08, Permutation, Monte Carlo, Cross-validation and Bayesian Methods, DSE5001

Name: Ryan Waterman Partner Name:

These are tactics which may be used for

-hypothesis testing

-estimating confidence intervals

particularly in situations where we don't have a good statistical model to use

#Permutation Tests

Permutation test are used to test hypotheses of differences between groups, or between variables We can create permutation versions of t-tests, F-test, R^2 values etc

#Permutation t-test

use the sleep data set as an example

```
data(sleep)
head(sleep)
```

```
##   extra group ID
## 1   0.7     1  1
## 2  -1.6     1  2
## 3  -0.2     1  3
## 4  -1.2     1  4
## 5  -0.1     1  5
## 6   3.4     1  6
```

This data set is used as an example for the t-test function in R,

```
t.test(extra~group, data=sleep)
```

```
##
##  Welch Two Sample t-test
##
## data:  extra by group
## t = -1.8608, df = 17.776, p-value = 0.07939
## alternative hypothesis: true difference in means between group 1 and group 2 is not equal to
0
## 95 percent confidence interval:
##  -3.3654832  0.2054832
## sample estimates:
## mean in group 1 mean in group 2
##            0.75            2.33
```

Looking at the results, group 1 has a mean of 0.75, group 2 has a mean of 2.33, the t value is -1.86

p-value is 0.07939, which is not significant

So we know the answer, but let's try this using a permutation test, just to see how permutation tests work

Stating the Null

We have an observed difference in the means of |0.75-2.33| = 1.58

If there is no difference between the groups, we could re-assign specimens randomly between the two groups, literally shuffling the factor identifications of specimens, to create a permutation set. We could calculate the absolute value difference in the means of the shuffled data.

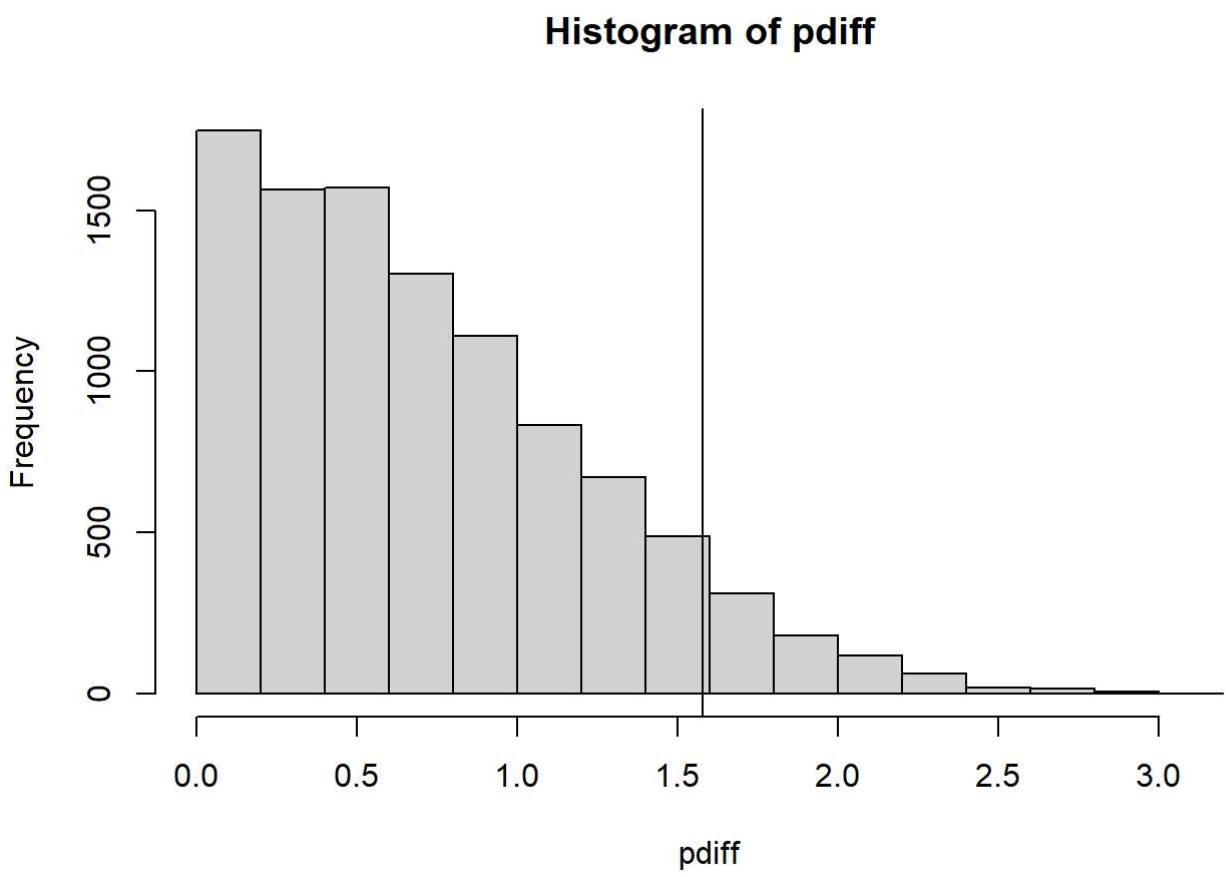
If we did this many times, say 10,000 times, then we would have a distribution of differences generated by the null hypotheis, and we can compare our observed difference of 1.58 to the distribution from the permutations, and estimate a probability that the permutation method could generate a value as large as 1.58 by random shuffling.

