Regression estimator

Blocks population

Consider again the `blocks` data, as our study population $\mathcal{P}_{Study}$ consisting of $N = 100$ blocks labelled $u = 1, 2, 3, \ldots, 100$.

The blocks are of of uniform thickness and density (all blocks were cut from the same opaque plastic sheet of about 5mm thickness), but have different convex shapes such as shown below:

Recall that data on this population of 100 blocks are available as an R data set `blocks` available in the package loon.data and is loaded as follows:

```
library(loon.data)
data("blocks", package = "loon.data")
```

Recall also the function `getAttributes()` from class

```
getAttributes <- function(samples,     # list of samples (as units)
                          pop,         # population data frame
                          variates,    # variates if needed by popAttr
                          popAttr      # the pop attribute function
                             )
  {# Easy if variates supplied
   if (!missing(variates)) { # lapply forces a list
     lapply(samples,
            FUN = function(samp) {popAttr(pop[samp, variates])} )
   } else {
     # variates are missing
     # Get number of dimensions of pop
     ndims <- length(dim(pop))
     if (!(ndims >= 2)) { # can only have a single index
       lapply(samples,
              FUN = function(samp) {popAttr(pop[samp])})
     } else { # first dimension identifies units in pop
       lapply(samples,
              FUN = function(samp) {popAttr(pop[samp, ])}) # note comma
     }
   }
}
```

---

46 marks

In this question, you will use `getAttributes()` to reproduce some of the results earlier obtained for various sampling plans and some new attributes.

Recall that `getAttributes()` requires its `samples` argument to be a `list` of the samples where each sample is a vector of indices identifying the units (rows) to appear in that sample.

Also, unlike `getAttributeVals()`, `getAttributes()` **returns** a `list` of the attribute evaluated on the

samples. If for example, the attribute is a single numerical value such as the average of some variate over the sample, these values will appear as individual elements in a `list`. To coerce the list of numerical results to a vector (of numeric values), it will be necessary to call `as.numeric()` on the list returned by `getAttributes()`.

a. **(7 marks)** Different sampling plans. In this question, lists of samples from the three different sampling plans you have seen before (simple random sampling, judgment sampling, stratified random sampling). To make it easy to reproduce these plans, we first

```
set.seed(1234567)
```

All sampling plans will produce samples of size `n = 10` blocks from the $N =$ `nrow(blocks) = 100` in the population of `blocks`. That means there are 1.7310309×10^{13} different samples of size 10 available.

i. *(2 marks)* Simple Random Sampling. Construct a collection of 1000 samples of size 10 by simple random sampling. Save the list of samples as the variable `srsSamples` as follows

```
set.seed(1234567)
N <- nrow(blocks)
n <- 10
nSamples <-1000
srsSamples <- lapply(1:nSamples,
                     FUN = function(i) {samp <- sample(N, n, replace = FALSE)})
head(srsSamples, 2)
```

```
## [[1]]
##  [1] 78 27 33 34 59 80 39 64  7 65
##
## [[2]]
##  [1] 80 16 98 19 60 32 71 28 30 44
```

ii. *(2 marks)* Judgment Sampling. Recall that there were judgment samples selected by students who had complete access to all 100 blocks. These are also available from `loon.data` as

The sample indices now appear as the values of variates identifying the order in which each student selected the blocks. These must be turned into a list of numeric vectors containing the indices for each sample. You must complete the following:

```
data("judgment", package = "loon.data")
judgment_data <- judgment[, c(2,3,4,5,6,7,8,9,10,11)]
dimension <- dim(judgment)[1] # dim(judgment) = 33, 11
judgmentSamples <- lapply(1:dimension,
                      FUN = function(x) {as.numeric(judgment_data[x[1],])})
head(judgmentSamples, 2)
```

```
## [[1]]
##  [1] 12 18 17 11 15 20 14 13 16 18
##
## [[2]]
##  [1] 34 35 70 56 32 14  5 88 81 73
```

iii. *(3 marks)* Stratified random sampling. Recall that two groups `A` and `B` had been identified in the population of `blocks`. The stratified random sampling plan is to select 5 blocks at random from each group. While there are many fewer possible stratified samples like this, there are still 4.4891439×10^{12} to choose from.

