TEST2_R

Sean Underwood

Thursday, November 19, 2015

To complete this assignment I created several functions and a few global variables. I describe all of the functions below. The three global variables are: abalone - a dataframe that is used as a parameter throughout the assignment; problems1_7 - calls the test() fucn

abalonedf() - We needed the abalone data set for this assingment. This function loads the data, creates a dataframe, and names the columns.

randomsample() - We have to take lots of random samples in this assingnment. This function creates a random sample of size n with or without replacement.

samples() - Takes a column of a dataframe and uses randomsample() to return a list. I specifically wanted a function that returned a list because I wanted one object that contained all of my initial sample data. A list allows me to have columns of different legnth which is what I wanted because a 1% sample is a different size than a 10% sample. This function completes the first half of question 1.

sampleStatistics()- This function completes question 1. It accepts a list created by the function samples(). The parameter is assumed to be a list whose first column contains a random sample of 1% of the df column. Column 2 will contain a 10% sample, column 3 will contain a 25% sample, column 4 will contain a 50% sample, column 5 will contain a 80% sample and column 6 will contain a 100% sample which is the original column. sampleStatistics() returns a dataframe that has 4 rows and 6 columns. Each column represents a samples size (1%,10%,25%,50%,80%,100%), and each row is a statistics (min,max,mean,variance).

list_hist_grid()- This function carries out the task set forth in question 2. It accepts the LIST created by samples()as a parameter.

sample1000() - This function is for problem 3. sample1000() accepts a dataframe column in the format dfname$colname. We then sample the dfcolumn 1000 times. During each iteration, we perform samples of 5 different sizes (1%,10%,25%,50%,80%). We take the mean of each sample and that mean becomes a row entry in a new dataframe. Our new dataframe will have 5 columns (each for a different sample size) and 1000 rows (each for the mean of a different sample).

df_hist_grid() - This function is for problem 4. It accepts the df created by sample1000() and the dfcolumn from the original used as input for the sample1000() function. The third parameter TITLE defaults to blank. This parameter gives the user an opportunity to enter a subtitle that will appear on line 2 of the histograms. The intent is that the name of the data is used as the title so a reader can tell what data is being used to create the histograms.

sample1000Statistics()- This function is for problem 5. sample1000Statistics() accepts three parameters: the sample1000df is the dataframe returned by the sample1000() function; dfcolumn is the original data that sample1000() performed all of its operations on; and sampleStatisticsdf is the dataframe created by sampleStatistics(). sample1000Statistics() calculates the min,max,mean, and variance of each column of sample1000df and dfcolumn and appends the results to sampleStatisticsdf.

hist_six()- This function is for problem 6. It accepts three parameters: sample1000Statisticsdf is the dataframe returned by sample1000Statistics(); dfcolumn is the original dataframe column; TITLE defaults to blank. This parameter gives the user an opportunity to enter a subtitle that will appear on line 2 of the histograms. The intent is that the name of the data is used as the title so a reader can tell what data is being used to create the histograms. hist_six() will create a histogram of 100% of the data and will overlay label lines that represent SampleMeans for each sample size.

hist_seven()- is the histogram asked for in problem seven. It accepts three parameters:sample1000Statisticsdf is the dataframe returned by sample1000Statistics(); dfcolumn is the original dataframe column; TITLE defaults to blank. This parameter gives the user an opportunity to enter a subtitle that will appear on line 2 of the histograms. The intent is that the name of the data is used as the title so a reader can tell what data is being used to create the histograms. hist_seven() will create a histogram of 100% of the data and will overlay label lines that represent ResampleMeans for each sample size.

test()- accepts two parameters: dfcolumn - a column from a dataframe; and title- defaults to black and gives the user the option of having a subtitle. test() calls all of the previously created fucntions to solve the problems given on the test. The variables below are created to store each question.

To complete tis assingment I created one global variable and three function calls.

1. abalone - a dataframe that is used as a parameter throughout the assignment
2. problems1_7 - call the test() fucntion on the abalone$Whole_wgt dataframe and stores the information
3. problem8a - call the test() function on the diamonds$carat dataframe and stores the information
4. problem8b - call the test() function on the faithful$eruptions dataframe and stores the information (this was acutally the second part of question 8)

Question (9) - The test asks us to provide a brief explanation of what we observed in the measures and histograms as it relates to the sample size and resampling.

For the first grid of histograms, where we were taking single samples of 1%,10%,25%, and 80%, we saw that the distriubtions of each sample became closer and closer to the actual distribution (shown by the 100% sample) as the sample size increased.

For the second second grid of histomgrams, where we were plotting the means of 1000 samples, we saw that as the sample size increased, the distributions of the means became increasing normal (as in the normal distribution).