This is a *non-parametric* simulation, since we do not estimate any parameters to create the simulation

```
# library with permute function

library("gtools")
```

```
## Warning: package 'gtools' was built under R version 4.4.3
```

```
Nperms=10000

# create an empty vector of permuation differences
pdiff=rep(0,Nperms)

#generate 10,000 permutations

extra=sleep$extra
group=sleep$group

for(i in 1:Nperms)
{
  # r has a built in permuation, we will shuffle the values of extra
  perm_extra=permute(extra)
  diffvalue= abs( mean(perm_extra[group==1])-mean(perm_extra[group==2]))
  pdiff[i]=diffvalue

}


#create a histogram of the distribution under the null of the permutation test

hist(pdiff,breaks=20)


# our observed difference in the mean was 1.58, show that on the plot
abline(v=1.58)
```



Just looking at the histogram and the plot showing where a difference of 1.58 is on the plot, it looks like the null hypothesis easily produces differences this large

How many pdiff values are greater than 1.58?

```
sum(pdiff>1.58)
```

```
## [1] 766
```

We have 744 permutation differences of greater than 1.58 in 10,000 trials {Note: this is a random process, your answer should differ slightly from the 744 I got}

plus the 1 original value out of 1 trial, we can estimate the permutation p as

(744+1)/(10000+1)

```
(744+1)/(10000+1)
```

```
## [1] 0.07449255
```

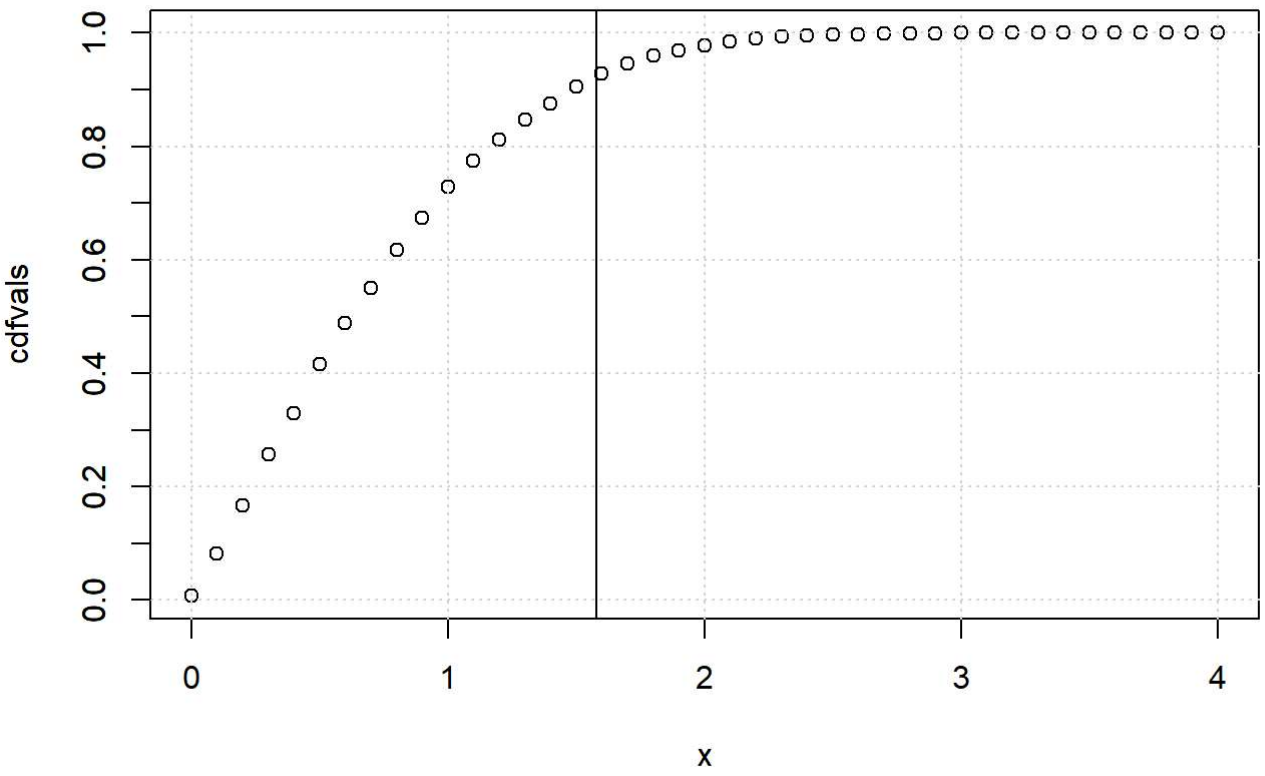
The permutation p when I ran this code was 0.0745, the analytic value from the t-test was 0.0793

You will probably get a slightly different p value due to the random nature of the test

empirical cummulative distribution

We can get a empirical estimate of the cummulative distribution as well

```
# create the empirical cdf function
perm_cdf=ecdf(pdiff)


# set up to plot
x=seq(0,4,0.1)

cdfvals=perm_cdf(x)

plot(x,cdfvals)
axis(2,at=c(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0))

# our observed difference in the mean was 1.58, show that on the plot
abline(v=1.58)

grid()
```



Again, we can see that the permutation has substantial numbers of results above 1.58.