Here you need to construct a list of 1000 such samples using this stratified random sampling plan and store the result on `stratSamples`.

To begin, first construct the two strata each containing the indices of that group. That is, complete

```r
units <- 1:N
groups <- with(blocks, unique(group))
## separate with groups into A and B
stratifiedSample <- lapply(groups, FUN = function(samp){with(blocks,blocks[group == samp,])})
stratumA <- stratifiedSample[[2]]$id
stratumB <- stratifiedSample[[1]]$id
stratSamples <-lapply(1:nSamples,
                      FUN = function(i) {sampleA <- sample(stratumA, size= 5, replace = FALSE)
                                         sampleB <- sample(stratumB, size= 5, replace = FALSE)
                                         c(sampleA, sampleB)})
head(stratSamples,2)
```

```
## [[1]]
##  [1] 52  7 80 51  4 61 82 44  5 48
##
## [[2]]
##  [1] 92 33 38 35  4  2 43 20 54 48
```

b. **(6 marks)** Sampling distributions. Now you will use `getAttributes()` to determine the sampling distribution of the average weight for each plan.

i. *(2 marks)* Simple random sampling. Using the function `getAttributes()` find the average weight for each sample in `srsSamples` and save the results as a numeric vector on `srsAverageWts`.

```
srsAverageWts <- as.numeric(getAttributes(srsSamples, blocks,
                                          "weight", mean))
head(srsAverageWts, 2)
```

```
## [1] 27.0 42.5
```

ii. *(2 marks)* Judgment sampling. Using the function `getAttributes()` find the average weight for each sample in `judgmentSamples` and save the results as a numeric vector on `judgmentAverageWts`.

Show your code in constructing `judgmentAverageWts` and its first two values as

```
head(judgmentAverageWts, 2)
```

```
judgmentAverageWts <- as.numeric(getAttributes(judgmentSamples, blocks,
                                               "weight",mean))
head(judgmentAverageWts, 2)
```

```
## [1] 44.0 39.5
```

iii. *(2 marks)* Stratified random sampling. Using the function `getAttributes()` find the average weight for each sample in `stratSamples` and save the results as a numeric vector on `stratAverageWts`.

```
stratAverageWts <- as.numeric(getAttributes(stratSamples, blocks,
                                            "weight", mean))
head(stratAverageWts, 2)
```

```
## [1] 32.0 25.5
```

c. **(11 marks)** The perimeters of each block is also available. Physical science informs us that the weight of a block (even these flat convex shaped blocks) is not going to be determined by its perimeter. But it should be related.

i. *(1 mark)* What is the correlation between `weight` and `perimeter` in the `blocks` population?

```
correlation1 <- with(blocks, cor(blocks$weight, blocks$perimeter))
correlation1
```

```
## [1] 0.9014263
```

ii. *(1 mark)* What is the Spearman's rho correlation between `weight` and `perimeter` in the `blocks` population?

```
correlation2 <- with(blocks, cor(x = blocks$weight, y= blocks$perimeter,
                     method = "spearman",))
correlation2
```

```
## [1] 0.9185079
```

iii. *(4 marks)* Fit `weight` as a straight line function of `perimeter` on the population. Show a summary of the fit.

What do you conclude about the fit?

```
fit1 <- lm(weight ~ perimeter, data= blocks)
summary(fit1)
```

```
##
## Call:
## lm(formula = weight ~ perimeter, data = blocks)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20.6215  -3.9615   0.0998   4.8429  21.2560
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -38.7261     3.5206  -11.00   <2e-16 ***
## perimeter     2.7075     0.1314   20.61   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.98 on 98 degrees of freedom
## Multiple R-squared:  0.8126,    Adjusted R-squared:  0.8107
## F-statistic: 424.9 on 1 and 98 DF,  p-value: < 2.2e-16
```

