

Bayesian SAE using Complex Survey Data

Lecture 2: Introduction to R

Richard Li

Department of Statistics
University of Washington

Outline

R and R studio

Example: simple Bayesian data analysis in R

Example: estimating population mean

Example: working with data and map together

Example: King county data

R and R studio

- ▶ R is a free software environment for statistical computing and graphics.
- ▶ To download R, go to the comprehensive R archive network (CRAN)
- ▶ A new major version of R comes out once a year
- ▶ 2-3 minor releases every year
- ▶ An active community and ecosystem
 - ▶ packages on CRAN
 - ▶ one of the top languages on stackoverflow
 - ▶ #rstat on twitter

- ▶ Unfortunately the default R app is not very user friendly
- ▶ RStudio is a good integrated development environment (IDE)
- ▶ It is also free and runs on multiple platforms with similar interface

The screenshot displays the RStudio IDE interface. The main window is titled "rds" and shows a project workspace. The interface is divided into several panes:

- Editor:** Contains R code for installing and loading packages. The code includes comments and instructions for downloading packages from CRAN and loading them into the current R session.
- Console:** Shows the execution of the R code. It displays the version of xtable (1.8-0) and yaml (2.1.13) installed from CRAN. It also shows the execution of `ggplot2` and `geom_bar` functions, resulting in an error: "Error: could not find function 'ggplot2'".
- Output:** Displays a bar chart showing the proportion of diamonds in different cut categories. The x-axis is labeled "cut" and has categories: "Iod", "Very Good", "Premium", and "Ideal". The y-axis is labeled "prop" and ranges from 0.00 to 1.00. The bars represent the proportion of diamonds in each category.

```
ges(pkgs)
115 -
116
117 # I will download the packages from CRAN and install them in your
118 # system library. If you have problems installing, make that you are
119 # connected to the internet, and that you haven't blocked
120 # <http://cran.r-project.org> in your firewall or proxy.
121
122 You will not be able to use the functions, objects, and help files
123 in a package until you load it with 'library()'. After you have
124 downloaded the packages, you can load any of the packages into
125 your current R session with the 'library()' command, e.g.
126
127 121 - ""(r, eval = FALSE)
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

- ▶ Check if R/RStudio is installed on your computer
- ▶ Find the version of R
 - ▶ R 3.5.0 released on 4/23
 - ▶ R 3.4.4 released on 3/15
 - ▶ R 3.4.3 is the one I'm using locally
 - ▶ If your R version is older than 3.4.2, consider reinstall a new version!
- ▶ Version of RStudio does not matter much in this course, but newer versions are better more stable.

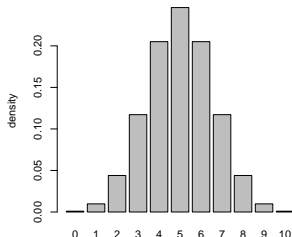
Script and workspace

- ▶ An **R script** contains the codes to perform analysis. It is always better to save an R script than directly typing into R console.
- ▶ As we run through codes, many objects (loaded data, intermediate values, results, ...) are stored in your R session.
- ▶ The multiple objects (e.g., x, y, and z above) are accumulated in your **workspace**.
- ▶ When quitting R, you need to decide if you want to save your workspace (into a .RData file)
 - ▶ Pro: if saved, next time you can reload and pick up all the objects you have created this time.
 - ▶ Con: if you accidentally modified something, there's no easy way to spot the change from looking at the .RData file
- ▶ Good practice for reproducibility: take your script as where the analysis 'live' instead of the workspace.

Simple R examples

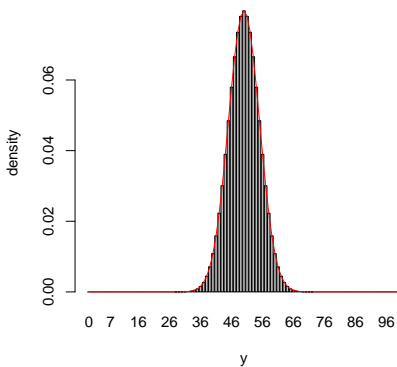
- ▶ In R, either “=” or “< -” can be used to assign values to an object.
- ▶ Object names cannot start with a digit and cannot contain certain special characters (e.g., comma or space)
- ▶ R has many built-in functions to perform common tasks.

```
n <- 10  
p <- 0.5  
y <- 0:n  
bar <- barplot(dbinom(x = y, size = n, prob = p), xlab = "y",  
               ylab = "density", names.arg = y)
```



Simple R examples

```
n <- 100
y <- 0:n
bar <- barplot(dbinom(x = y, size = n, prob = p), xlab = "y",
               ylab = "density", names.arg = y)
lines(bar[, 1], dbinom(x = y, size = n, prob = p),
       col = "red")
```



Data types

Some common types

- ▶ **vector**: a set of values (ordered) of the same mode. We create vectors by using the `c` function (short for combine or concatenate)
- ▶ **matrix**: a 2-dimensional version of vector (internally saved as vector)
- ▶ **array**: a k-dimensional version of vector
- ▶ **data.frame**: a list of vectors where each vector is a column. The vectors may be of different modes. This is typically what we have in mind as 'data'
- ▶ **list**: an ordered collection of any objects

Some common modes

- ▶ numeric: `c(1, 2/5, -2.10)`
- ▶ character: `c("red", "blue", "Seattle")`
- ▶ factor: `factor(c("red", "blue", "Seattle"))`
- ▶ logical: `c("TRUE", "FALSE", "FALSE")`

vectorization

- ▶ One of the most efficient way to deal with data in R is the vectorization
- ▶ i.e., functions can be written to perform operations on vectors element-wise
- ▶ e.g., instead of

```
x <- 1:1000
for (i in 1:1000) {
  x[i] <- x[i]^2 + 1
}
```

we may write

```
x <- x^2 + 1
```

- ▶ Similarly for matrices and arrays.

- ▶ More example:

```
x <- matrix(1:12, nrow = 3, ncol = 4)
y <- x^2
z <- y + x
```

- ▶ Warning: vectorized operations of different length

```
x <- c(1, 2, 3)
y <- c(1, 2, 3, 4, 5)
z <- x + y
```

- ▶ Warning: operation on objects with different dimensions

```
x <- matrix(1:12, nrow = 3, ncol = 4)
y <- x + c(100, 1000)
```

factors and characters

- ▶ Factors are used to represent ordinal data.
- ▶ They are internally saved as integer values.
- ▶ Sometimes R automatically make your data vectors, which can be troubling, e.g., you cannot assign values to a factor vector outside its levels.
 - ▶ e.g., common data input functions like `read.csv`, `read.table`, ...
- ▶ Especially when you think you have a character variable, check they are not factors!
- ▶ There are some new packages that deals with undesired factors better, e.g. `readr`, `tibble`, `forcats`, but we'll stick to R's default behaviors in this course.

Reassign levels to factors