Monte Carlo

This approach uses the random number generator in R to simulate an event, with multiple components

Suppose we have the following. My commute to work takes 12 minutes under good traffic conditions, but…

about 1 day in 30, there seems to be a detour. If there is a detour add a random time of 2 to 5 minutes to the time

There is one bad turn onto route 114, if there are cars in front of me, add a random normal(mean= 0.75 minute,sd=1/4 minute) time per car in front of me

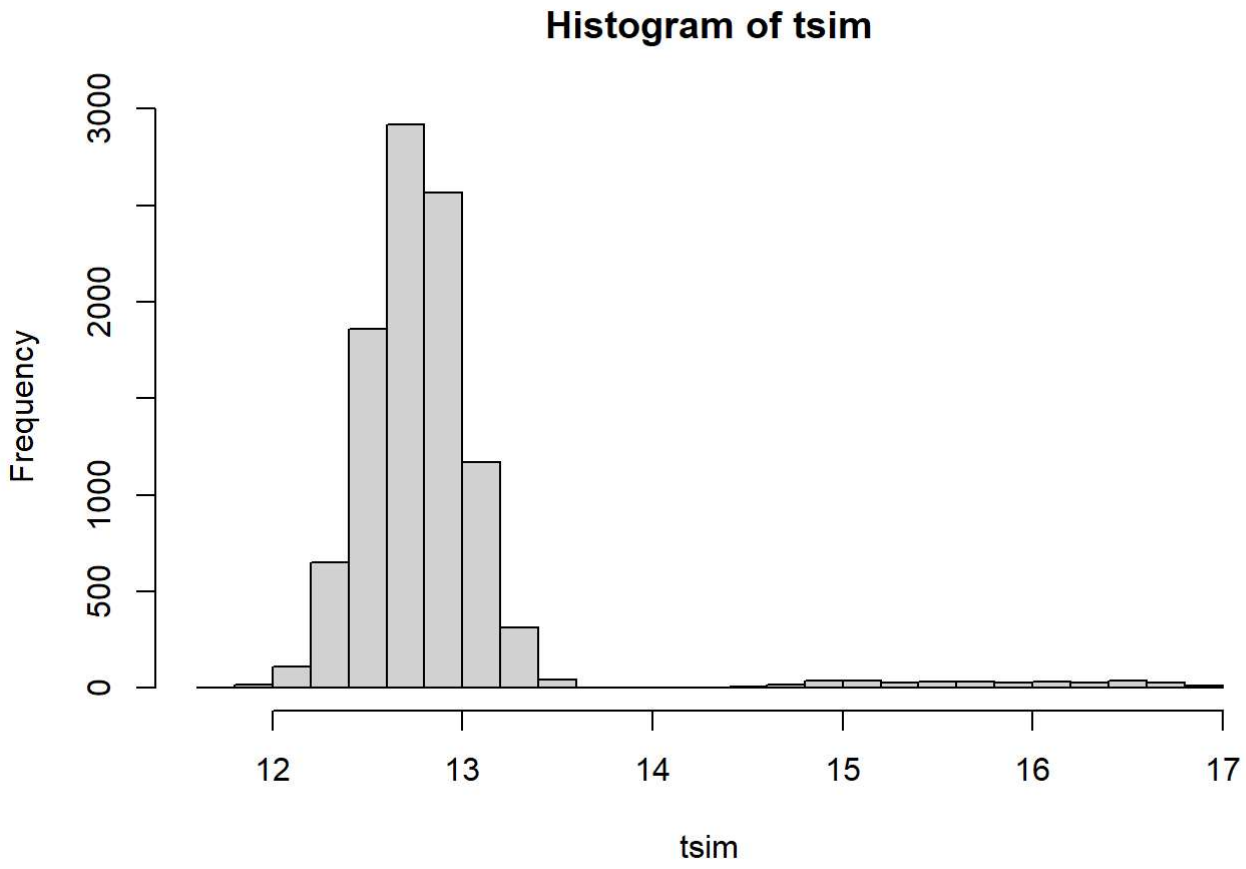
It seems like at that intersection, the number of cars in front of me is a poison distribution with a mean of about 2

Given all this, what is the distribution of my drive time to work? What is the mean? the std deviation and the 95% upper bound

We will create nsim=10,000 simulations of this to generate a distribution

This is a *parametric simulation* because we are estimating the parameters used in the simulation

```
Nsims=10000

# create an empty vector of permuation differences
tsim=rep(0,Nperms)


# ideal drive time without issues
tmin=12

#generate 10,000 simulation


for(i in 1:Nsims)
{
   t=tmin

   # simulate detour
   p=runif(1)
   #if p<1/30, add a random unif time between 2 and 4 minutes
   if(p<1/30)
   {
     t=t+runif(1,min=2,max=4)
   }

   #cars at the turn onto rt 114, a poison with lambda=3

   ncars=rpois(1,3)
   #for each car, add a random normal time
   for(j in ncars)
   {
     t=t+rnorm(1,mean=0.75, sd=0.25)
   }

   # add this time to the list
   tsim[i]=t
}


#create a histogram of the distribution under the null of the permutation test

hist(tsim,breaks=20)
```



What does this plot mean?