Finally, on the histograms created in problems 6 and 7, we see that the means of the samples are kind of close to the actual mean but the Resample means are approxmimately equal to the actual mean.

```
# In order to complete the assignment, we load the necessary packages and data
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
require(stats)
require(graphics)
require(grid)
```

```
## Loading required package: grid
```

```
require(triangle)
```

```
## Loading required package: triangle
```

```
require(scales)
```

```
## Loading required package: scales
```

```
require(gridExtra)
```

```
## Loading required package: gridExtra
```

```
abalonedf <- function()
  {
    # abalonedf is a function that creates the abalone dataframe


    # load abalone data set using a http address
    uciaddress <- "http://archive.ics.uci.edu/ml/machine-learning-databases/"

    # create a variable called dataset and assign the abalone data to it
    dataset <- "abalone/abalone.data"

    getdataset <- paste(uciaddress, dataset, sep="")

    # read the contents of the abalone dataset and assign the data to a variable-abalone
    abalone <- read.csv(getdataset)

    # add column names to our new data frame abalone
    colnames(abalone) <- c("Gender","Length","Diameter",
                          "Height", "Whole_wgt",
                          "Shucked_wgt", "Viscera wgt",
                          "Shell wgt", "Rings")
    return(abalone)
  }

abalone<-abalonedf()


randomsample <- function(dataframe,n,replacement)
  { # randomsample() is a function creates a random sample using three parameters:
      # dataframe
      # n = the size of the sample
      # TRUE/FALSE - True for sampling with replacement and False for sampling without re
p

    # the dataframe function creates a dataframe.  the sample function creates a random s
ample

  return (dataframe[sample(nrow(dataframe), n, replace=replacement),]) }
```

```
samples <- function (dfcolumn)
  {
    # samples() takes a column of a data frame as a parameter in the form dfname$colname
and returns a list of random samples.  The first column of the list will contain a random
sample of 1% of the df column.  Column 2 will contain a 10% sample, column 3 will contain
a 25% sample, column 4 will contain a 50% sample, column 5 will contain a 80% sample and
column 6 will contain a 100% sample which is the original column.

    # we ensure that our input is a dataframe with a single column
  df <- data.frame(dfcolumn)

    # create 5 random samples using the random sample function created above.  We use the
round function to round up the multiplication operation we perform on the number of rows
to get a sample of the desired size.  Each random sample is a numeric vector assined to a
variable
  one <- randomsample(df, round((nrow(df))*.01),FALSE)
  ten <- randomsample(df, round((nrow(df))*.1),FALSE)
  twentyfive <- randomsample(df, round((nrow(df))*.25),FALSE)
  fifty <- randomsample(df, round((nrow(df))*.5),FALSE)
  eighty <- randomsample(df, round((nrow(df))*.8),FALSE)

  # I chose to return a list containing each of the random samples because a list can con
tain vectors of different lengths.  This will allow us to extract each vector and convert
it to a data frame later when we need the data.
  return(list(one,ten,twentyfive,fifty,eighty,dfcolumn))

  }

sampleStatistics <- function (listname)
  {
  # sampleStatistics() accepts a list created by the function samples().  The parameter i
s assumed to be a list whose first column contains a random sample of 1% of the df colum
n.  Column 2 will contain a 10% sample, column 3 will contain a 25% sample, column 4 will
contain a 50% sample, column 5 will contain a 80% sample and column 6 will contain a 100%
sample which is the original column.  sampleStatistics() returns a dataframe that has 4 r
ows and 6 columns.  Each column represents a samples size (1%,10%,25%,50%,80%,100%), and
each row is a statistics (min,max,mean,variance).


  # create a new vector 4 elements long for each of our random sample vectors
  onestats <- c(min(listname[[1]]),max(listname[[1]]),mean(listname[[1]]),var(listnam
e[[1]]))
  tenstats <- c(min(listname[[2]]),max(listname[[2]]),mean(listname[[2]]),var(listnam
e[[2]]))
  twentyfivestats <- c(min(listname[[3]]),max(listname[[3]]),mean(listname[[3]]),var(list
name[[3]]))
  fiftystats <- c(min(listname[[4]]),max(listname[[4]]),mean(listname[[4]]),var(listnam
e[[4]]))
  eightystats <- c(min(listname[[5]]),max(listname[[5]]),mean(listname[[5]]),var(listnam
e[[5]]))
```

```
  hundredstats <- c(min(listname[[6]]),max(listname[[6]]),mean(listname[[6]]),var(listnam
e[[6]]))

  #combine the 4 vectors into a dataframe so each vector is a column of the new dataframe
  newdf <- cbind(onestats, tenstats, twentyfivestats, fiftystats, eightystats, hundredsta
ts)

  # name the columns of the df
  colnames(newdf)<- c('1% Sample', '10% Sample','25% Sample','50% Sample','80% Sample','1
00% Sample')

  # name the rows of the df
  rownames(newdf) <- c('SampleMin','SampleMax','SampleMean','SampleVariance')

  return(newdf)
  }

list_hist_grid <- function(listname,title='')
  {
    # list_hist_grid() accepts a LIST as a parameter like the list created by the sample
s() function. The parameter is assumed to be a list whose first column contains a random
sample of 1% of the df column.  Column 2 will contain a 10% sample, column 3 will contain
a 25% sample, column 4 will contain a 50% sample, column 5 will contain a 80% sample and
column 6 will contain a 100% sample which is the original column.  The second parameter T
ITLE defaults to blank.  This parameter gives the user an opportunity to enter a subtitle
that will appear on line 2 of the histograms.  The intent is that the name of the data is
used as the title so a reader can tell what data is being used to create the histograms.
list_hist_grid() will create a grid of histograms.  There will be a total of 6 histogram
s, one for each sample.

  # create our bin widths by calculating the range and dividing it by 8 to get 8 bins
    bin_one <- (max(listname[[1]])-min(listname[[1]]))/8
    bin_ten <- (max(listname[[2]])-min(listname[[2]]))/8
    bin_twentyfive <- (max(listname[[3]])-min(listname[[3]]))/8
    bin_fifty <- (max(listname[[4]])-min(listname[[4]]))/8
    bin_eighty <- (max(listname[[5]])-min(listname[[5]]))/8
    bin_hundred <- (max(listname[[6]])-min(listname[[6]]))/8


    # the code for each histogram will follow the same approach.  First we convert the 1s
t column of the parameter list to a data frame so we can use ggplot.
    onedf <- data.frame(listname[1])

    # we assign a column name to our single column df
    colnames(onedf) <-c("Sample")

    # create a plot from the new df. we use onedf as the input and the column data as the
x axis .  We add a histogram layer with our binwidths created above.  I chose to make the
se histograms black outline with white fill.  We add a title layer.  The main title is th
e sample size and the subtitle is the optional parameter that the user can input
```

```
    hist_one <-ggplot(onedf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_one, colour="black", fill="white") +
    ggtitle(sprintf("1%% Sample\n%s",title))

  # the next 5 histograms follow the exact steps as we used for the first histogram.  We
use column 2 of parameter listname to create a dataframe.  We give the data frame a colum
n name and then create a plot with added layers of histogram and title.
    tendf <- data.frame(listname[2])
    colnames(tendf) <-c("Sample")
    hist_ten <-ggplot(tendf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_ten, colour="black", fill="white") +
    ggtitle(sprintf("10%% Sample\n%s",title))

    twentyfivedf <- data.frame(listname[3])
    colnames(twentyfivedf) <-c("Sample")
    hist_twentyfive <-ggplot(twentyfivedf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_twentyfive, colour="black", fill="white") + ggtitle(sprin
tf("25%% Sample\n%s",title))

    fiftydf <- data.frame(listname[4])
    colnames(fiftydf) <-c("Sample")
    hist_fifty <-ggplot(fiftydf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_fifty, colour="black", fill="white") +
    ggtitle(sprintf("50%% Sample\n%s",title))

    eightydf <- data.frame(listname[5])
    colnames(eightydf) <-c("Sample")
    hist_eighty <-ggplot(eightydf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_eighty, colour="black", fill="white") +
    ggtitle(sprintf("80%% Sample\n%s",title))

    hundreddf <- data.frame(listname[6])
    colnames(hundreddf) <-c("Sample")
    hist_hundred <-ggplot(hundreddf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_hundred, colour="black", fill="white") +
    ggtitle(sprintf("100%% Sample\n%s",title))

    # use the grid.arrange() function from the gridExtra library to arrange the histogram
s into 3 rows and 2 columns.
    grid.arrange(hist_one,hist_ten,hist_twentyfive,hist_fifty,hist_eighty,hist_hundred, n
col=2, nrow =3)

    # return a NuLL value
  return()
  }
```

```
sample1000 <- function(dfcolumn)
  {
  # sample1000() accepts a dataframe column in the format dfname$colname.  We then sample
the dfcolumn 1000 times.  During each iteration, we perform samples of 5 different sizes
(1%,10%,25%,50%,80%).  We take the mean of each sample and that mean becomes a row entry
in a new dataframe.  Our new dataframe will have 5 columns (each for a different sample s
ize) and 1000 rows (each for the mean of a different sample).

  # convert df column into a data frame
  df<- data.frame(dfcolumn)

  # initialize several new variables to the NULL value
  answerone<-NULL
  answerten<-NULL
  answertwentyfive<-NULL
  answerfifty<-NULL
  answereighty<-NULL

  # inititae a for loop.
  for(i in 1:1000)
    {

    # each iteration of the loop will set a new seed so we have a different "random" samp
le each time
    set.seed(i)

    # each iteration we take a sample of n= 1% of the # of observations in the dataframe
rounded to the nearest integer value without replacement.  We then take the mean of that
sample and store it in a variable.  Finally, we bind our new variable as a row in a new d
ata frame.  Each iteration will add another row  This means that each row in the "answer"
data frame will be a mean from the sample.  Our loop executes 1000 times so each "answer"
dataframe will have 1000 rows.
    resultone <- mean(sample(dfcolumn,round((nrow(df))*.01),replace=FALSE))
    answerone <- data.frame(rbind(answerone,resultone))

    # the next four blocks of code follow the same logic as above.  we are building a sep
arate dataframe for each sample size with 1 column and 1000 rows.  Every iteration of the