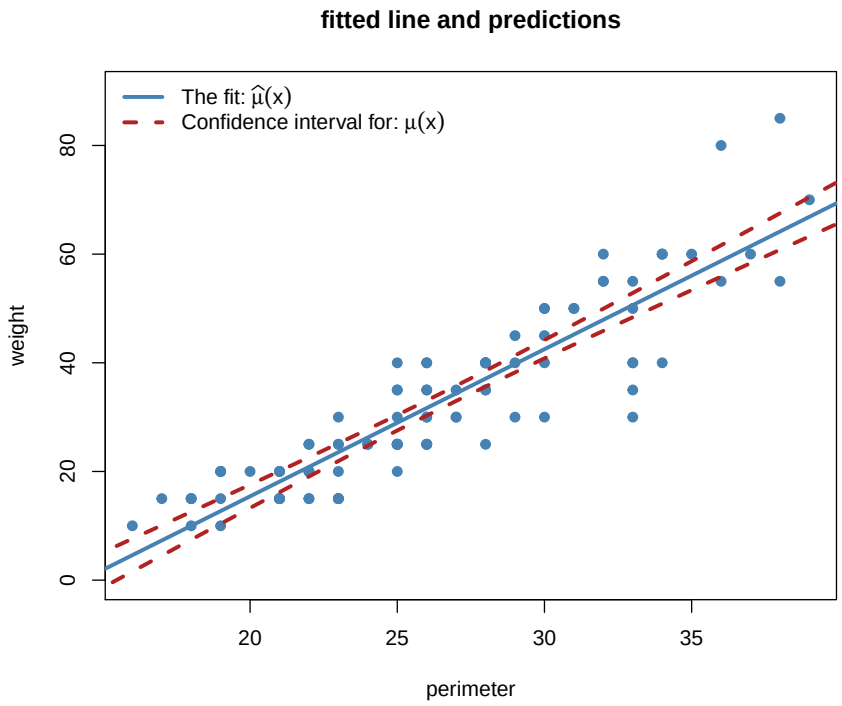
Th Pr(>t) is the probability of observing any value equal or greater than t. Note that three stars represents a highly significant p-value. With small p-value for the weight and perimeter, we can reject the null hypothesis which allows us to conclude that there is a relationship between weight and perimeter.

iv. *(5 marks)* Produce a (nicely labelled) scatterplot of the pairs (`perimeter`, `weight`) for the population.

- Add the least squares fit of the conditional mean function $E(Y) = \mu(x)$ where Y corresponds to `weight` and x to `perimeter`.
- the 95% confidence intervals for $\mu(x)$
- try to fit everything into the display by judicious choice of `xlim` and/or `ylim`

- add a legend to distinguish the fitted $\mu(x)$ from its confidence interval

```
with(blocks,
    {plot(blocks$perimeter, blocks$weight, pch = 19, col = "steelblue",ylim = c(0, 90),
    xlab = "perimeter", ylab = "weight")
      abline(coef(fit1), col = "steelblue", lwd = 3)
      #points(blocks$weight, predict(fit), col = "red")
      title(main = "fitted line and predictions")
    })
## ----new values-----------------------------------------
newValues <- data.frame(perimeter = seq(0, max(blocks$weight), length.out = 100))
## ----confidence intervals-------------------------------
confIntervals <- predict(fit1, newdata = newValues,
                         interval = "confidence", level = 0.95)
lowerConf <- confIntervals[, "lwr"]
upperConf <- confIntervals[, "upr"]
with(newValues,
      lines(perimeter, lowerConf, col = "firebrick", lty = 2, lwd = 3))
with(newValues,
      lines(perimeter, upperConf, col = "firebrick", lty = 2, lwd = 3))
legend("topleft",  bty = "n",
      legend = c(expression(paste("The fit: ", widehat(mu),(x))),
                 expression(paste("Confidence interval for: ", mu(x))),
                 ""),
      col = c("steelblue", "firebrick", "white"),
      lty = c(1, 2, 3), lwd = 3)
```



d. **(2 marks)** Suppose that we know that the average perimeter of the population is $\overline{X} = 26.27$. We might use this to get a different estimate of the population average weight by using a line fitted to our sample and estimating the average weight to be the **predicted** weight when the perimeter is $\overline{X}$ from the line fitted on the sample.

That is, we could estimate the population average weight by

$$\hat{\alpha}_{\mathcal{S}} + \hat{\beta}_{\mathcal{S}} \times \overline{X}$$

where $\hat{\alpha}_{\mathcal{S}}$ and $\hat{\beta}_{\mathcal{S}}$ are the intercept and slope estimates from the line fitted to the `weight` and `perimeter` values from the **sample** $\mathcal{S}$ and $\overline{X} = 26.27$ is the **population** average perimeter.