We have a cluster around 12-14 minutes, plus a few around 14-17 minutes

The rare detours create a long tail on the right hand side

The standard deviation won't describe this distribution well, there is a large right hand tail

We can get a mean

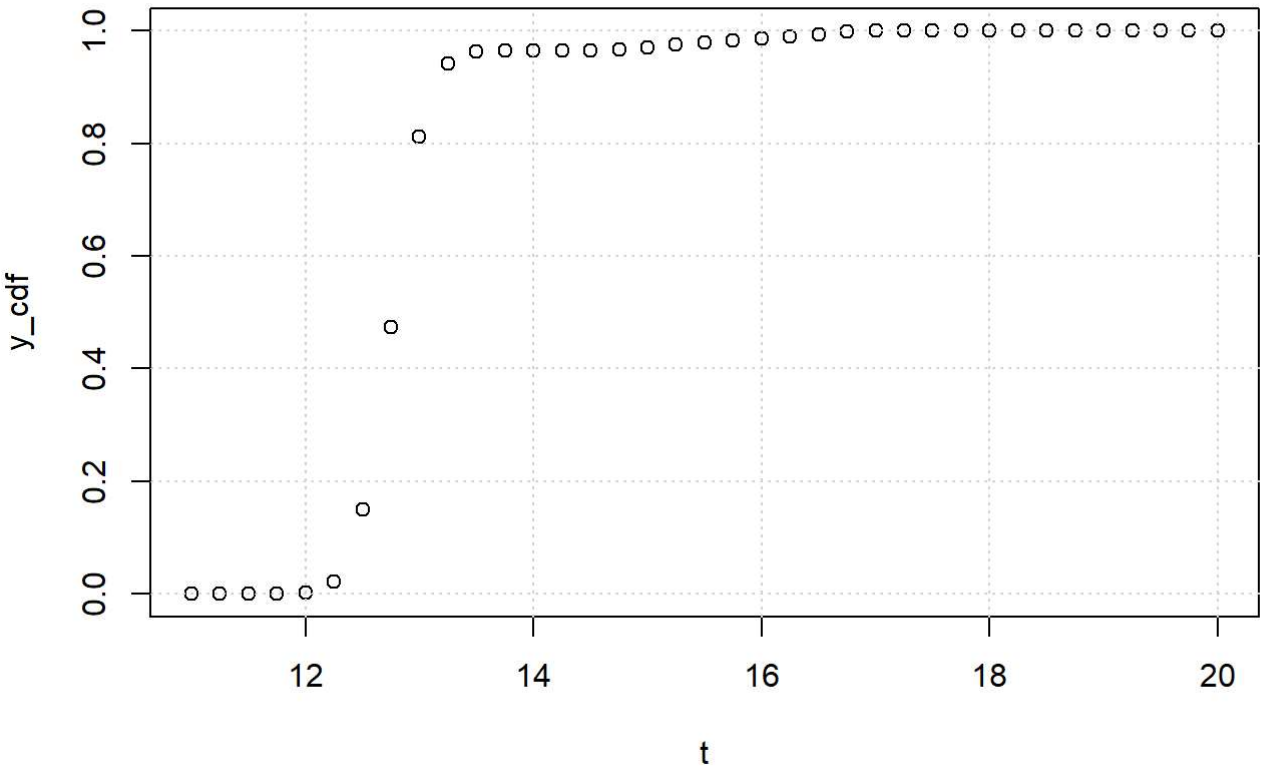
```
mean(tsim)
```

```
## [1] 12.85562
```

```
max(tsim)
```

```
## [1] 16.96193
```

```
#get the ecdf

time_cdf=ecdf(tsim)

t=seq(11,20,0.25)
y_cdf=time_cdf(t)
plot(t,y_cdf)
grid()
```



What percentage of the time is my drive 14 minutes or less?

```
time_cdf(14)
```

```
## [1] 0.9652
```