loop adds the mean of the sample to a row.

    resultten <- mean(sample(dfcolumn,round((nrow(df))*.1),replace=FALSE))
    answerten <- data.frame(rbind(answerten,resultten))

    resulttwentyfive <- mean(sample(dfcolumn,round((nrow(df))*.25),replace=FALSE))
    answertwentyfive <- data.frame(rbind(answertwentyfive,resulttwentyfive))

    resultfifty <- mean(sample(dfcolumn,round((nrow(df))*.50),replace=FALSE))
    answerfifty <- data.frame(rbind(answerfifty,resultfifty))

    resulteighty <- mean(sample(dfcolumn,round((nrow(df))*.80),replace=FALSE))
    answereighty <- data.frame(rbind(answereighty,resulteighty))
```

```
    }

  # now that the loop is complete we have 5 single column dataframes each with 1000 rows.
We create the final dataframe constructed of the the 5 dataframes that we built in our lo
op.  The result is a 1000 x 5 data frame

  newdf <- cbind(answerone,answerten,answertwentyfive,answerfifty,answereighty)
  # add column names to the new dataframe
  colnames(newdf)<- c('1% Sample Means', '10% Sample Means','25% Sample Means','50% Sampl
e Means','80% Sample Means')

  return(newdf)

  }




df_hist_grid <-function(df,dfcolumn,title='')
  {
  #accepts the df created by sample1000() and the dfcolumn from the original used as inpu
t for the sample1000() function.  The third parameter TITLE defaults to blank.  This para
meter gives the user an opportunity to enter a subtitle that will appear on line 2 of the
histograms.  The intent is that the name of the data is used as the title so a reader can
tell what data is being used to create the histograms.  list_hist_grid() will create a gr
id of histograms.

  # df_hist_grid() creates histograms for each column in df and the dfcolumn

  # the input from sample1000() represents the means from each of 1000 samples taking fro
m the original data (represented by dfcolumn).

  # create bin sizes, I chose to have 8 bins by dividing the range by 8
    bin_one <- (max(df[1])-min(df[1]))/8
    bin_ten <- (max(df[2])-min(df[2]))/8
    bin_twentyfive <- (max(df[3])-min(df[3]))/8
    bin_fifty <- (max(df[4])-min(df[4]))/8
    bin_eighty <- (max(df[5])-min(df[5]))/8
    bin_hundred <- (max(dfcolumn)-min(dfcolumn))/8

    # The following 6 blocks of code follow the same process.  We create a new data frame
for each column of the input dataframe df.  We add a title to the new dataframe.  We crea
te a plot and then add the following layers: histgram - using our binwidths from above, I
chose to use a red outline on these plots to help the reader distinguish between the plot
s of samples and the plots of means, Title- primary title and then an optional subtitle t
hat can be entered as a parameter, Theme- this layer is necessary because I needed to con
trol the font size.  My title is so large that when I allow ggplot to chose the size I ge
t overlapping text

    onedf <- data.frame(df[1])
```

```
    colnames(onedf) <-c("Sample")
    hist_one <-ggplot(onedf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_one, colour="red", fill="white") +
    ggtitle(sprintf("Distribution of Means of 1000 1%% Samples\n%s",title)) +
    theme(plot.title = element_text(size = 8,colour="black"))


    tendf <- data.frame(df[2])
    colnames(tendf) <-c("Sample")
    hist_ten <-ggplot(tendf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_ten, colour="red", fill="white") +
    ggtitle(sprintf("Distribution of Means of 1000 10%% Samples\n%s",title))+
    theme(plot.title = element_text(size = 8,colour="black"))


    twentyfivedf <- data.frame(df[3])
    colnames(twentyfivedf) <-c("Sample")
    hist_twentyfive <-ggplot(twentyfivedf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_twentyfive, colour="red", fill="white") + ggtitle(sprint
f("Distribution of Means of 1000 25%% Samples\n%s",title))+
    theme(plot.title = element_text(size = 8,colour="black"))


    fiftydf <- data.frame(df[4])
    colnames(fiftydf) <-c("Sample")
    hist_fifty <-ggplot(fiftydf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_fifty, colour="red", fill="white") +
    ggtitle(sprintf("Distribution of Means of 1000 50%% Samples\n%s",title))+
    theme(plot.title = element_text(size = 8,colour="black"))


    eightydf <- data.frame(df[5])
    colnames(eightydf) <-c("Sample")
    hist_eighty <-ggplot(eightydf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_eighty, colour="red", fill="white") +
    ggtitle(sprintf("Distribution of Means of 1000 80%% Samples\n%s",title))+
    theme(plot.title = element_text(size = 8,colour="black"))

    hundreddf <- data.frame(dfcolumn)
    colnames(hundreddf) <-c("Sample")
    hist_hundred <-ggplot(hundreddf, aes(x=Sample)) +
    geom_histogram(binwidth=bin_hundred, colour="red", fill="white") +
    ggtitle(sprintf("100%% Sample\n%s",title))+
    theme(plot.title = element_text(size = 8,colour="black"))

    # use the grid.arrange function to create the desired grid

    grid.arrange(hist_one,hist_ten,hist_twentyfive,hist_fifty,hist_eighty,hist_hundred, n
col=2, nrow =3)
```

```
return()
}
```

```
sample1000Statistics <- function(sample1000df,dfcolumn,sampleStatisticsdf)
  {
  # sample1000Statistics() accepts three parameters.

  #sample1000df is the dataframe returned by the sample1000() function. Sample1000() acce
pts a dataframe column in the format dfname$colname.  We then sample the dfcolumn 1000 ti
mes.  During each iteration, we perform samples of 5 different sizes (1%,10%,25%,50%,8
0%).  We take the mean of each sample and that mean becomes a row entry in a new datafram
e.  Our new dataframe will have 5 columns (each for a different sample size) and 1000 row
s (each for the mean of a different sample).

  #dfcolumn is the second parameter.  dfcolumn is the original data that sample1000() per
formed all of its operations on

  #sampleStatisticsdf is the dataframe created by sampleStatistics(). sampleStatistics()
accepted a list created by the function samples().  The parameter is assumed to be a list
whose first column contains a random sample of 1% of the df column.  Column 2 will contai
n a 10% sample, column 3 will contain a 25% sample, column 4 will contain a 50% sample, c
olumn 5 will contain a 80% sample and column 6 will contain a 100% sample which is the or
iginal column.  sampleStatistics() returns a dataframe that has 4 rows and 6 columns.  Ea
ch column represents a samples size (1%,10%,25%,50%,80%,100%), and each row is a statisti
cs (min,max,mean,variance).

  #sample1000Statistics() calculates the min,max,mean, and variance of each column of sam
ple1000df and dfcolumn and appends the results to sampleStatisticsdf.

  # create a new vector 4 elements long for each of our random sample vectors
  onestats <- c(min(sample1000df[[1]]),max(sample1000df[[1]]),mean(sample1000df[[1]]),va
r(sample1000df[[1]]))

  tenstats <- c(min(sample1000df[[2]]),max(sample1000df[[2]]),mean(sample1000df[[2]]),va
r(sample1000df[[2]]))

  twentyfivestats <- c(min(sample1000df[[3]]),max(sample1000df[[3]]),mean(sample1000d
f[[3]]),var(sample1000df[[3]]))

  fiftystats <- c(min(sample1000df[[4]]),max(sample1000df[[4]]),mean(sample1000df[[4]]),v
ar(sample1000df[[4]]))

  eightystats <- c(min(sample1000df[[5]]),max(sample1000df[[5]]),mean(sample1000d
f[[5]]),var(sample1000df[[5]]))

  hundredstats <- c(min(dfcolumn),max(dfcolumn),mean(dfcolumn),var(dfcolumn))

  #combine the 4 vectors into a dataframe so each vector is a column of the new dataframe
  newdf <- cbind(onestats, tenstats, twentyfivestats, fiftystats, eightystats, hundredsta
ts)


  # name the rows of the df
```

```
rownames(newdf) <- c('ResampleMin','ResampleMax','ResampleMean','ResampleVariance')
answerdf <-rbind(sampleStatisticsdf,newdf)
return(answerdf)
}
```

```
hist_six <- function(sample1000Statisticsdf,dfcolumn,title='')
  {
  # hist_six is the histogram asked for in problem six.  It accepts three parameters:
  # sample1000Statisticsdf is the dataframe returned by sample1000Statistics().  It conta
ins 8 rows and 6 columns.  Each column represents a sample size (1%,10%,25%,80%,100%).  E
ach row represents a statistic (sample min, sample max, sample mean, sample variance, res
ample min, resample max, resample mean, resample variance).  In each case, the sample sta
tistics are from a sample of n=(sample size given in the column) from the original datafr
ame dfcolumn.  The resample statistics are the means of 1000 samples of n=(sample size gi
ven in the column).

  # dfcolumn is the original dataframe column

  # The third parameter TITLE defaults to blank.  This parameter gives the user an opport
unity to enter a subtitle that will appear on line 2 of the histograms.  The intent is th
at the name of the data is used as the title so a reader can tell what data is being used
to create the histograms.

  # hist_six() will create a histogram of 100% of the data and will overlay label lines t
hat represent SampleMeans for each sample size.

  # create bin sizes, we want 8 bins
  bin_hundred <- (max(dfcolumn)-min(dfcolumn))/8

  # ggplot2 works through data frames so I create a dataframe named vlines with the infor
mation that I want the vertical lines to be created with.  The first vector in the data f
rame contains the values and the second contains the labels that I want to appear in the
legend.  I use the data.frame function.  I want the Sample Mean data which is in the thir
d row of the sample1000Statisticsdf.  Then for each sample size I change the column i in
the [3,i] code.  The i=2 is the Ssample mean for 10% samples size, i=3 is the 25% sample,
etc.  The second vector SampleSize contains text of the Titles that I want in the legend.

  vlines <- data.frame(value = c(sample1000Statisticsdf[3,1],sample1000Statisticsd
f[3,2],sample1000Statisticsdf[3,3],sample1000Statisticsdf[3,4],sample1000Statisticsd
f[3,5],sample1000Statisticsdf[3,6]), SampleSize = c("1%", "10%","25%","50%","80%","10
0%"))

  # convert or ensure that the dfcolumn is a dataframe
  hundreddf <- data.frame(dfcolumn)

  # add a column name to the data frame
  colnames(hundreddf) <-c("Sample")

  #create the plot using the 100% data data frame and mapping the aesthetic, add the hist
ogram laeyr specifying bin width and color, add the title layer, add the vertical line la
yer using the new dataframe we created at the beginning of this function. We specify wher
e the data comes from, that we want a line at each "value" in the first vector of the dat
a frame, the color will be different for each entry and the legend will show us the color
matched with our entry in the second vector of our data frame.
  hist_hundred <-ggplot(hundreddf, aes(x=Sample)) +
```