```
x <- factor(c("r", "g", "b"))
levels(x)

## [1] "b" "g" "r"

levels(x) <- c("blue", "green", "red")
```

Turn factors into characters

```
as.character(x)

## [1] "red" "green" "blue"
```

Change orders of factors

```
x <- factor(c("r", "g", "b"))  
as.numeric(x)
```

```
## [1] 3 2 1
```

```
x <- factor(x, levels = c("r", "g", "b", "o"))  
as.numeric(x)
```

```
## [1] 1 2 3
```

More examples

Missing values

```
x <- c(1, 2, NA, 4, 5, 6)
mean(x)
which(is.na(x))
mean(x, na.rm = TRUE)
```

Subsetting data

```
x[c(1, 2)]
x[-c(1, 2)]
x[x != "r"]
x[x %in% c("r", "g")]
```

Combine data

```
x <- c(1, 2, 3, 4)
y <- x^2
cbind(x, y)
rbind(x, y)
data.frame(x = x, y = y)
```


Functions and R packages

```
getsuminv <- function(x, y, z) {  
  return(1/x + 1/y + 1/z)  
}  
suminv <- getsuminv(c(1, 2), c(2, 3), c(3, 4))
```

- ▶ A function has a name, a list of arguments/inputs, and a returned object (to return multiple objects, combine them into a list)
- ▶ Variables defined in functions are local. Variables outside of the function can be used within functions without being passed in (but sometimes dangerous!)
- ▶ **Packages** are the fundamental unit of shareable codes, data, and document. Many packages are hosted on CRAN.
- ▶ Use `install.packages("pkgname")` to download and install from CRAN.
- ▶ Use `library("pkgname")` to load them

List of packages we will use in this course

```
install.packages("ggplot2", dep=TRUE)
install.packages("INLA", repos=c(getOption("repos"),
  INLA="https://inla.r-inla-download.org/R/stable"), dep=TRUE)
install.packages("maptools", dep=TRUE)
install.packages("spdep", dep=TRUE)
install.packages("survey", dep=TRUE)
```

And some optional ones:

```
install.packages("ggrepel", dep=TRUE)
install.packages("RColorBrewer", dep=TRUE)
install.packages("gridExtra", dep=TRUE)
install.packages("rgdal", dep=TRUE)
install.packages("SUMMER", dep=TRUE)
```

R programming resources

- ▶ R for Data science book (online):
<http://r4ds.had.co.nz/introduction.html>
 - ▶ A more modern tutorial in dealing with data efficiently in R
- ▶ Some other popular R courses:
 - ▶ DataCamp's free R tutorial:
<https://www.datacamp.com/courses/free-introduction-to-r>
 - ▶ Jenny Bryan's course materials at UBC:
<http://stat545.com/index.html>
 - ▶ R Programming on Coursera (from Johns Hopkins University)
- ▶ (for Advanced R user) The R Inferno
http://www.burns-stat.com/pages/Tutor/R_inferno.pdf
 - ▶ many many miscellaneous R weird things that may trip you up

Example: simple Bayesian data analysis in R

Bayesian statistics: recap

Three step process

- ▶ Prior: Setup up a full probability model.
- ▶ Data: Condition on observed data.
- ▶ Posterior: evaluate the fit of the model and posterior distribution's implications.

Inference

- ▶ Estimate unknown quantities (parameters) of interest.
- ▶ Estimate uncertainty of the unknown quantities.
 - ▶ “There is a 95% probability that the unknown parameter is in the interval ...”

A simple binomial example

- ▶ p is the infection rate of a virus in a community.
- ▶ y is the number of infected people in screening n randomly selected people.
- ▶ p and y are unknown before screening.
- ▶ p should be less uncertain after observing y .
- ▶ p should be less uncertain after observing y with a large n .

A simple binomial example

- ▶ Data likelihood: $y|p \sim \text{Binomial}(n, p)$
- ▶ Prior: $p \sim \text{Beta}(a, b)$
- ▶ Posterior: distribution of $p|y$
- ▶ Bayes rule:

$$p(p|y) = \frac{p(p)p(y|p)}{p(y)} \propto p^{a+y-1}(1-p)^{b+n-y-1}$$

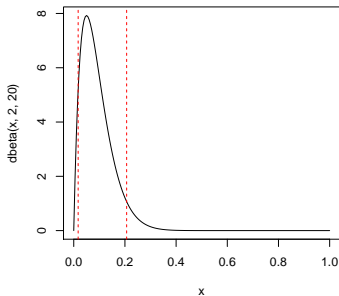
- ▶ We have an analytical representation of the posterior

$$p|y \sim \text{Beta}(a + y, b + n - y)$$

Choosing a and b

- ▶ Choice of a and b reflects the “prior” belief of the infection rate.
- ▶ e.g., for $a = 2$, $b = 20$, 90% of the mass lies between 0.02 and 0.21

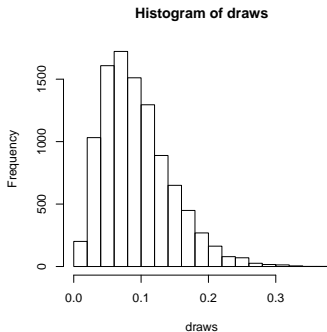
```
x <- seq(0, 1, len = 200)
plot(x, dbeta(x, 2, 20), type = "l")
abline(v = qbeta(0.05, 2, 20), lty = 2, col = "red")
abline(v = qbeta(0.95, 2, 20), lty = 2, col = "red")
```



A simple binomial example

- ▶ Assume we observe $n = 10$, $y = 1$
- ▶ Sample from the posterior

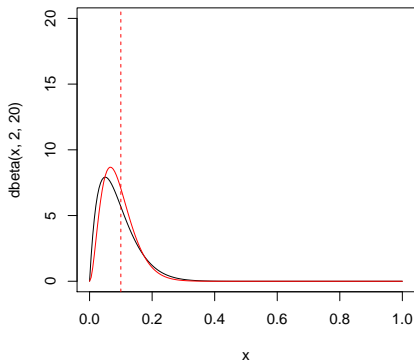
```
Nsim <- 10000  
y <- 1  
n <- 10  
draws <- rbeta(Nsim, 2 + y, 20 + n - y)  
hist(draws)
```



A simple binomial example

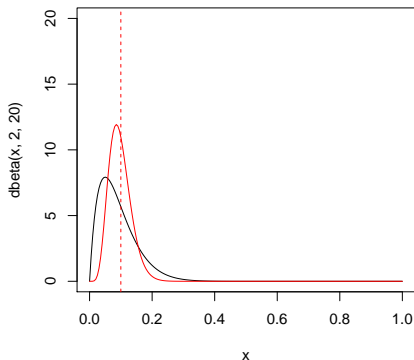
- ▶ The posterior distribution is available analytically