Bayesian Linear Regression

Use Bayesian methods to evaluate parameter uncertainty in a regression model

See https://www.rensvandeschoot.com/tutorials/r-linear-regression-bayesian-using-brms/ (https://www.rensvandeschoot.com/tutorials/r-linear-regression-bayesian-using-brms/)

for the source material

NOTE: rstan is complex and may not install easily on your system. If it doesn't install easily, don't try to fix matters, just read my discussion of the results

```
library(rstan)
```

```
## Warning: package 'rstan' was built under R version 4.4.3
```

```
## Loading required package: StanHeaders
```

```
## Warning: package 'StanHeaders' was built under R version 4.4.3
```

```
##
## rstan version 2.32.6 (Stan version 2.32.2)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(brms)
```

```
## Warning: package 'brms' was built under R version 4.4.3
```

```
## Loading required package: Rcpp
```

```
## Loading 'brms' package (version 2.22.0). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
```

```
##
## Attaching package: 'brms'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```
## The following objects are masked from 'package:gtools':
##
##     ddirichlet, rdirichlet
```

```
## The following object is masked from 'package:stats':
##
##     ar
```

```
library(psych) #to get some extended summary statistics
```

```
## Warning: package 'psych' was built under R version 4.4.3
```

```
##
## Attaching package: 'psych'
```

```
## The following object is masked from 'package:brms':
##
##     cs
```

```
## The following object is masked from 'package:rstan':
##
##     lookup
```

```
## The following object is masked from 'package:gtools':
##
##     logit
```

```
library(tidyverse) # needed for data manipulation and plotting
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ ggplot2::%+%()   masks psych::%+%()
## ✖ ggplot2::alpha() masks psych::alpha()
## ✖ tidyr::extract() masks rstan::extract()
## ✖ dplyr::filter()  masks stats::filter()
## ✖ dplyr::lag()     masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

Get the data from a remote site

```
dataPHD <- read.csv2(file="https://raw.githubusercontent.com/LaurentSmeets/Tutorials/master/Blavaan/phd-delays.csv")
colnames(dataPHD) <- c("diff", "child", "sex","age","age2")
```

```
head(dataPHD)
```

```
##   diff child sex age age2
## 1   47     1   1  35 1225
## 2   29     1   0  38 1444
## 3   22     1   1  31  961
## 4   19     1   0  39 1521
## 5   17     1   1  36 1296
## 6   15     1   1  38 1444
```

diff- the delay time in months between planned completion of a phd and the actual completion time

child- the number of children the PhD candidate has

sex- sex of the candidate

age- age of candidate

age2- squared age allows non-linear response to age

Bayes Model 1, dependence on age and age2

this cell will run slowly!

```
model <- brm(formula = diff ~ age + age2,
             data    = dataPHD,
             seed    = 123)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

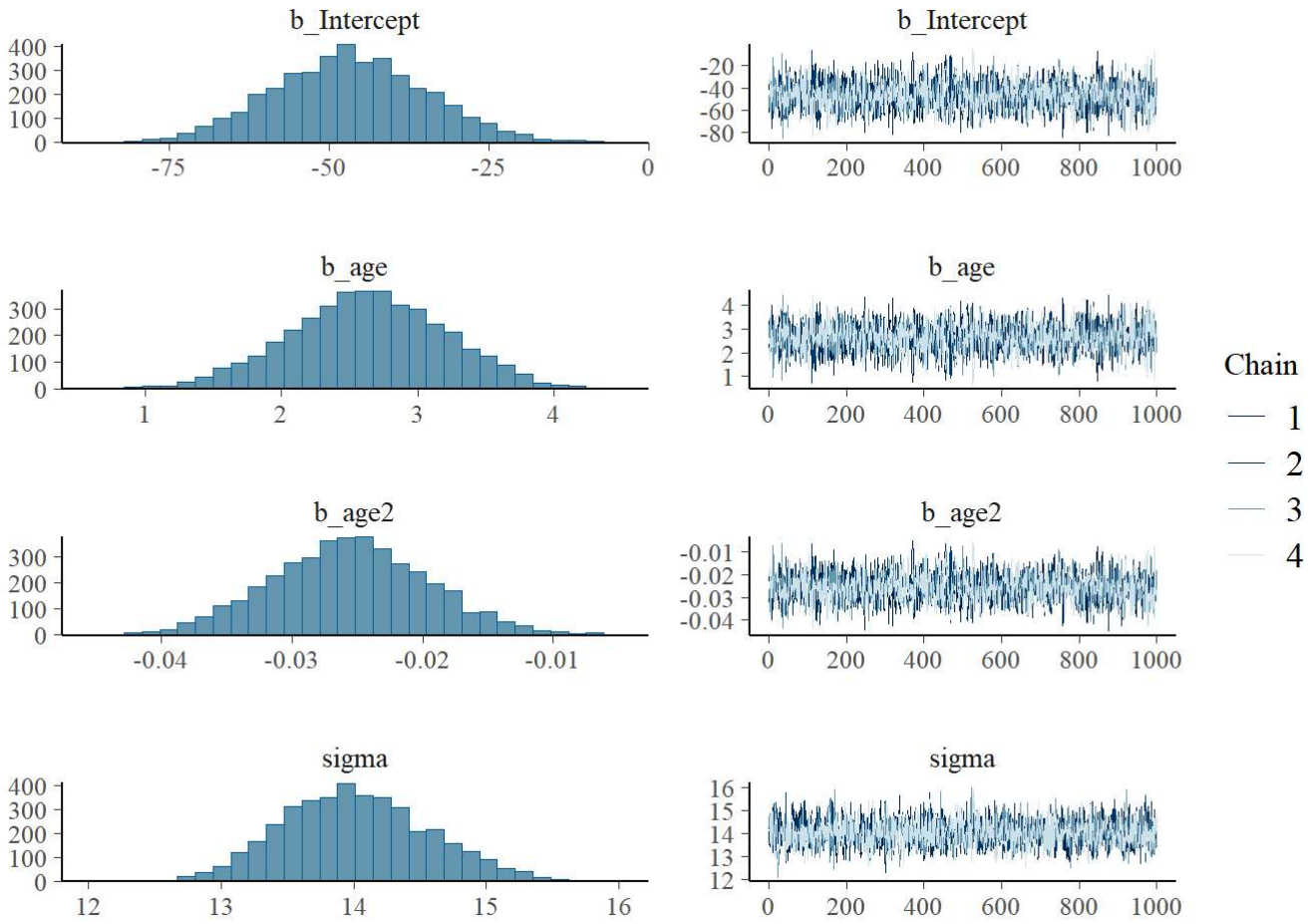
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.22 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.077 seconds (Warm-up)
## Chain 1:                0.039 seconds (Sampling)
## Chain 1:                0.116 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.103 seconds (Warm-up)
## Chain 2:                0.039 seconds (Sampling)
## Chain 2:                0.142 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
```