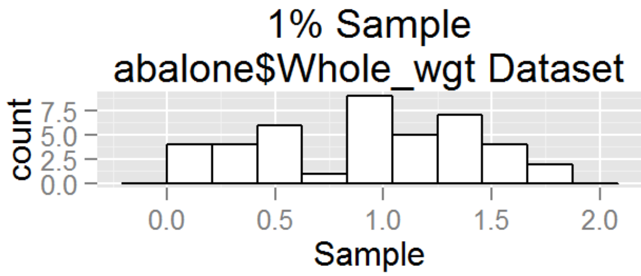
```
      geom_histogram(binwidth=bin_hundred, colour="black", fill="gray") + ggtitle(sprint
f("100%% Sample With Means From Samples\n%s",title)) + geom_vline(data=vlines,aes(xinterc
ept=value, colour=SampleSize), size=1.15, linetype="F1", show_guide=TRUE)


    print(hist_hundred)

    return(NULL)
    }
```

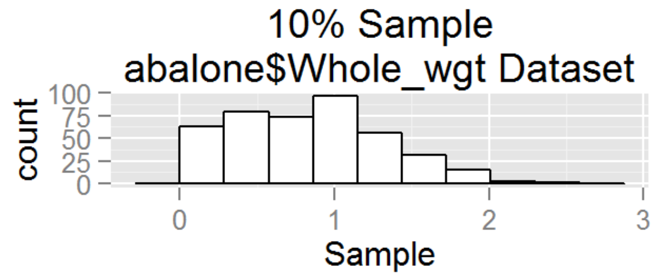
```
hist_seven <- function(sample1000Statisticsdf,dfcolumn,title='')
  {
  # hist_seven is the histogram asked for in problem seven.  It accepts three parameters:
  # sample1000Statisticsdf is the dataframe returned by sample1000Statistics().  It conta
ins 8 rows and 6 columns.  Each column represents a sample size (1%,10%,25%,80%,100%).  E
ach row represents a statistic (sample min, sample max, sample mean, sample variance, res
ample min, resample max, resample mean, resample variance).  In each case, the sample sta
tistics are from a sample of n=(sample size given in the column) from the original datafr
ame dfcolumn.  The resample statistics are the means of 1000 samples of n=(sample size gi
ven in the column).

  # dfcolumn is the original dataframe column

  # The third parameter TITLE defaults to blank.  This parameter gives the user an opport
unity to enter a subtitle that will appear on line 2 of the histograms.  The intent is th
at the name of the data is used as the title so a reader can tell what data is being used
to create the histograms.

  # hist_seven() will create a histogram of 100% of the data and will overlay label lines
that represent ResampleMeans for each sample size.

  # create bin sizes, we want 8 bins
  bin_hundred <- (max(dfcolumn)-min(dfcolumn))/8

  # ggplot2 works through data frames so I create a dataframe with the information that I
want the vertical lines to be created with.  The first vector in the data frame contains
the values and the second contains the labels that I want to appear in the legend.  I use
the data.frame function.  I want the Resample Mean data which is in the seventh row of th
e sample1000Statisticsdf.  Then for each sample size I change the column i in the [7,i] c
ode.  The i=2 is the Resample mean for 10% samples size, i=3 is the 25% sample, etc.  The
second vector SampleSize contains text of the Titles that I want in the Legend.

  vlines <- data.frame(value = c(sample1000Statisticsdf[7,1],sample1000Statisticsd
f[7,2],sample1000Statisticsdf[7,3],sample1000Statisticsdf[7,4],sample1000Statisticsd
f[7,5],sample1000Statisticsdf[7,6]), SampleSize = c("1%", "10%","25%","50%","80%","10
0%"))

  # convert or ensure that we have a dataframe to work with
  hundreddf <- data.frame(dfcolumn)

  # add a column name to the data frame
  colnames(hundreddf) <-c("Sample")

  #create the plot using the 100% data data frame and mapping the aesthetic, add the hist
ogram laeyr specifying bin width and color, add the title layer, add the vertical line la
yer using the new dataframe we created at the beginning of this function. We specify wher
e the data comes from, that we want a line at each "value" in the first vector of the dat
a frame, the color will be different for each entry and the legend will show us the color
matched with our entry in the second vector of our data frame.
  hist_hundred <-ggplot(hundreddf, aes(x=Sample)) +
```

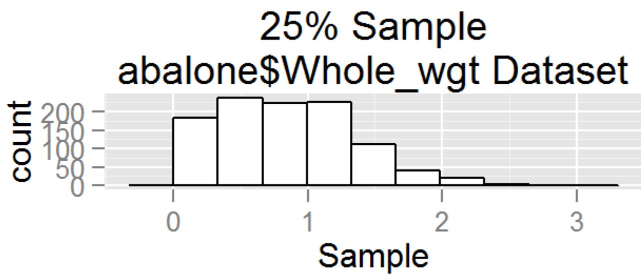
```
      geom_histogram(binwidth=bin_hundred, colour="steelblue", fill="gray") + ggtitle(spr
intf("100%% Sample With Resample Means\n%s",title)) + geom_vline(data=vlines,aes(xinterce
pt=value, colour=SampleSize), size=1.00, linetype="F1", show_guide=TRUE)


    print(hist_hundred)

    return(NULL)
    }
```

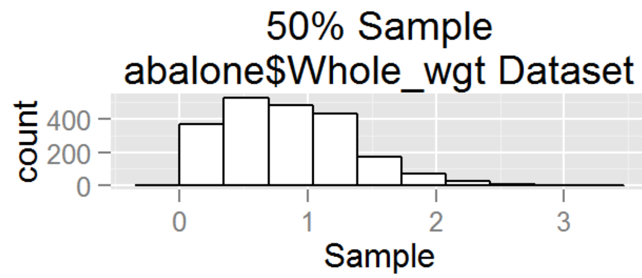
```
test <- function(dfcolumn,title='')
  {
  # test() accepts two parameters: dfcolumn - a column from a dataframe; and title- defau
lts to black and gives the user the option of having a subtitle

  # test() calls all of the previously created fucntions to solve the problems given on t
he test.  The variables below are created to store each question.
  p1 <- samples(dfcolumn)
  p1b <- sampleStatistics(p1)
  print(p1b)
  p2 <- list_hist_grid(p1,title)
  p3 <- sample1000(dfcolumn)
  p4 <- df_hist_grid(p3,dfcolumn,title)
  p5 <- sample1000Statistics(p3,dfcolumn,p1b)
  print(p5)
  p6 <- hist_six(p5,dfcolumn,title)
  p7 <- hist_seven(p5,dfcolumn,title)

  return(NULL)
  }
# we create 3 function calls to answer the problems on the test.

# answer problems 1-7
test(abalone$Whole_wgt,title="abalone$Whole_wgt Dataset")
```

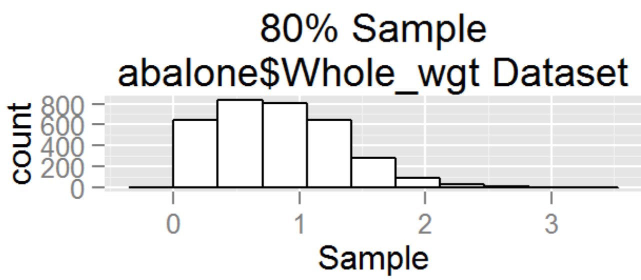
```
##                1% Sample 10% Sample 25% Sample 50% Sample 80% Sample
## SampleMin      0.0740000  0.0240000  0.0105000  0.0080000  0.0020000
## SampleMax      1.7405000  2.3235000  2.6570000  2.7795000  2.8255000
## SampleMean     0.9115595  0.8315120  0.8478075  0.8364001  0.8281205
## SampleVariance 0.2283977  0.2329197  0.2512489  0.2421862  0.2413725
##                100% Sample
## SampleMin        0.0020000
## SampleMax        2.8255000
## SampleMean       0.8288175
## SampleVariance   0.2405153
```

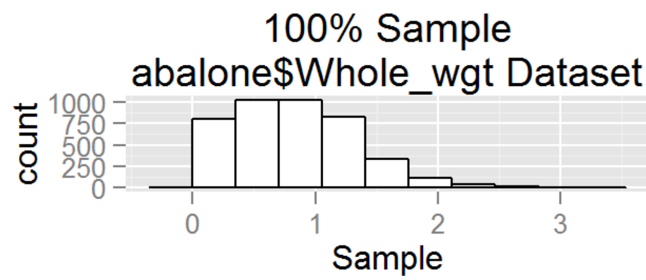

1% Sample
abalone$Whole_wgt Dataset
count
7.5
5.0
2.5
0.0
0.0
0.5
1.0
1.5
2.0
Sample