```
plot(x, dbeta(x, 2, 20), type = "l", ylim = c(0, 20))  
lines(x, dbeta(x, 2 + y, 20 + n - y), col = "red")  
abline(v = y/n, lty = 2, col = "red")
```



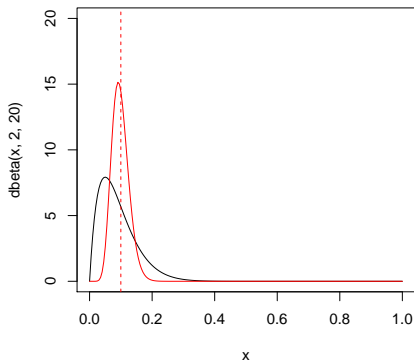
A simple binomial example

```
n <- 50  
y <- 5  
plot(x, dbeta(x, 2, 20), type = "l", ylim = c(0, 20))  
lines(x, dbeta(x, 2 + y, 20 + n - y), col = "red")  
abline(v = y/n, lty = 2, col = "red")
```



A simple binomial example

```
n <- 100
y <- 10
plot(x, dbeta(x, 2, 20), type = "l", ylim = c(0, 20))
lines(x, dbeta(x, 2 + y, 20 + n - y), col = "red")
abline(v = y/n, lty = 2, col = "red")
```



Example: estimating population mean

Bayesian inference for the normal distribution: recap

- ▶ Data: $y_1, \dots, y_n | \mu \sim \text{Normal}(\mu, \sigma^2)$.
- ▶ Prior: $\mu \sim \text{Normal}(\mu_0, \sigma_0^2)$.
- ▶ Assume σ , μ_0 , and σ_0 are known
- ▶ Posterior: $\mu | y_1, \dots, y_n \sim \text{Normal}(\mu_n, \sigma_n^2)$,

$$\mu_n = \mu_0(1 - w) + \bar{y}w, \quad \sigma_n^2 = w \frac{\sigma^2}{n}$$

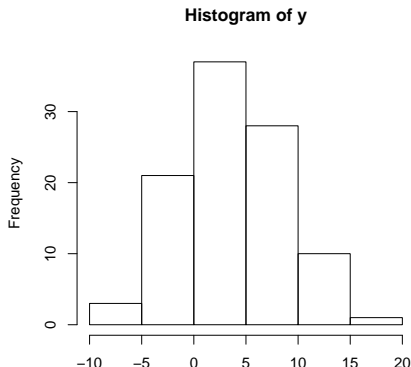
- ▶ The weight

$$w = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2/n}$$

A simple normal example

- ▶ Simulate a dataset

```
sigma <- 5  
mu <- 4  
n <- 100  
y <- rnorm(n, mu, sigma)  
hist(y)
```



A simple normal example

- ▶ MLE and variance of MLE

```
mean(y) # MLE of sample mean
## [1] 3.804005

var(y)/n # variance of sample mean
## [1] 0.2379144
```


A simple normal example

- ▶ What if sample size is smaller

```
set.seed(1)
n <- 10
yy <- rnorm(n, mu, sigma)
mean(yy) # MLE of sample mean

## [1] 4.661014

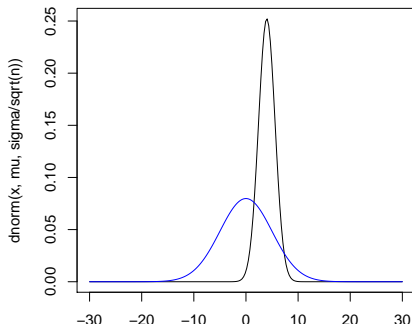
var(yy)/n # variance of sample mean

## [1] 1.523286
```

A simple normal example

- ▶ A wide prior on μ : $\mu \sim \text{Normal}(0, 20)$

```
mu0 <- 0
sigma0 <- 5
x <- seq(-30, 30, len = 200)
plot(x, dnorm(x, mu, sigma/sqrt(n)), type = "l")
lines(x, dnorm(x, mu0, sigma0), col = "blue")
```



A simple normal example: the posterior

- ▶ Posterior: $\mu|y_1, \dots, y_n \sim \text{Normal}(\mu_n, \sigma_n^2)$,

$$\mu_n = \mu_0(1 - w) + \bar{y}w, \quad \sigma_n^2 = w \frac{\sigma^2}{n}$$

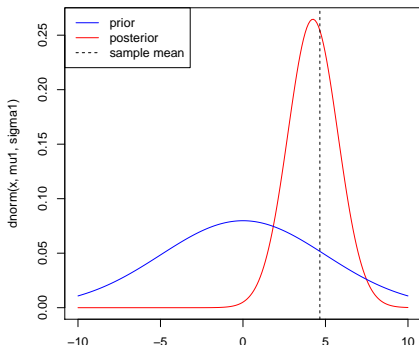
- ▶ The weight

$$w = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2/n}$$

```
w <- sigma0^2/(sigma0^2 + sigma^2/length(yy))
mu1 <- mu0 * (1 - w) + mean(yy) * w
sigma1 <- sqrt(w * sigma^2/length(yy))
```

A simple normal example: the posterior

```
x <- seq(-10, 10, len = 200)
plot(x, dnorm(x, mu1, sigma1), type = "l", col = "red")
lines(x, dnorm(x, mu0, sigma0), col = "blue")
abline(v = mean(yy), lty = 2)
legend("topleft", c("prior", "posterior", "sample mean"),
      col = c("blue", "red", "black"), lty = c(1, 1,
        2))
```



A simple normal example: the posterior

- ▶ Prior mean: 0, prior variance: 25.
- ▶ MLE mean: 4.66, MLE variance: 2.5.
- ▶ Posterior mean: 4.24, posterior variance: 2.27.

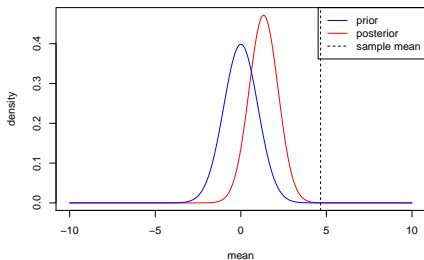
A simple normal example: a more informative prior

- ▶ It's usually easier to write functions in R to perform tasks with varying parameters.