Let's do that.

Write a function corresponding to the population attribute that returns the predicted value of the fitted straight line at $x = \overline{X} = 26.27$.

That is, complete the following function as a population attribute (to be used as `popAttr` in `getAttributes`):

```
regEstimate <- function(samp) {
  fit1 = lm(weight ~ perimeter, data=samp)
  predict(fit1,data.frame(perimeter=c(26.27)))
  }
regEstimate(blocks)
```

```
##    1
## 32.4
```

e. **(6 marks)** The sampling distributions of the **regression estimator** `regEstimate()`. Knowing the population average perimeter, we can now look at the sampling distribution of the attribute `regEstimate()` under the various plans.

i. *(2 marks)* Simple random sampling. Using `getAttributes()` produce the numeric vector containing the `regEstimate()` for samples in `srsSamples`

```
srsRegEstimates <- as.numeric(getAttributes(srsSamples, blocks,
                                            c("weight","perimeter"), regEstimate))
head(srsRegEstimates, 2)
```

```
## [1] 36.16654 32.58153
```

ii. *(2 marks)* Judgment sampling. Using `getAttributes()` produce the numeric vector containing the `regEstimate()` for samples in `judgmentSamples`

```
judgmentRegEstimates <- as.numeric(getAttributes(judgmentSamples, blocks,
                                            c("weight","perimeter"),regEstimate))
head(judgmentRegEstimates, 2)
```

```
## [1] 38.14073 34.22705
```

iii. *(2 marks)* Stratified random sampling. Using `getAttributes()` produce the numeric vector containing the `regEstimate()` for samples in `stratSamples`

```
stratRegEstimates <- as.numeric(getAttributes(stratSamples, blocks,
                                              c("weight","perimeter"),regEstimate))
head(stratRegEstimates, 2)
```

```
## [1] 34.24993 27.71805
```

f. **(12 marks)** Comparing sampling distributions. Here the effect of using a regression estimate will be investigated by comparing histograms within each sampling plan.

The header of each R code block to produce a pair of histograms will be `{r, fig.height = 7, fig.width = 6, out.height = "100%"}` and each block should include

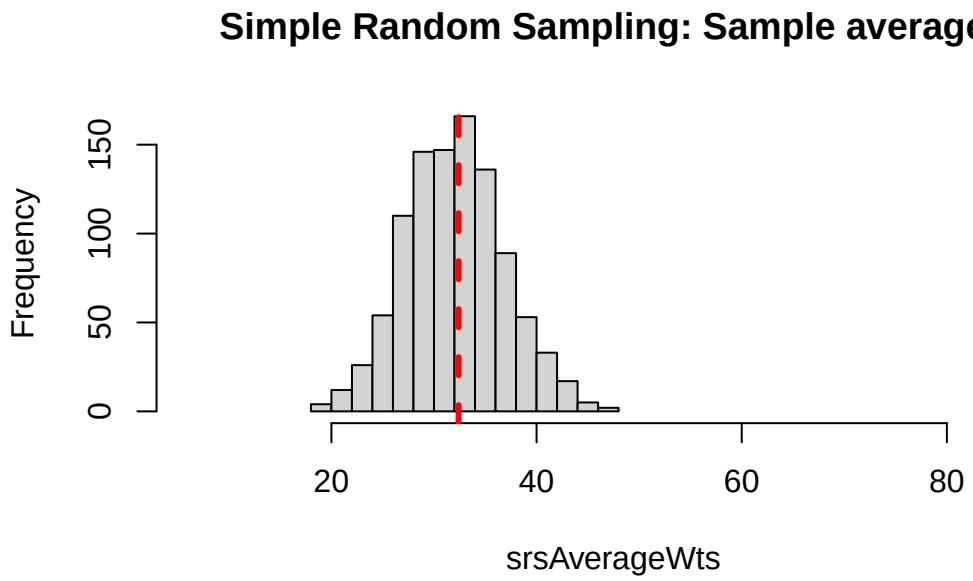
```
xlim <- extendrange(blocks$weight)
popAveWeight <- mean(blocks$weight)
savePar <- par(mfrow = c(2,1))
# do the plotting of the two histograms here with the above xlim
# in each case and a vertical red line at popAveWeight
# hist(aves ... etc,
       main = "Whatever plan: Sample averages",
       xlim = xlim)
# hist(regEstimates ... etc,
       main = "Whatever plan: Regression estimates",
       xlim = xlim)
# then reset the paraneters
par(savePar)
```