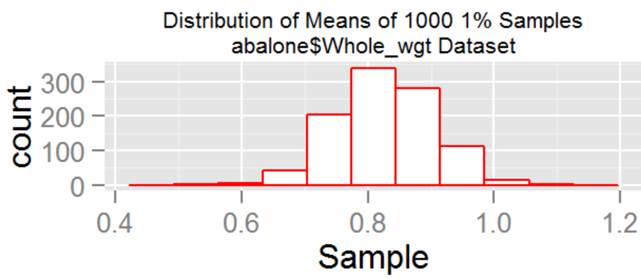

10% Sample
abalone$Whole_wgt Dataset
count
100
75
50
25
0
0
1
2
3
Sample

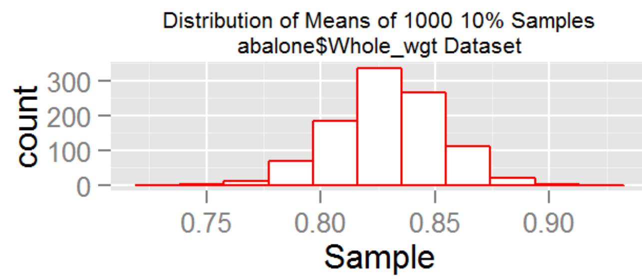

25% Sample
abalone$Whole_wgt Dataset
count
200
150
100
50
0
0
1
2
3
Sample

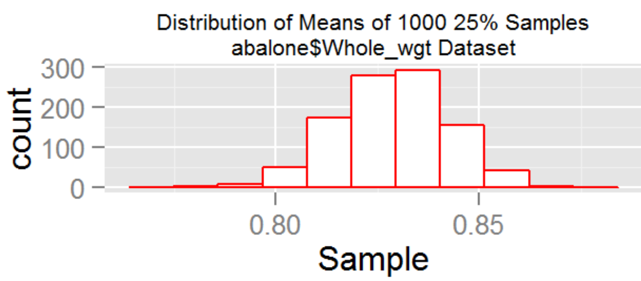

50% Sample
abalone$Whole_wgt Dataset
count
400
200
0
0
1
2
3
Sample

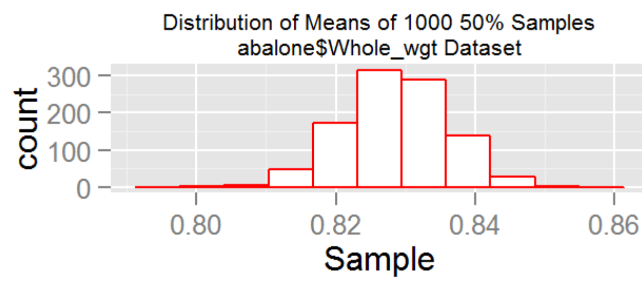

80% Sample
abalone$Whole_wgt Dataset
count
800
600
400
200
0
0
1
2
3
Sample

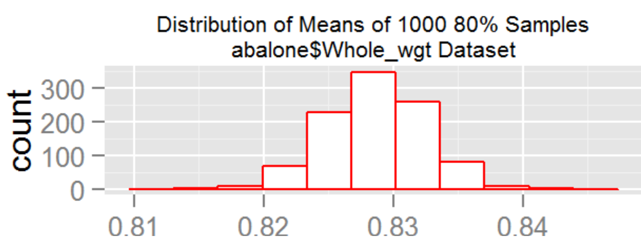

100% Sample
abalone$Whole_wgt Dataset
count
1000
750
500
250
0
0
1
2
3
Sample

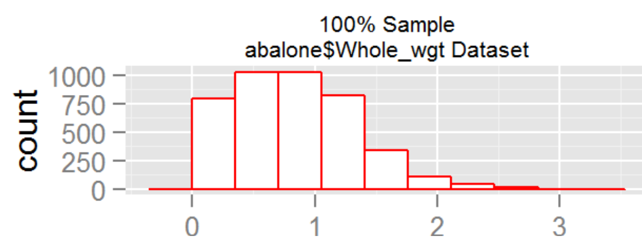

Distribution of Means of 1000 1% Samples
abalone$Whole_wgt Dataset
count
300
200
100
0
0.4
0.6
0.8
1.0
1.2
Sample


Distribution of Means of 1000 10% Samples
abalone$Whole_wgt Dataset
count
300
200
100
0
0.75
0.80
0.85
0.90
Sample

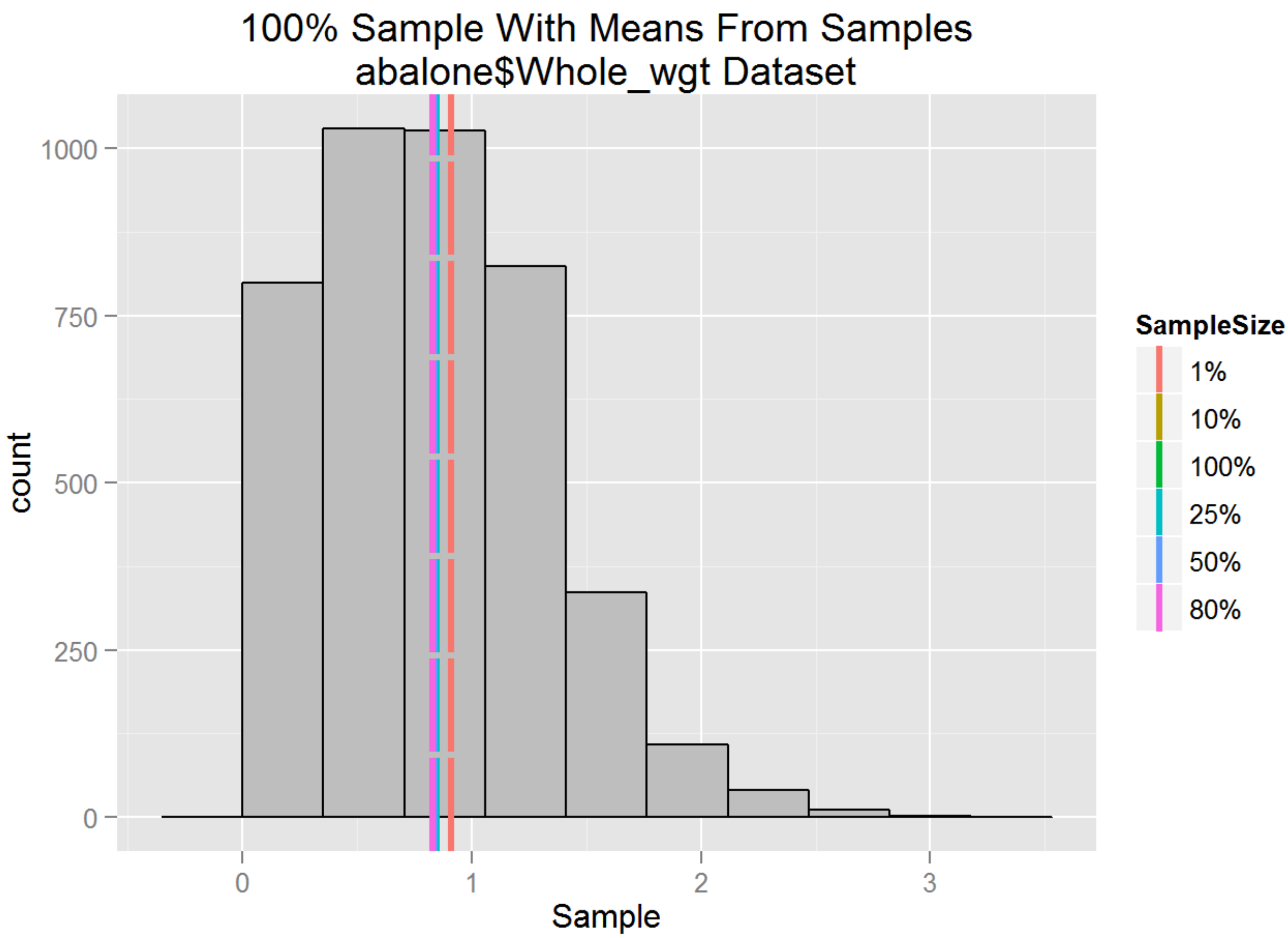

Distribution of Means of 1000 25% Samples
abalone$Whole_wgt Dataset
count
300
200
100
0
0.80
0.85
Sample