```
getpost <- function(y, sigma, mu0, sigma0) {  
  w <- sigma0^2/(sigma0^2 + sigma^2/length(yy))  
  mu1 <- mu0 * (1 - w) + mean(yy) * w  
  sigma1 <- sqrt(w * sigma^2/length(yy))  
  return(list(mu1 = mu1, sigma1 = sigma1))  
}  
  
plotpost <- function(y, sigma, mu0, sigma0) {  
  post <- getpost(y, sigma, mu0, sigma0)  
  x <- seq(-10, 10, len = 200)  
  plot(x, dnorm(x, post$mu, post$sigma1), type = "l",  
       col = "red", xlab = "mean", ylab = "density")  
  lines(x, dnorm(x, mu0, sigma0), col = "blue")  
  abline(v = mean(yy), lty = 2)  
  legend("topright", c("prior", "posterior", "sample mean"),  
       col = c("blue", "red", "black"), lty = c(1,  
       1, 2))  
}
```

A simple normal example: a more informative prior

```
mu0 <- 0
sigma0 <- 1
posterior <- getpost(yy, sigma, mu0, sigma0)
plotpost(yy, sigma, mu0, sigma0)
```



- ▶ Prior mean: 0, prior variance: 1.
- ▶ MLE mean: 4.66, MLE variance: 2.5.
- ▶ Posterior mean: 1.33, posterior variance: 0.71.

Example: working with data and map together

Simulated dataset: King country example

- ▶ Read the simulated dataset, `simKing`, from `http://faculty.washington.edu/jonno/PAA-SAE.html`.
- ▶ `.rda` is the extension for saved R datasets
- ▶ In R, you can either load data directly from an URL, or download the file first and read from local directory

```
load(url("http://faculty.washington.edu/jonno/PAA-SAE/simKing.rda"))
```

Or

```
load("../data/simKing.rda")
```

Simulated dataset: King country example

The data contains 172,406 observations of simulated “population” data.

- ▶ **areas**: numerical indicator of areas, 1, ..., 49.
- ▶ **weight**: numerical value of weight in pounds.
- ▶ **diabetes**: binary indicator of diabetes.
- ▶ **areaname**: name of the areas.

```
dim(pop)
```

```
## [1] 172406      4
```

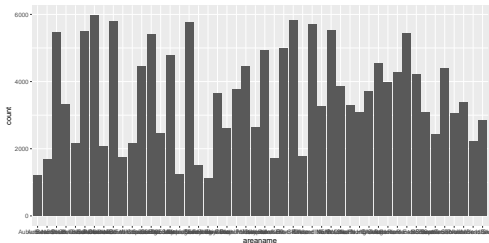
```
head(pop)
```

```
##   area  weight diabetes  areaname
## 1    1 188.5412         0 Sammamish
## 2    1 166.1342         0 Sammamish
## 3    1 186.1471         0 Sammamish
## 4    1 183.1891         0 Sammamish
## 5    1 171.5022         0 Sammamish
## 6    1 177.1157         0 Sammamish
```

Getting to know the data

- ▶ It's usually helpful to first get to know your data by visually checking to
- ▶ `ggplot2` package makes exploratory analysis easier most of the time

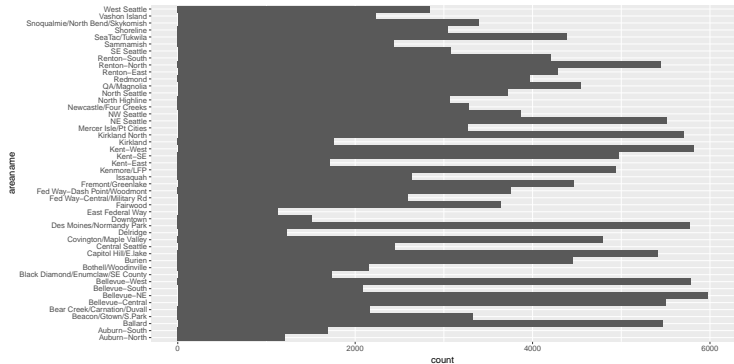
```
library(ggplot2)
ggplot(data = pop) + geom_bar(aes(x = areaname))
```



Not very pretty at first!

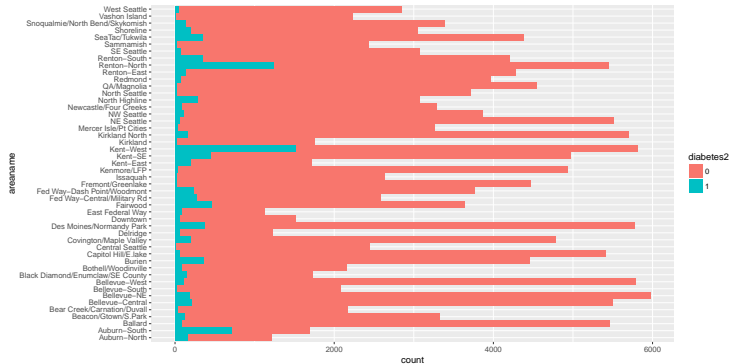
Getting to know the data

```
ggplot(data = pop) + geom_bar(aes(x = areaname)) +  
  coord_flip()
```



Getting to know the data

```
pop$diabetes2 <- factor(pop$diabetes, levels = c(0,
  1))
ggplot(data = pop) + geom_bar(aes(x = areaname, fill = diabetes2)) +
  coord_flip()
```



Getting to know the data

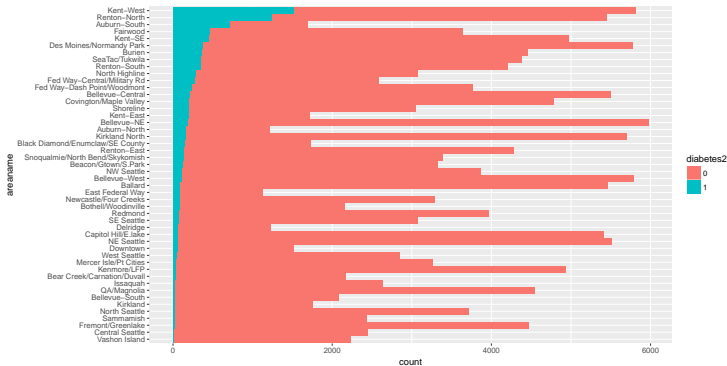
```
n.d <- aggregate(diabetes ~ areaname, pop, sum)
head(n.d)
```

##	areaname	diabetes
## 1	Auburn-North	166
## 2	Auburn-South	724
## 3	Ballard	92
## 4	Beacon/Gtown/S.Park	132
## 5	Bear Creek/Carnation/Duvall	40
## 6	Bellevue-Central	212

Getting to know the data

Count the number of diabetes cases by region.

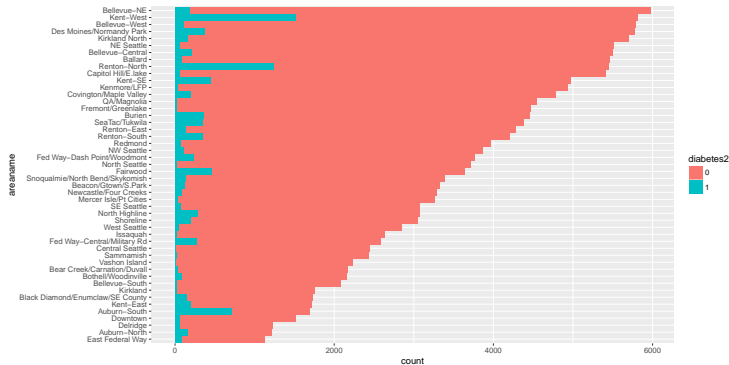
```
names.ordered <- n.d[order(n.d[, 2]), 1]
pop$areaname <- factor(pop$areaname, levels = names.ordered)
ggplot(data = pop) + geom_bar(aes(x = areaname, fill = diabetes2)) +
  coord_flip()
```



Getting to know the data

Count the number of total observations by region.