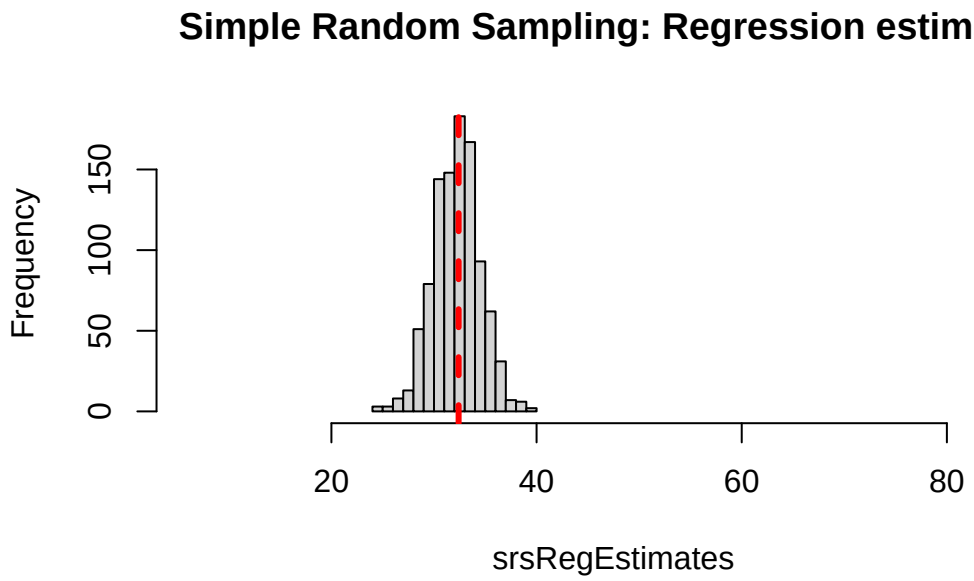
Each display will then consist of two suitably labelled histograms (in separate plots and one above the other), first of the values of the average weights for those samples and second of the `regEstimates` for the same samples.

A vertical "red" line at `v = popAveWeight` is to appear on every histogram.

i. *(4 marks)* Simple random sampling. Draw two suitably labelled histograms in separate plots, one above the other, first of the values `srsAverageWts` and then of `srsRegEstimates`.

```
xlim <- extendrange(blocks$weight)
popAveWeight <- mean(blocks$weight)
savePar <- par(mfrow = c(2,1))
hist(srsAverageWts,
        main = "Simple Random Sampling: Sample averages",
        xlim = xlim)
abline(v = popAveWeight, col="red", lwd=3, lty =2)
hist(srsRegEstimates,
        main = "Simple Random Sampling: Regression estimates",
        xlim = xlim)
abline(v = popAveWeight, col="red", lwd=3, lty =2)
```