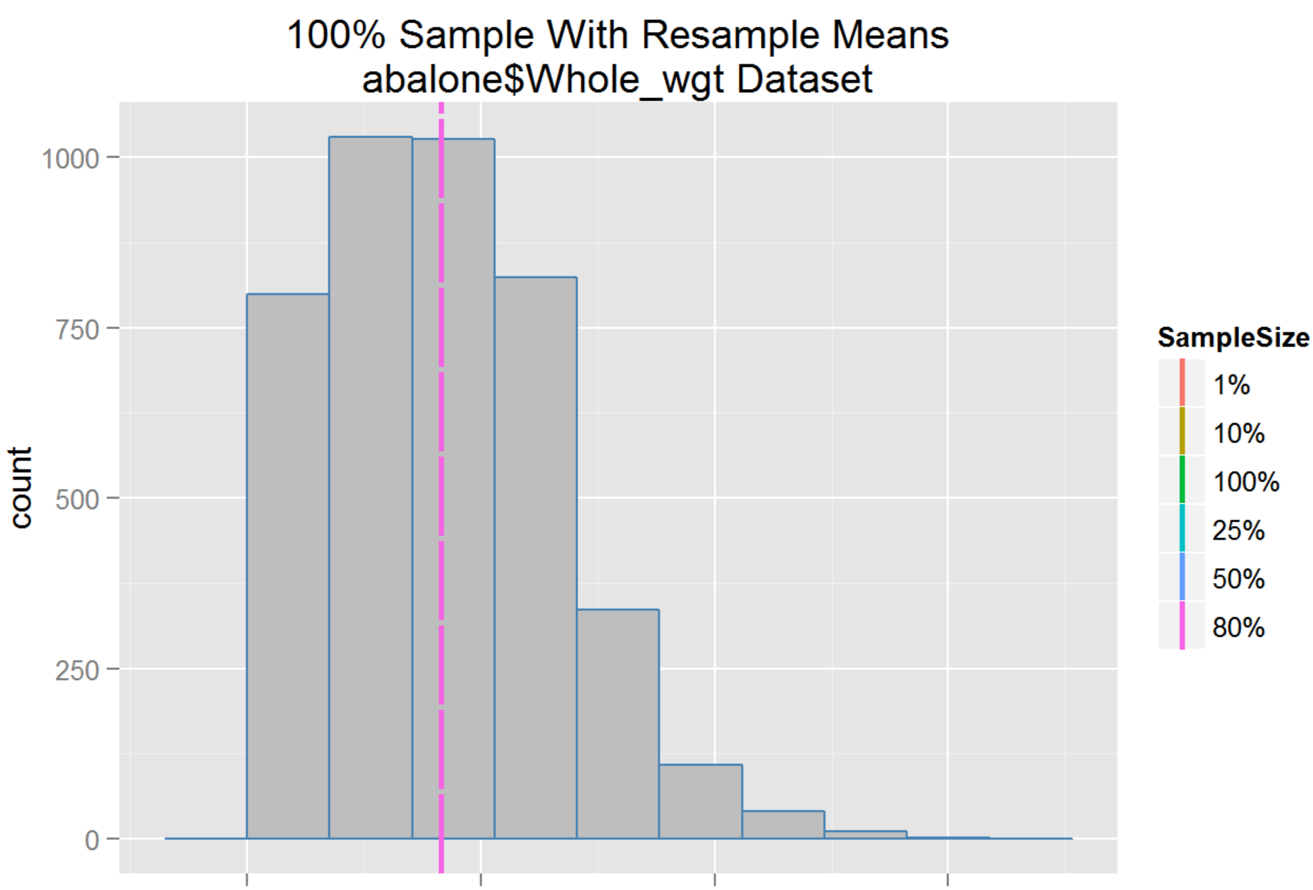

Distribution of Means of 1000 50% Samples
abalone$Whole_wgt Dataset
count
300
200
100
0
0.80
0.82
0.84
0.86
Sample


Distribution of Means of 1000 80% Samples
abalone$Whole_wgt Dataset
count
300
200
100
0
0.81
0.82
0.83
0.84


100% Sample
abalone$Whole_wgt Dataset
count
1000
750
500
250
0
0
1
2
3


```
##                      1% Sample  10% Sample   25% Sample   50% Sample
## SampleMin        0.074000000 0.024000000 0.0105000000 8.000000e-03
## SampleMax        1.740500000 2.323500000 2.6570000000 2.779500e+00
## SampleMean       0.911559524 0.831511962 0.8478074713 8.364001e-01
## SampleVariance   0.228397746 0.232919658 0.2512488843 2.421862e-01
## ResampleMin      0.532452381 0.744299043 0.7846350575 8.012898e-01
## ResampleMax      1.095226190 0.899714115 0.8719482759 8.523333e-01
## ResampleMean     0.827113976 0.829432012 0.8287349861 8.287016e-01
## ResampleVariance 0.005859794 0.000518082 0.0001844333 5.384583e-05
##                    80% Sample 100% Sample
## SampleMin        0.002000000   0.0020000
## SampleMax        2.825500000   2.8255000
## SampleMean       0.828120473   0.8288175
## SampleVariance   0.241372546   0.2405153
## ResampleMin      0.813998204   0.0020000
## ResampleMax      0.841442682   2.8255000
## ResampleMean     0.828642201   0.8288175
## ResampleVariance 0.000014097   0.2405153
```


100% Sample With Means From Samples
abalone$Whole_wgt Dataset
count
1000
750
500
250
0
0
1
2
3
Sample
SampleSize
1%
10%
100%
25%
50%
80%

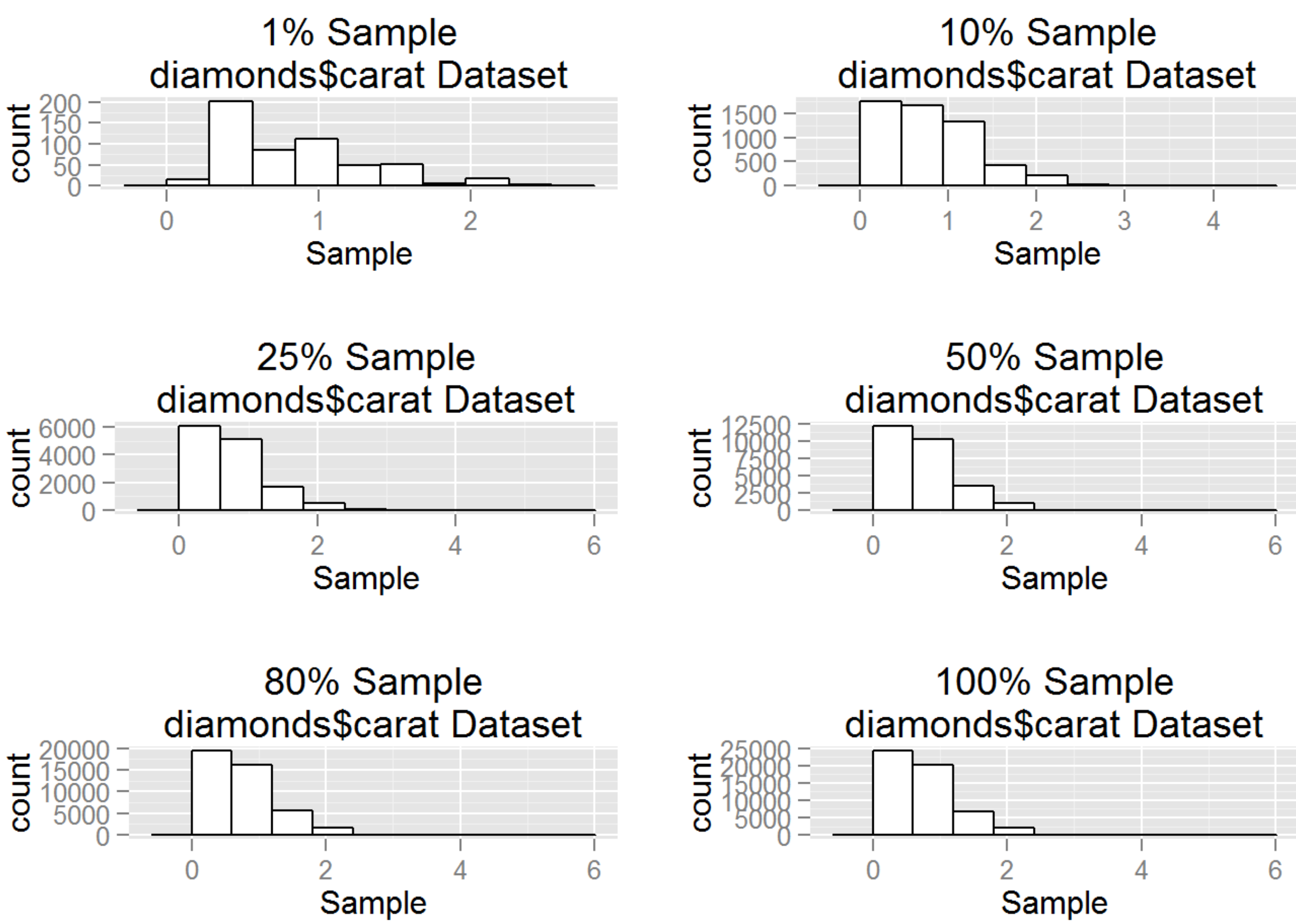

100% Sample With Resample Means
abalone$Whole_wgt Dataset
count
1000
750
500
250
0
0
1
2
3
SampleSize
1%
10%
100%
25%
50%
80%