```
n.t <- aggregate(cbind(total = diabetes) ~ areaname,  
  pop, length)  
names.ordered <- n.t[order(n.t[, 2]), 1]  
pop$areaname <- factor(pop$areaname, levels = names.ordered)  
ggplot(data = pop) + geom_bar(aes(x = areaname, fill = diabetes2)) +  
  coord_flip()
```



- ▶ ESRI (a company one of whose products is ArcGIS) shapefiles consist of three files, and this is a common form.
- ▶ The first file (*.shp) contains the geography of each shape.
- ▶ The second file (*.shx) is an index file which contains record offsets.
- ▶ The third file (*.dbf) contains feature attributes with one record per feature.

We will briefly discuss the basics of using geographic data in R, for more detailed tutorials, see

<https://geocompr.robinlovelace.net/index.html>

Read map files into R

Download the shapefiles from

http://faculty.washington.edu/jonno/PAA-SAE/HRA_ShapFiles/
maptools package provide the easiest way to read in shapefile. However, it does not load in the spatial referencing information.

```
# install.packages('maptools')
library(maptools)
f <- "../data/HRA_ShapeFiles/HRA_2010Block_Clip.shp"
kingshape <- readShapePoly(f)
```

rgdal package provides more powerful ways to read in geographical data. But, additional steps to install GDAL is required.

```
# install.packages('rgdal')
library(rgdal)
kingshape <- readOGR("../data/HRA_ShapeFiles", layer = "HRA_2010Block_Clip")

## OGR data source with driver: ESRI Shapefile
## Source: "../data/HRA_ShapeFiles", layer: "HRA_2010Block_Clip"
## with 48 features
## It has 9 fields
```

Read map files into R

- ▶ Installing GDAL can be tricky for some operating systems.
- ▶ There are many packages to manipulate and plot geographical data. In this course, we will mostly use `ggplot2`
- ▶ A more complete spatial visualization tutorial:
<https://github.com/Robinlovelace/Creating-maps-in-R>

Combine map and data

Turn the map into a 'data.frame' in R

```
geo <- fortify(kingshape, region = "HRA2010v2_")
```

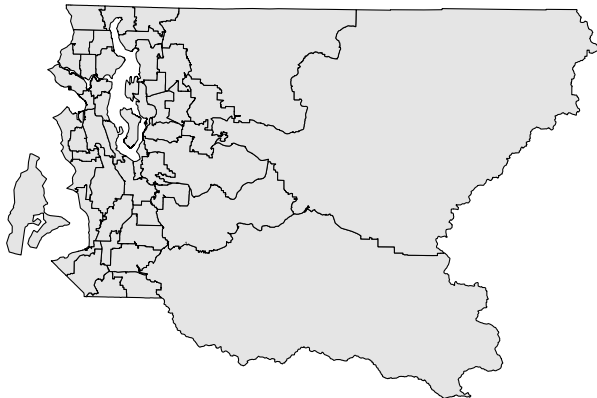
Merge the dataset with the map

```
geo <- merge(geo, n.t, by = "id", by.y = "areaname")
```

Visualize maps

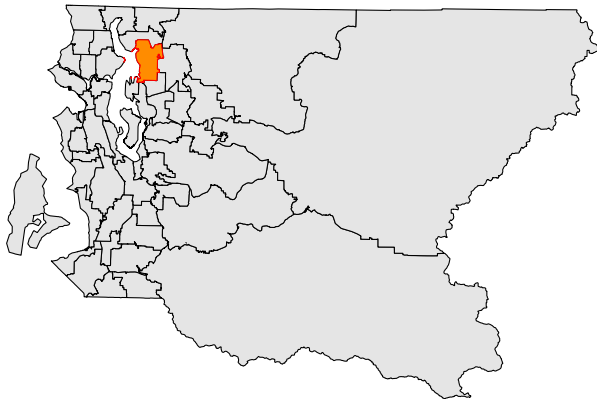
Count the number of total observations by region.

```
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group), color = "black", fill = "gray90")  
g <- g + theme_void()  
g
```



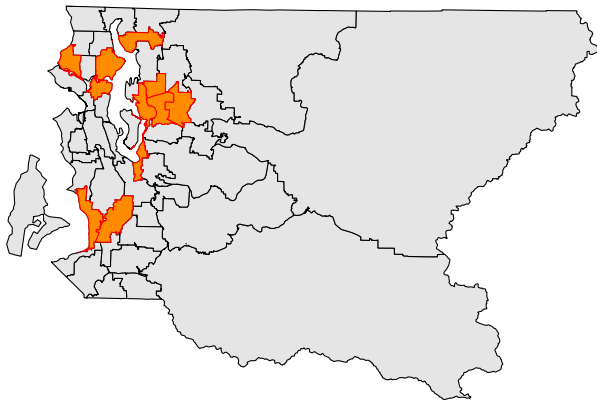
Visualize maps: highlight a region

```
g <- ggplot(geo, aes(x = long, y = lat, group = group)) +  
  geom_polygon(color = "black", fill = "gray90")  
g <- g + geom_polygon(data = subset(geo, id == "Kirkland"),  
  fill = "darkorange", color = "red")  
g <- g + theme_void()  
g
```



Visualize maps: highlight a region

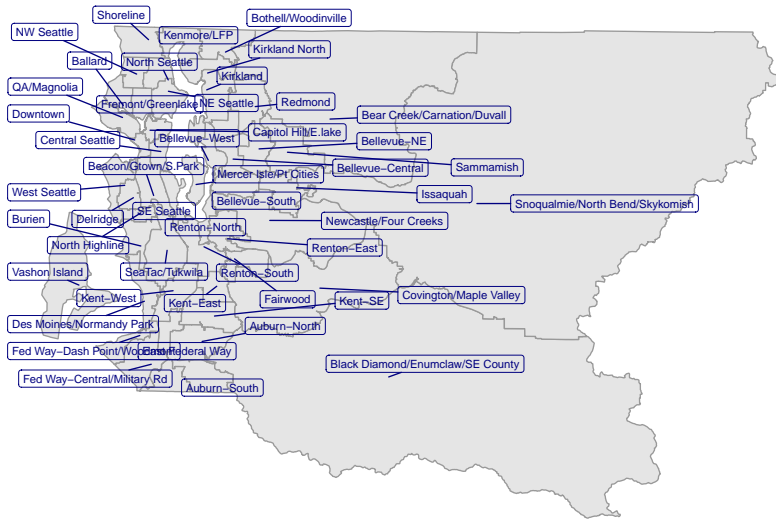
```
g <- ggplot(geo, aes(x = long, y = lat, group = group)) +  
  geom_polygon(color = "black", fill = "gray90")  
g <- g + geom_polygon(data = subset(geo, total > 5000),  
  fill = "darkorange", color = "red")  
g <- g + theme_void()  
g
```



Visualize maps: label regions