```
## Chain 3:
## Chain 3: Gradient evaluation took 3e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.065 seconds (Warm-up)
## Chain 3:                0.038 seconds (Sampling)
## Chain 3:                0.103 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.078 seconds (Warm-up)
## Chain 4:                0.039 seconds (Sampling)
## Chain 4:                0.117 seconds (Total)
## Chain 4:
```

```
summary(model)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: diff ~ age + age2
##    Data: dataPHD (Number of observations: 333)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   -46.65     12.19   -69.91   -22.37 1.00     2184     2114
## age           2.63      0.58     1.49     3.74 1.00     2146     1923
## age2         -0.03      0.01    -0.04    -0.01 1.00     2159     2178
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    14.02      0.55    13.02    15.12 1.00     2605     1969
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
plot(model)
```



We actually get the parameter value, but also a distribution for each parameter, this is a lot of detail about how certain we are about the model.

We can also specify initial estimates of the parameters and their distributions called “the priors”, and use the data as a way to update the priors. This is a huge advantage in situations where we know things about the priors.

Here is how we could specify the priors for age and age2

```
priors2 <- c(set_prior("normal(3, 0.632)", class = "b", coef = "age"),
             set_prior("normal(0, 0.316)", class = "b", coef = "age2"))
```

```
model2 <- brm(formula = diff ~ age + age2,
              data    = dataPHD,
              prior = priors2,
              seed    = 123)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.08 seconds (Warm-up)
## Chain 1:                0.036 seconds (Sampling)
## Chain 1:                0.116 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 6e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.114 seconds (Warm-up)
## Chain 2:                0.04 seconds (Sampling)
## Chain 2:                0.154 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
```