Sample

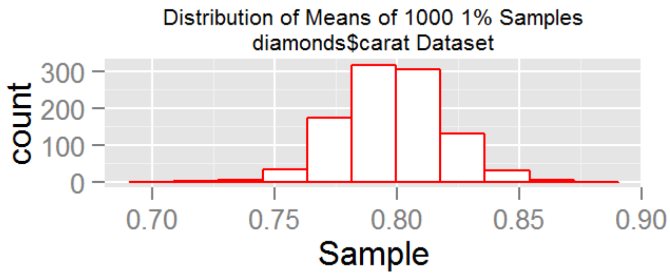
```
## NULL
```

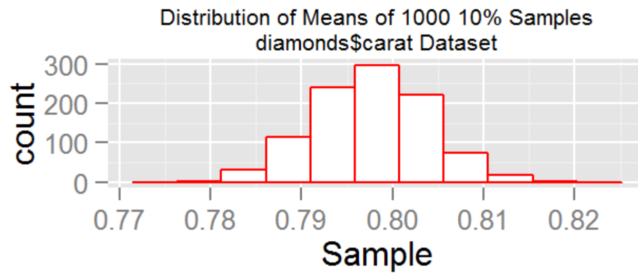
```
# answer the first part of 8
test(diamonds$carat,title="diamonds$carat Dataset")
```

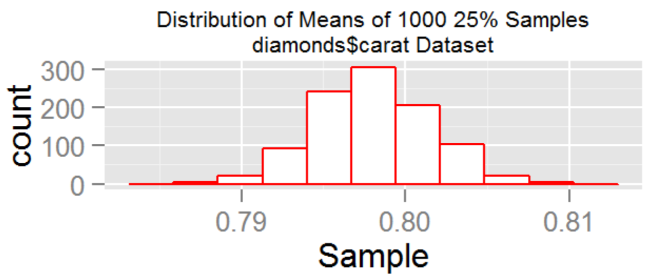
```
##                1% Sample 10% Sample 25% Sample 50% Sample 80% Sample
## SampleMin      0.2300000  0.2300000  0.2000000  0.2000000  0.2000000
## SampleMax      2.4800000  4.0000000  5.0100000  5.0100000  5.0100000
## SampleMean     0.8252319  0.8009251  0.8017078  0.7988732  0.7990360
## SampleVariance 0.2224428  0.2270332  0.2287023  0.2242071  0.2257007
##                100% Sample
## SampleMin        0.2000000
## SampleMax        5.0100000
## SampleMean       0.7979397
## SampleVariance   0.2246867
```

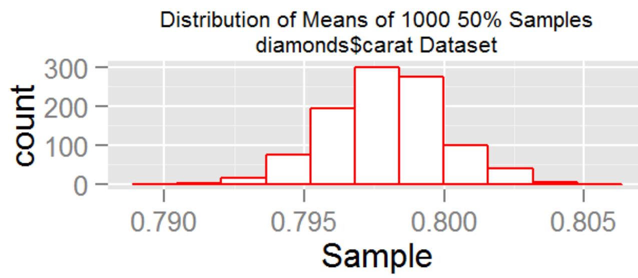


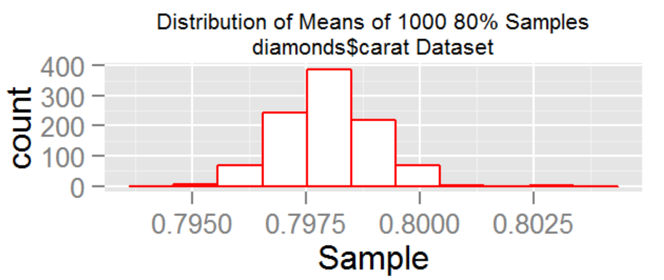

```
## Warning: position_stack requires constant width: output may be incorrect
```