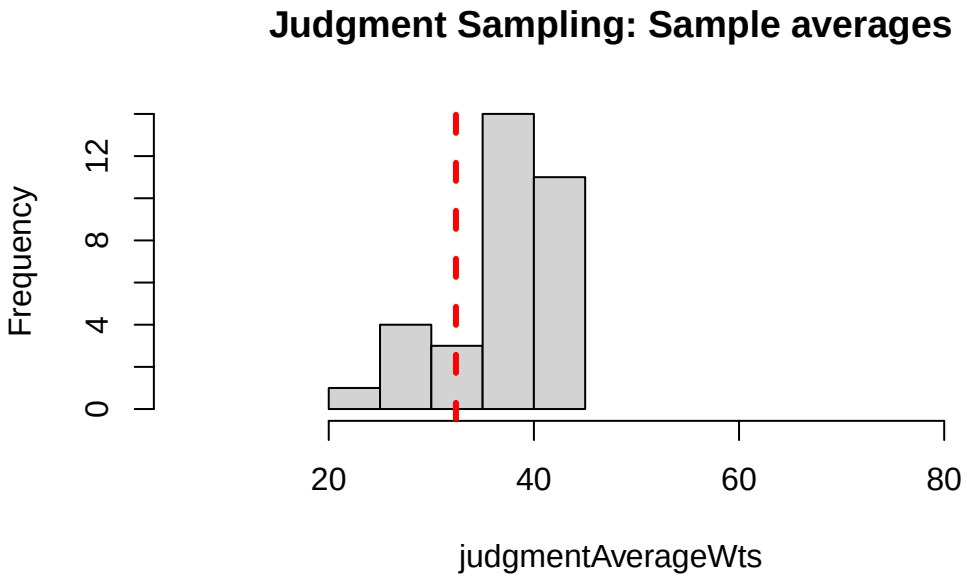


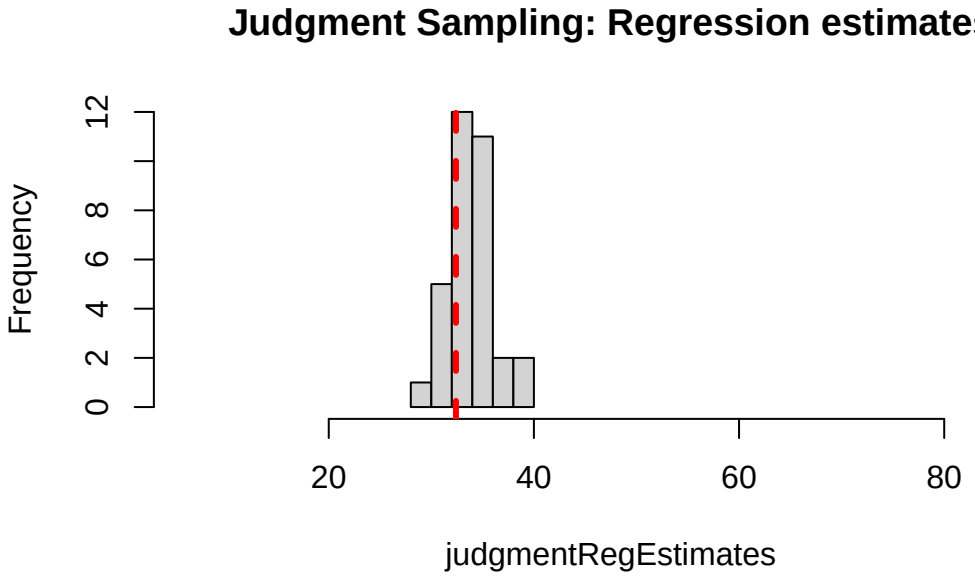

```
par(savePar)
```

ii. *(4 marks)* Judgment sampling. Draw two suitably labelled histograms in separate plots, one above the other, first of the values `judgmentAverageWts` and then of `judgmentRegEstimates`.

```
xlim <- extendrange(blocks$weight)
popAveWeight <- mean(blocks$weight)
savePar <- par(mfrow = c(2,1))
hist(judgmentAverageWts,
        main = "Judgment Sampling: Sample averages",
        xlim = xlim)
abline(v = popAveWeight, col="red", lwd=3, lty =2)
hist(judgmentRegEstimates,
        main = "Judgment Sampling: Regression estimates",
        xlim = xlim)
```

```
abline(v = popAveWeight, col="red", lwd=3, lty =2)
```