```
## Chain 3:
## Chain 3: Gradient evaluation took 4e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.13 seconds (Warm-up)
## Chain 3:                0.037 seconds (Sampling)
## Chain 3:                0.167 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 5e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.042 seconds (Warm-up)
## Chain 4:                0.039 seconds (Sampling)
## Chain 4:                0.081 seconds (Total)
## Chain 4:
```

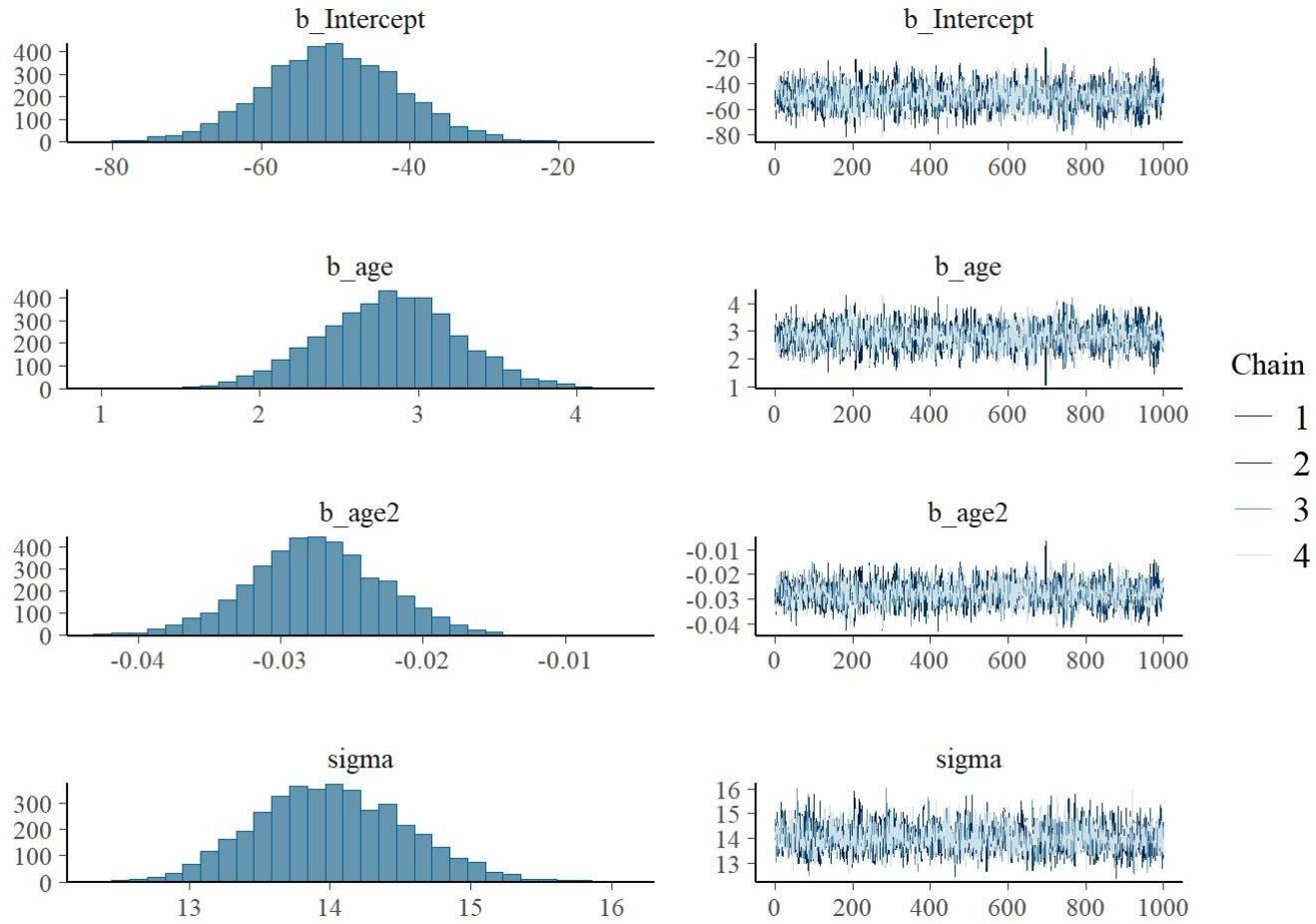
The Bayesian caculation is running a MCMC calculation, at Monte Carlo, Markov Chain calculation. It uses a prior distribution of the parameter values and then a long series of simulations to update the prior distribution based on the model and the data, to generate a posterior distribution of parameter values

This trial used a “flat prior” or “uninformative prior” based on the idea that we have not idea what the parameter values should be.

```
summary(model2)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: diff ~ age + age2
##    Data: dataPHD (Number of observations: 333)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   -50.50      9.33   -69.09   -31.97 1.00     2180     2018
## age           2.82      0.44     1.95     3.71 1.00     2104     1784
## age2         -0.03      0.00    -0.04    -0.02 1.00     2110     1889
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    14.02      0.54    13.03    15.11 1.00     2531     2318
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
plot(model2)
```



If we look at the results from this second Bayesian model, the uncertainty in the coefficients of age and age2 has been decreased by specifying a more realistic prior, incorporating prior knowledge of the parameters.

Bayesian methods can produce more accurate estimates of parameter distributions, that is the big advantage

Cross Validation

See

https://rpubs.com/muxicheng/1004550 (https://rpubs.com/muxicheng/1004550)

When we fit a model to data and then try to test the predictive performance of that model on the same data, we typically find the performance of the model on new data is not as good.

This is a phenomena called “overfitting”, it’s kind of like drawing a curve through all the points on a graph, the model is too close the data

Ideally we would

1.) Train the model on a “training set” 2.) Go and collect new data as a “test set” 3.) Test our model on the new “test set”

But there are other ways called cross validation to do this

a.) Split the data into “training data” and “validation data” fit the model to the training data, evaluate it on the validation data b.) Repeated cross validation- systematically split the data many times into a training set and a validation set, train and evaluate- do this many times to generate many cross validation measures

We will look at only the test/train split method here, using the iris data

Note:require() is an alternate version of library()

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
require(dplyr)
require(tidyverse)
```