```
library(ggrepel)
cnames <- aggregate(cbind(long, lat) ~ id, data = geo,
  function(x) {
    mean(x)
  })
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,
  group = group), color = "gray60", fill = "gray90")
g <- g + geom_label_repel(aes(x = long, y = lat, label = id),
  data = cnames, size = 3, color = "navy", fill = NA)
g <- g + theme_void()
g
```

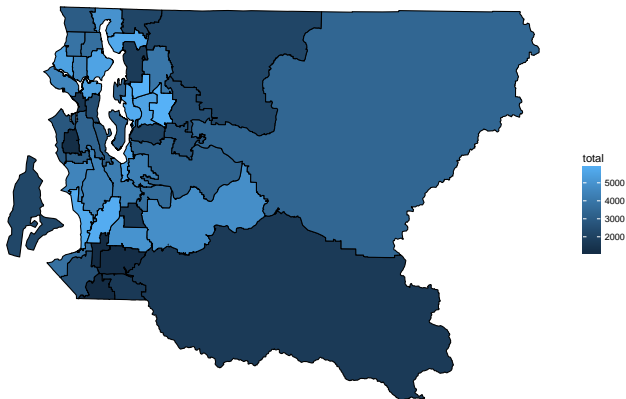

Visualize maps: label regions



Visualize maps: color by variable

Visualize the total population size

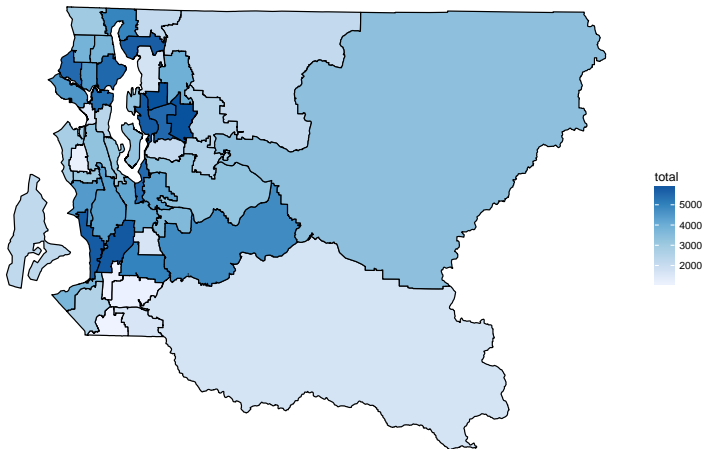
```
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group, fill = total), color = "black")  
g <- g + theme_void()  
g
```



Visualize maps: change color scales

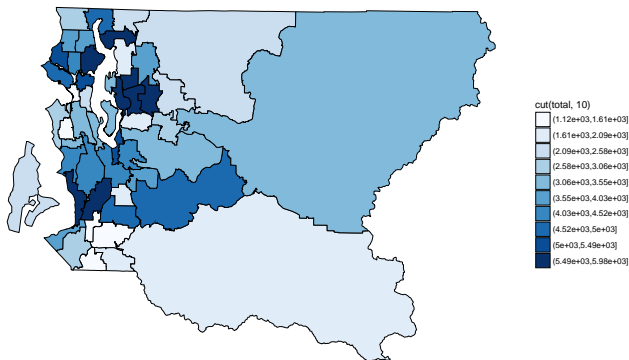
Reverse the scale so that lighter color represent smaller number

```
g <- g + scale_fill_distiller(direction = 1)  
g
```



Visualize maps: bin continuous variables

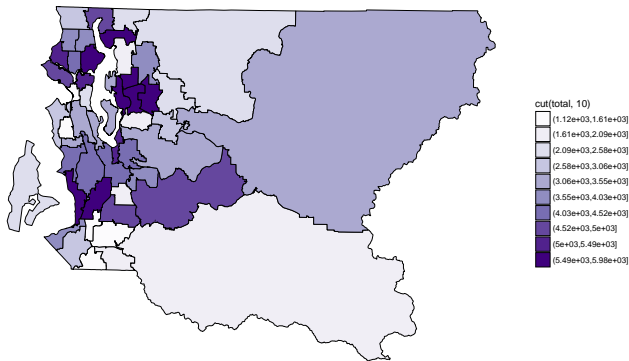
```
library(RColorBrewer)
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,
  group = group, fill = cut(total, 10)), color = "black") +
  theme_void()
g <- g + scale_fill_manual(values = colorRampPalette(brewer.pal(9,
  "Blues"))(10))
g
```



Visualize maps: change color scheme

```
g <- g + scale_fill_manual(values = colorRampPalette(brewer.pal(9,  
  "Purples"))(10))
```

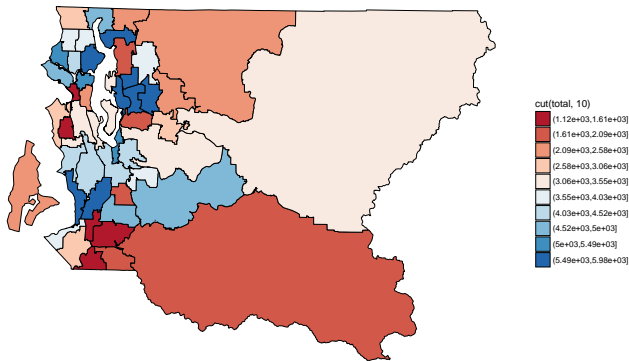
```
g
```



Visualize maps: change color scheme

```
g <- g + scale_fill_manual(values = colorRampPalette(brewer.pal(9,  
  "RdBu"))(10))
```

