IQR prediction example

Jeremy

2022-10-19

Ok, let's create a quick brms fit object

```
library(brms)
```

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

```
## Loading 'brms' package (version 2.18.0). 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:stats':
##
##     ar
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
fit_obj <- brm(
  mpg ~ cyl + disp,
  data = mtcars
)
```

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

```
## Trying to compile a simple C file
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/Rcpp/inclu
## In file included from /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/Core:88
##                  from /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/Dense:1
##                  from /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/StanHeaders/include/stan/math/p
##                  from <command-line>:
## /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1:
##   628 | namespace Eigen {
##       | ^~~~~~~~~
```

```
## /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17
##   628 | namespace Eigen {
##       |                 ^
## In file included from /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/Dense:1
##                  from /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/StanHeaders/include/stan/math/p
##                  from <command-line>:
## /home/jeremy/R/x86_64-pc-linux-gnu-library/4.2/RcppEigen/include/Eigen/Core:96:10: fatal error: compl
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:168: foo.o] Error 1
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL '667cda6873b3c0bde53ba0b20ec47308' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.13 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: 0.047731 seconds (Warm-up)
## Chain 1:                0.023838 seconds (Sampling)
## Chain 1:                0.071569 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '667cda6873b3c0bde53ba0b20ec47308' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 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: 0.087893 seconds (Warm-up)
## Chain 2:                0.026718 seconds (Sampling)
## Chain 2:                0.114611 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '667cda6873b3c0bde53ba0b20ec47308' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 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: 0.058834 seconds (Warm-up)
## Chain 3:                0.0291 seconds (Sampling)
## Chain 3:                0.087934 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '667cda6873b3c0bde53ba0b20ec47308' 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: 0.050103 seconds (Warm-up)
## Chain 4:                0.025218 seconds (Sampling)
## Chain 4:                0.075321 seconds (Total)
## Chain 4:
```

Now, here's the function I want to use to predict the change in mpg based on changes in the other variables. I believe that `posterior_predict` is what I want, but I'm feeling less confident after seeing https://www.andrewheiss.com/blog/2022/09/26/guide-visualizing-types-posteriors/#tldr-diagrams-and-cheat-sheets

```
IQR_prediction <- function(fit, var, quantiles = c(.25, .75)) {
  df = fit$data
  qs <- quantile(df[,var], quantiles)
  first_val = qs[1]
  second_val = qs[2]
  df[,var] = first_val
  first_quartile_pred = predict(fit, newdata = df)
  df[,var] = second_val
  third_quartile_pred = predict(fit, newdata = df)
  delta <- third_quartile_pred - first_quartile_pred
  percent_change <- 100 * delta / first_quartile_pred
  return(percent_change)
}
```

Right now, it keeps the predictions for each row in the dataset. Let's look at an example.

```
preds <- IQR_prediction(fit_obj, 'cyl')
preds
```

```
##        Estimate  Est.Error      Q2.5     Q97.5
##  [1,] -25.57083  16.886359 -41.39672 -16.92421
##  [2,] -26.14208  13.926473 -40.28093 -17.35893
##  [3,] -24.89883  25.452982 -42.31540 -15.05170
##  [4,] -27.83097  -8.872892 -38.00730 -23.04361
##  [5,] -30.69223 -23.212005 -34.69817 -28.99410
##  [6,] -26.70621  -4.819120 -36.53599 -21.05390
##  [7,] -30.36419 -24.109222 -34.74168 -28.83194
##  [8,] -25.29980  17.159366 -40.12291 -16.37764
##  [9,] -25.25820  14.868428 -39.81423 -15.80633
## [10,] -25.98148  10.478691 -38.75750 -18.45662
## [11,] -25.53834   7.490336 -38.33956 -17.22224
## [12,] -28.15979 -10.523961 -37.13512 -23.57521
## [13,] -28.28368 -14.493689 -35.66397 -24.72146
## [14,] -27.93829 -13.664575 -35.66200 -24.81679
## [15,] -34.06681 -33.073445 -36.02986 -33.61594
## [16,] -34.41671 -30.582279 -37.87290 -32.54566
## [17,] -32.74740 -33.161160 -30.87023 -33.47480
## [18,] -23.84938  31.901923 -41.62157 -12.75759
## [19,] -23.90377  30.023117 -41.71733 -13.60005
## [20,] -24.34615  33.428840 -43.03504 -12.78642
## [21,] -25.09960  23.423680 -41.35801 -16.09857
## [22,] -29.23573 -18.643566 -34.39794 -26.43145
## [23,] -28.99121 -15.762165 -35.85649 -25.74607
## [24,] -30.03449 -21.700114 -36.98026 -28.80048
## [25,] -32.14844 -27.579617 -35.71229 -30.88328
```

```
## [26,] -23.56195  31.173450 -41.87000 -13.20833
## [27,] -24.69444  22.632145 -39.81740 -14.38808
## [28,] -24.09143  23.790757 -39.87267 -13.42952
## [29,] -30.22924 -24.124608 -34.78995 -27.88392
## [30,] -25.25728  17.936920 -40.34519 -16.81605
## [31,] -28.74021 -18.348493 -34.44029 -26.74045
## [32,] -24.73898  23.401661 -40.61019 -15.62668
```

My intuition is to do something like:

```
print(paste0("The expected change is between ", round(mean(preds[3]), 2), "% and ", round(mean(preds[4]]
```

```
## [1] "The expected change is between -24.9% and -27.83%."
```

but this feels like it's not capturing the correct uncertainty