Get the iris data

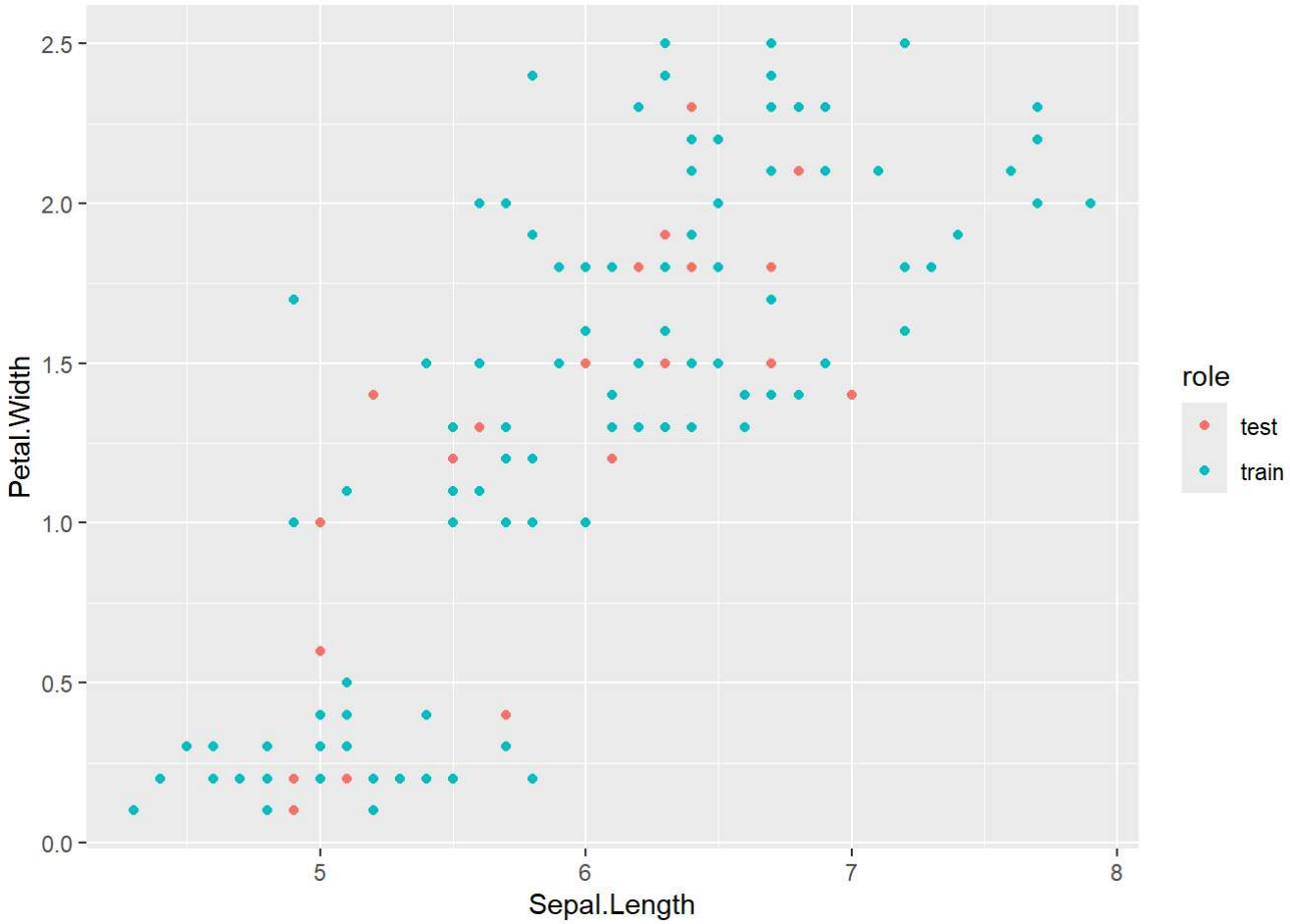
```
data(iris)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Split the data into train and test sets

80% is test 20% is train

```
### Data splitting

# set seed to generate a reproducible random sample
set.seed(123)

# create training and testing data set using index, training data contains 80% of the data set
# 'list = FALSE' allows us to create a matrix data structure with the indices of the observation
s in the subsets along the rows.
train.index.vsa <- createDataPartition(iris$Species, p= 0.8, list = FALSE)
train.vsa <- iris[train.index.vsa,]
test.vsa <- iris[-train.index.vsa,]

# see how the the subsets are randomized
role = rep('train',nrow(iris))
role[-train.index.vsa] = 'test'
ggplot(data = cbind(iris,role)) + geom_point(aes(x = Sepal.Length,
                                                 y = Petal.Width,
                                                 color = role))
```



Looking at how the test and train sets split up the data

Now fit a linear regression model to the training data, then use it to predict the test data

This computes

RMSE- root mean square error, the standard deviation of the error

R^2- the R squared value

MAE- mean average error per point

```
### Training
model.vsa <- lm(Petal.Width ~., data = train.vsa)


### Testing
predictions.vsa <- model.vsa %>% predict(test.vsa)


### Evaluating
test_res=data.frame(RMSE = RMSE(predictions.vsa, test.vsa$Petal.Width),
           R2 = R2(predictions.vsa, test.vsa$Petal.Width),
           MAE = MAE(predictions.vsa, test.vsa$Petal.Width))

test_res
```

```
##        RMSE        R2      MAE
## 1 0.1675093 0.9497864 0.128837
```

We can do the same thing for the training data

These are called “resubstitution estimates”

```
### Training
model.vsa <- lm(Petal.Width ~., data = train.vsa)


### Testing
predictions.train <- model.vsa %>% predict(train.vsa)


### Evaluating
train_res=data.frame(RMSE = RMSE(predictions.train, train.vsa$Petal.Width),
           R2 = R2(predictions.train, train.vsa$Petal.Width),
           MAE = MAE(predictions.train, train.vsa$Petal.Width))

train_res
```

```
##        RMSE       R2       MAE
## 1 0.1630759 0.956026 0.1206628
```

Notice that the R^2 is just slightly higher for the training data, and the RMSE and MAE are just slightly lower.

There is not a lot of overfitting here, but there is some

The cross validation estimate are a better estimate of how the model will perform on new data