```
g
```

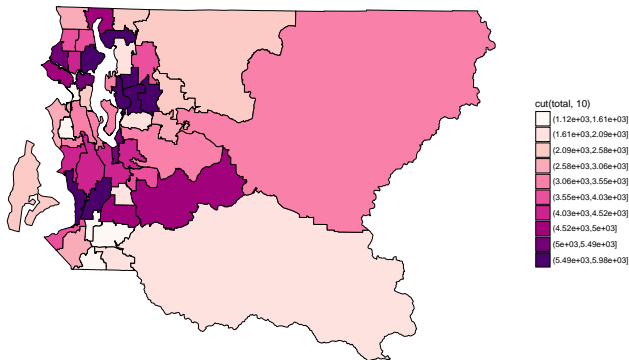


Visualize maps: change color scheme

An useful website to choose color palette: <http://colorbrewer2.org/>

```
g <- g + scale_fill_manual(values = colorRampPalette(c("#fff7f3",  
  "#fde0dd", "#fcc5c0", "#fa9fb5", "#f768a1", "#dd3497",  
  "#ae017e", "#7a0177", "#49006a"))(10))
```

g



More about visualization

- ▶ `ggplot2` allows many more customization
- ▶ Check out `theme_bw()`, `theme_dark()`, `theme_linedraw()`, `coord_map()`, ...

Some useful `ggplot2` tutorials:

- ▶ Introductory tutorial: <http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html>
- ▶ Gallery: <https://www.r-graph-gallery.com/portfolio/ggplot2-package/>
- ▶ Geocomputation with R book: <https://geocompr.robinlovelace.net/index.html>

Example: King county data

Simple random sampling

- ▶ Perform a simple random sampling of 2,000 observations in the simulated population.
- ▶ Calculate the average weight and observed fraction of diabetes in this sample.
- ▶ `set.seed()` is usually a good idea to improve reproducibility.

```
set.seed(123)
n <- 2000
is.sample <- sample(1:dim(pop)[1], n)
sample <- pop[is.sample, ]
w.hat <- aggregate(weight ~ areaname, sample, mean)
d.hat <- aggregate(diabetes ~ areaname, sample, mean)
```

Simple random sampling: MLE

Recall the formulas for the MLE before

```
# simple MLE
size <- aggregate(weight ~ areaname, sample, length)[,
  2]
d.sum <- aggregate(diabetes ~ areaname, sample, sum)[,
  2]
w.sd <- aggregate(weight ~ areaname, sample, sd)[,
  2]/sqrt(size)
d.sd <- sqrt(d.hat[, 2] * (1 - d.hat[, 2])/size)
```

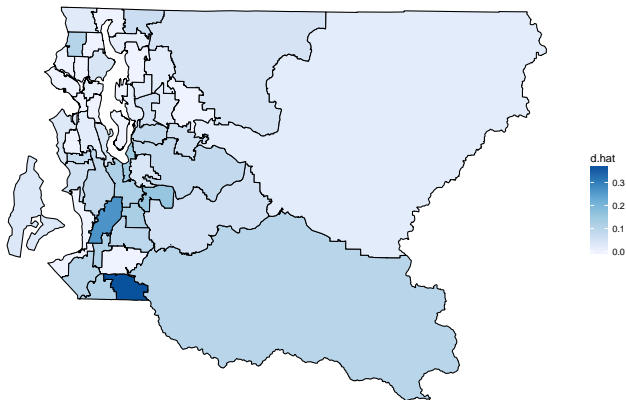
Change the 0 into NA for more informative visualization

```
d.sd[d.hat[, 2] == 0] <- NA
d.sum[d.sum == 0] <- NA
mle <- data.frame(areaname = w.hat[, 1], size = size,
  w.hat = w.hat[, 2], w.sd = w.sd, d.sum = d.sum,
  d.hat = d.hat[, 2], d.sd = d.sd)
```

Simple random sampling: MLE

Visualize the observed fraction of diabetes

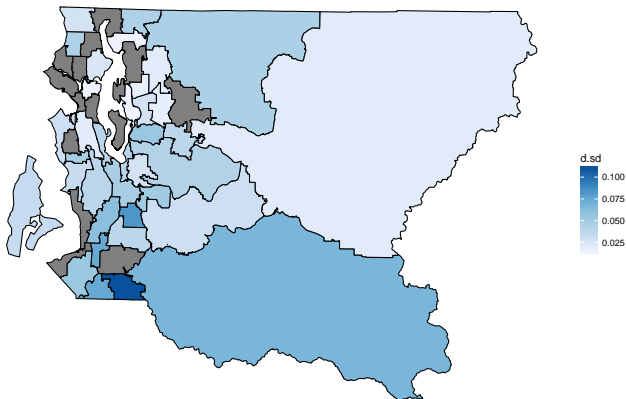
```
geo <- merge(geo, mle, by = "id", by.y = "areaname")  
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group, fill = d.hat), color = "black")  
g <- g + theme_void() + scale_fill_distiller(direction = 1)  
g
```



Simple random sampling: MLE

Visualize the standard error of the observed fraction of diabetes

```
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group, fill = d.sd), color = "black")  
g <- g + theme_void() + scale_fill_distiller(direction = 1)  
g
```



Simple random sampling: Bayesian calculation

Similar to before, write a function for posterior mean and standard deviation for normal and binomial distribution respectively.

```
getpost <- function(y, sigma, mu0, sigma0) {  
  w <- sigma0^2/(sigma0^2 + sigma^2/length(y))  
  mu1 <- mu0 * (1 - w) + mean(y) * w  
  sigma1 <- sqrt(w * sigma^2/length(y))  
  return(c(mu1, sigma1))  
}  
  
getpost.bin <- function(y, a, b) {  
  a1 <- a + sum(y == 1)  
  b1 <- b + length(y) - sum(y == 1)  
  mu1 <- a1/(a1 + b1)  
  sd1 <- sqrt(a1 * b1/(a1 + b1)^2/(a1 + b1 + 1))  
  return(c(mu1, sd1))  
}
```

Simple random sampling: Bayesian calculation

- ▶ Let y_i and n_i denote the number of individuals having type II diabetes and the number of samples in the i -th area.
- ▶ Let w_{ij} denote the weight of the j -th individual in the i -th area.
- ▶ To start, here we estimate the most basic independent model assuming for $i = 1, \dots, n$

$$y_i | p_i \sim \text{Binomial}(n_i, p_i), \quad p_i \sim \text{Beta}(a, b)$$

and

$$w_{ij} | \mu \sim \text{Normal}(\mu_i, \sigma^2), \quad \mu_i \sim \text{Normal}(\mu_0, \sigma_0^2)$$

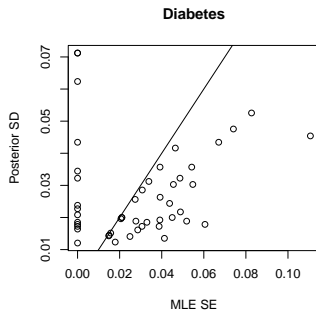
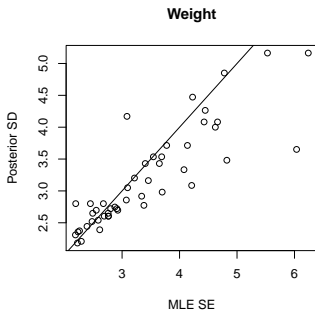
- ▶ Assume we know σ^2 and fix a, b, μ_0 , and σ_0^2 .
- ▶ For now, we model each area independently. We'll revisit and relax this in the next lecture.

Simple random sampling: Bayesian calculation