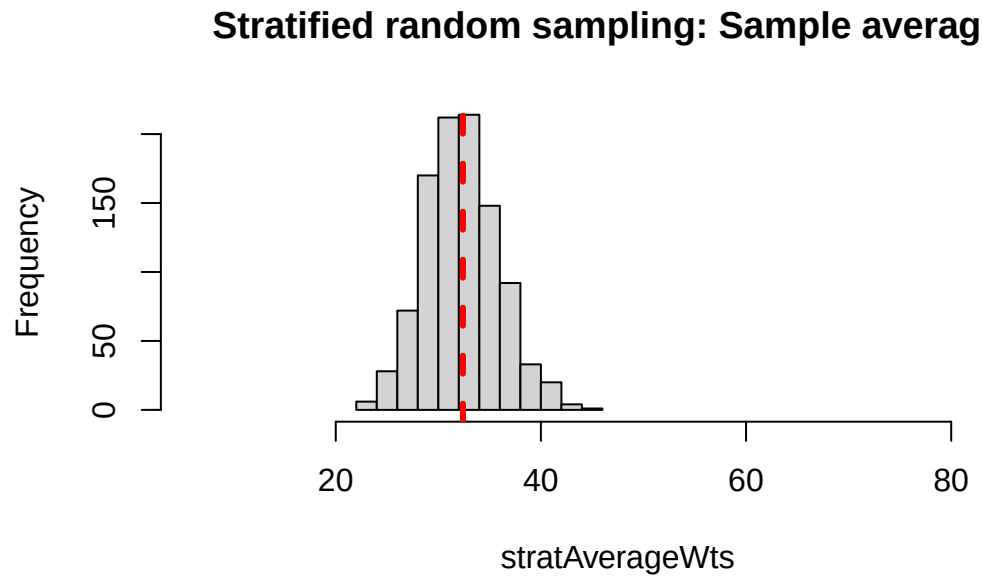


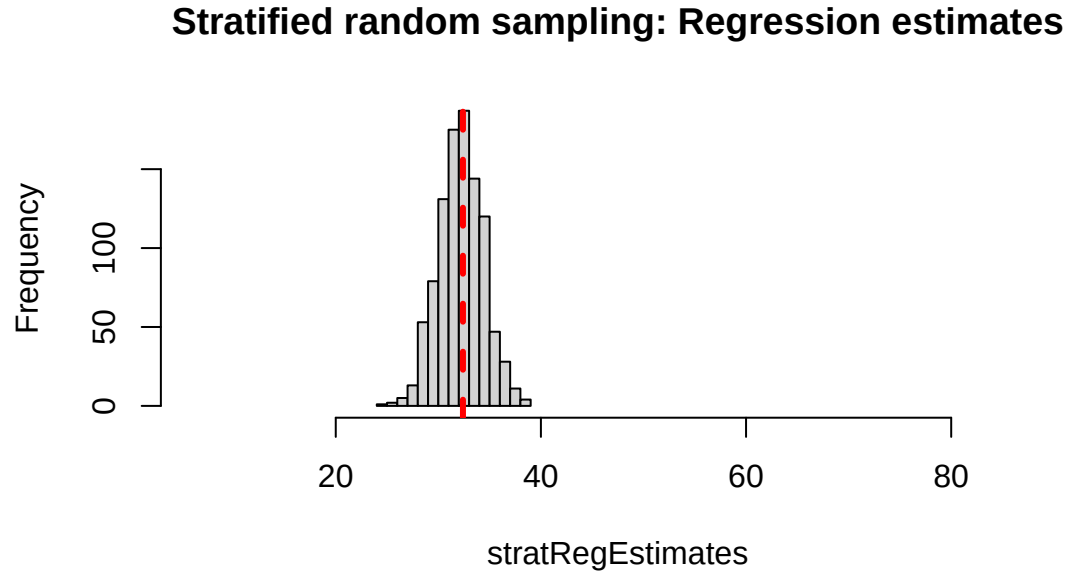

```
par(savePar)
```

iii. *(4 marks)* Stratified random sampling. Draw two suitably labelled histograms in separate plots, one above the other, first of the values `stratAverageWts` and then of `stratRegEstimates`.

```
xlim <- extendrange(blocks$weight)
popAveWeight <- mean(blocks$weight)
savePar <- par(mfrow = c(2,1))
hist(stratAverageWts,
        main = "Stratified random sampling: Sample averages",
        xlim = xlim)
abline(v = popAveWeight, col="red", lwd=3, lty =2)
hist(stratRegEstimates,
```

```
        main = "Stratified random sampling: Regression estimates",
        xlim = xlim)
abline(v = popAveWeight, col="red", lwd=3, lty =2)
```






```
par(savePar)
```

g. **(2 marks)** In light of the foregoing analysis, comment on the meaning of the phrase "All models are wrong, but some are useful" and its relevance in this context.

I think the meaning for "all models are wrong" is because in reality all models have errors and none of them are perfect. And for "but some are useful" means despite all models are wrong, we can still capture the relationship of the variables or anything from the models. In our case, we have three model: simple random sampling, stratified random sampling, and judgment sampling. Even though all the models are wrong, we can still conclude that both simple random sampling and stratified random sampling are the better models when compare to judgment sampling. And we can make use of both simple random sampling and stratified random sampling that are useful and not using judgment sampling which is useless.