```
post.est <- matrix(NA, 48, 4)
sigma = 20
mu0 = 180
sigma0 = 10
a0 = 1
b0 = 1
rownames(post.est) <- unique(sample$areaname)
for (area in rownames(post.est)) {
  sub <- sample[sample$areaname == area, ]
  w.post <- getpost(sub$weight, sigma, mu0, sigma0)
  d.post <- getpost.bin(sub$weight, a0, b0)
  post.est[area, ] <- c(w.post, d.post)
}
colnames(post.est) <- c("w.hat2", "w.sd2", "d.hat2",
  "d.sd2")
post.est <- data.frame(post.est)
post.est$areaname <- rownames(post.est)
```

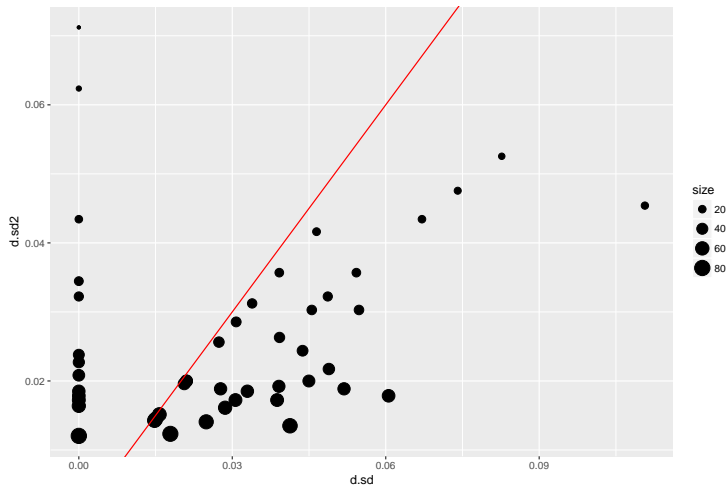

Compare standard errors

```
post.est <- merge(post.est, mle)
post.est$d.sd[is.na(post.est$d.sd)] <- 0
par(mfrow = c(1, 2))
plot(post.est$w.sd, post.est$w.sd2, xlab = "MLE SE",
     ylab = "Posterior SD", main = "Weight")
abline(c(0, 1))
plot(post.est$d.sd, post.est$d.sd2, xlab = "MLE SE",
     ylab = "Posterior SD", main = "Diabetes")
abline(c(0, 1))
```



Compare standard errors

```
g <- ggplot(post.est, aes(x = d.sd, y = d.sd2, size = size))  
g <- g + geom_point() + geom_abline(color = "red")  
g
```



Visualize results with map

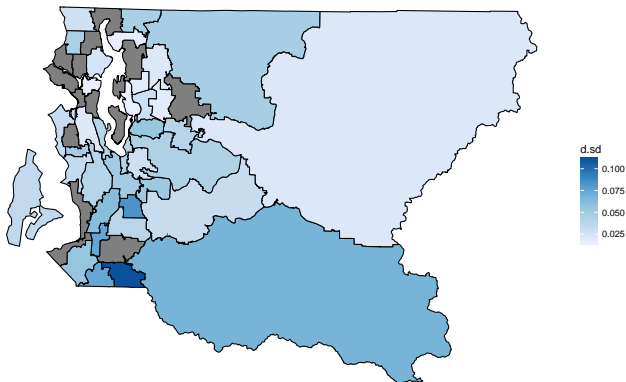
Merge the posterior estimates into the spatial data frame

```
post.est$d.sd[post.est$d.sd == 0] <- NA
geo <- fortify(kingshape, region = "HRA2010v2_")
geo <- merge(geo, post.est, by = "id", by.y = "areaname")
lim <- range(c(post.est$d.sd, post.est$d.sd2), na.rm = TRUE)
```

Visualize results with map: MLE SE

```
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group, fill = d.sd), color = "black")  
g <- g + theme_void() + scale_fill_distiller(limits = lim,  
  direction = 1)  
g <- g + ggtitle("Probability of diabetes by HRA: SE of the MLE")  
g
```

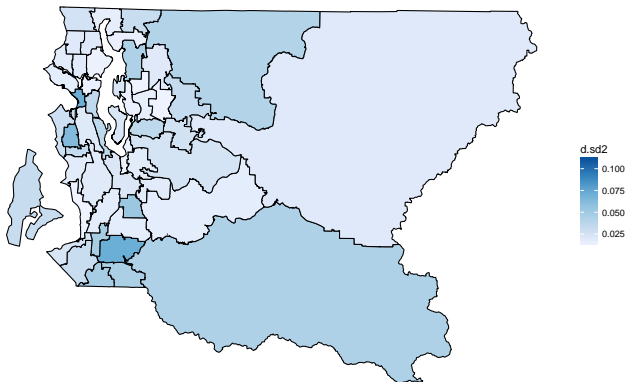
Probability of diabetes by HRA: SE of the MLE



Visualize results with map: Posterior SD

```
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,  
  group = group, fill = d.sd2), color = "black")  
g <- g + theme_void() + scale_fill_distiller(limits = lim,  
  direction = 1)  
g <- g + ggtitle("Probability of diabetes by HRA: Posterior SD")  
g
```

Probability of diabetes by HRA: Posterior SD



Visualize multiple metrics with map

Transform data from 'wide' to 'long' format

```
library(reshape2)
library(plyr)
post.est.wide <- melt(post.est[, c(1, 5, 11)])
head(post.est.wide)
```

##	areaname	variable	value
## 1	Auburn-North	d.sd2	0.07121693
## 2	Auburn-South	d.sd2	0.04540298
## 3	Ballard	d.sd2	0.01785419
## 4	Beacon/Gtown/S.Park	d.sd2	0.01960392
## 5	Bear Creek/Carnation/Duvall	d.sd2	0.04162727
## 6	Bellevue-Central	d.sd2	0.01612686

```
post.est.wide$variable <- revalue(post.est.wide$variable,
  c(d.sd = "MLE SE", d.sd2 = "Posterior SE"))
```

Visualize multiple metrics with map

```
geo <- merge(geo, post.est.wide, by = "id", by.y = "areaname")
g <- ggplot(geo) + geom_polygon(aes(x = long, y = lat,
  group = group, fill = value), color = "black")
g <- g + theme_void() + facet_wrap(~variable)
g <- g + scale_fill_distiller(direction = 1)
g
```