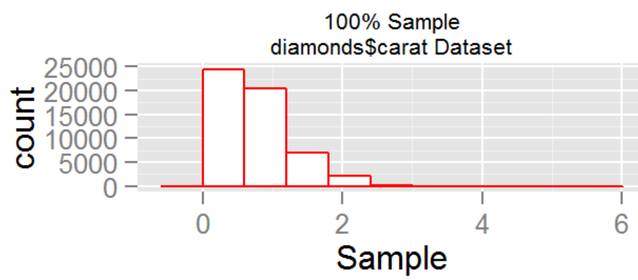




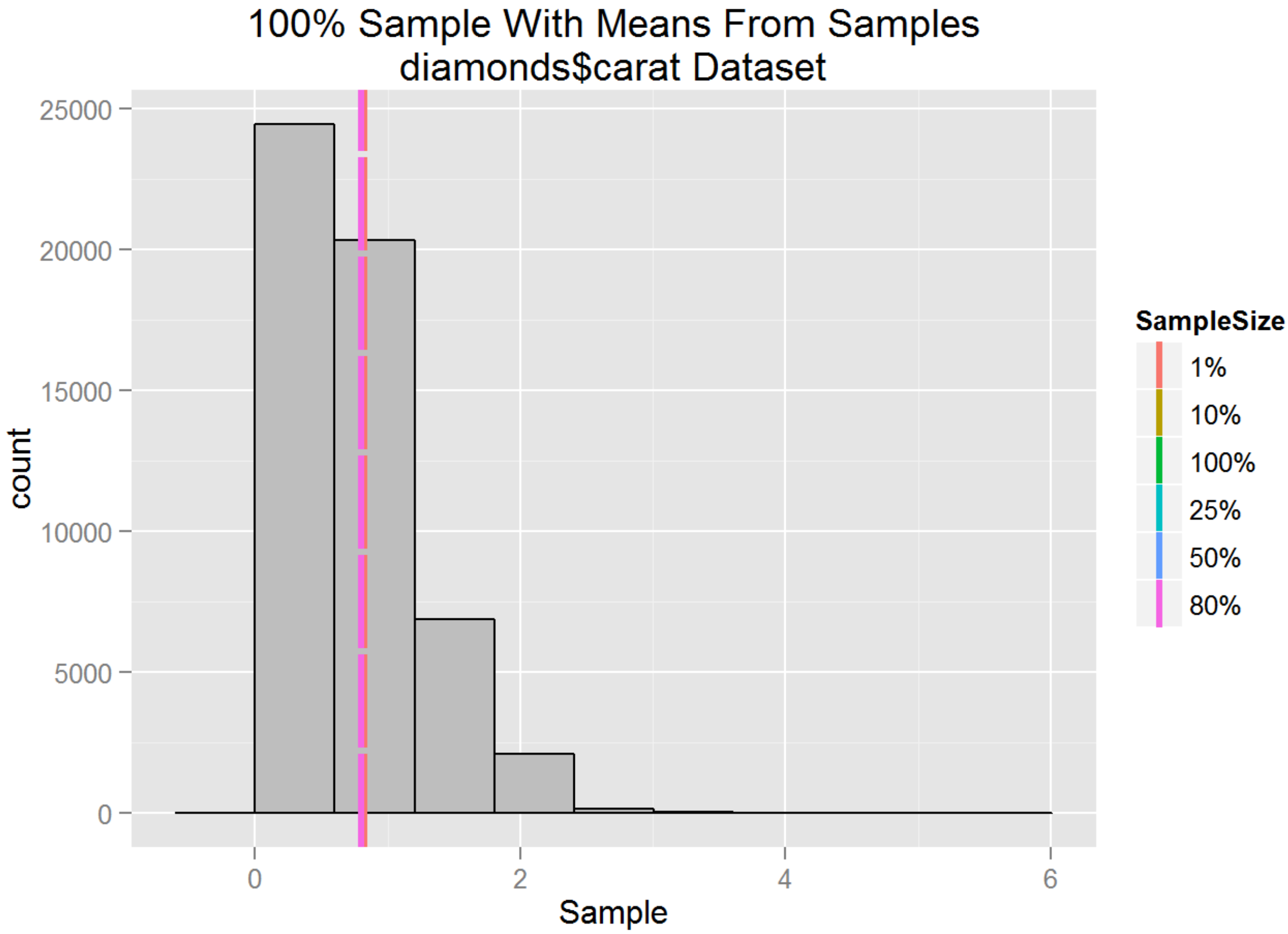


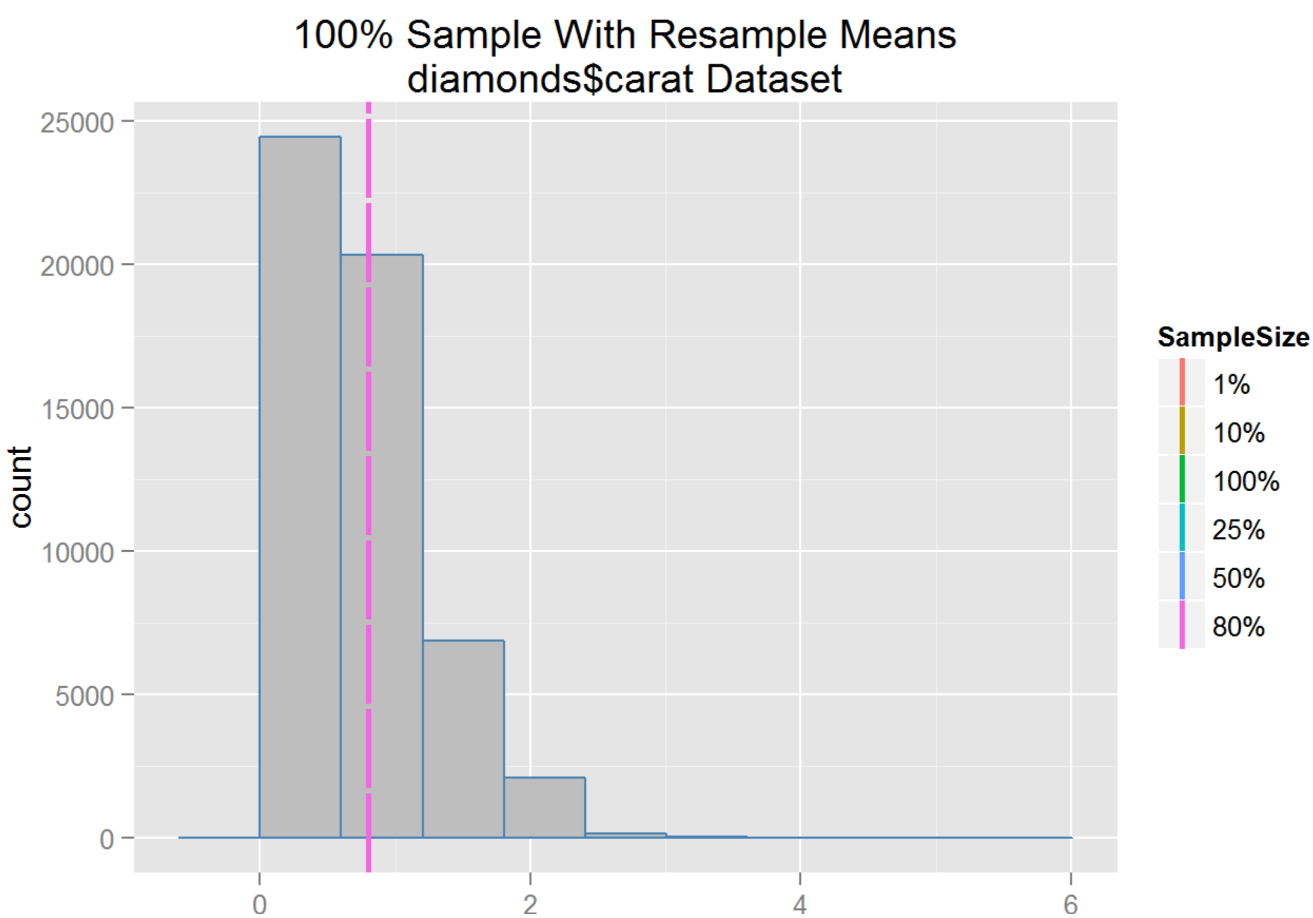







```
##                    1% Sample   10% Sample   25% Sample   50% Sample
## SampleMin        0.2300000000 2.300000e-01 2.000000e-01 2.000000e-01
## SampleMax        2.4800000000 4.000000e+00 5.010000e+00 5.010000e+00
## SampleMean       0.8252319109 8.009251e-01 8.017078e-01 7.988732e-01
## SampleVariance   0.2224428365 2.270332e-01 2.287023e-01 2.242071e-01
## ResampleMin      0.7224860853 7.766518e-01 7.868854e-01 7.919726e-01
## ResampleMax      0.8678478664 8.157119e-01 8.085643e-01 8.046960e-01
## ResampleMean     0.7979126902 7.975353e-01 7.980392e-01 7.979375e-01
## ResampleVariance 0.0004285421 3.878898e-05 1.249113e-05 4.043692e-06
##                   80% Sample 100% Sample
## SampleMin        2.000000e-01   0.2000000
## SampleMax        5.010000e+00   5.0100000
## SampleMean       7.990360e-01   0.7979397
## SampleVariance   2.257007e-01   0.2246867
## ResampleMin      7.949351e-01   0.2000000
## ResampleMax      8.027348e-01   5.0100000
## ResampleMean     7.979606e-01   0.7979397
## ResampleVariance 1.006025e-06   0.2246867
```


100% Sample With Means From Samples
diamonds$carat Dataset
25000
20000
15000
10000
5000
0
count
0
2
4
6
Sample
SampleSize
1%
10%
100%
25%
50%
80%

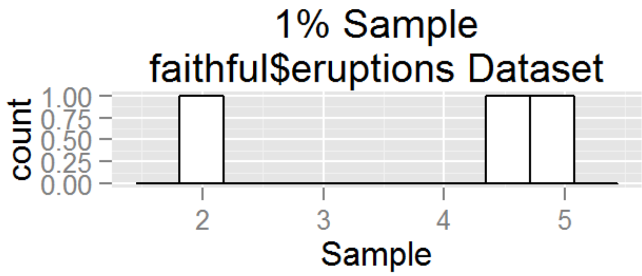

100% Sample With Resample Means
diamonds$carat Dataset
25000
20000
15000
10000
5000
0
count
0
2
4
6
SampleSize
1%
10%
100%
25%
50%
80%

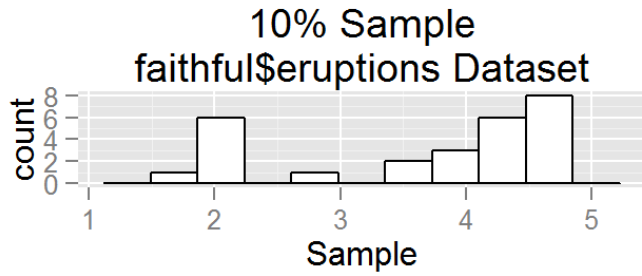
Sample

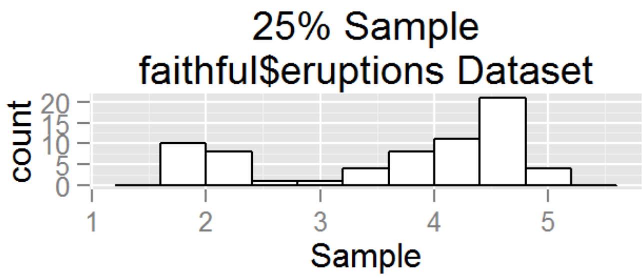
```
## NULL
```

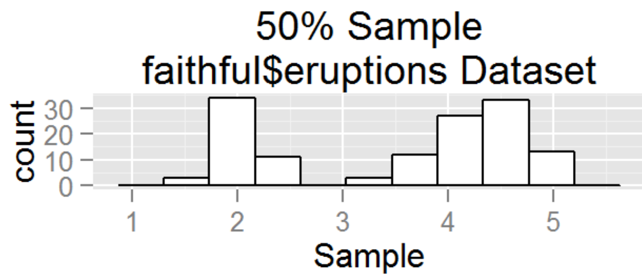
```
# answer the last part of 8
test(faithful$eruptions,title="faithful$eruptions Dataset")
```

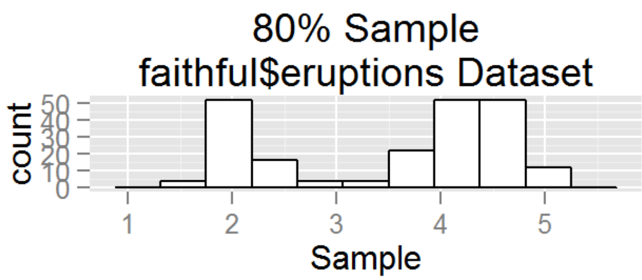
```
##                1% Sample 10% Sample 25% Sample 50% Sample 80% Sample
## SampleMin       1.833000   1.833000   1.733000   1.600000   1.600000
## SampleMax       4.733000   4.817000   4.933000   5.067000   5.100000
## SampleMean      3.644333   3.640148   3.637956   3.479515   3.518638
## SampleVariance  2.494185   1.146245   1.193931   1.390843   1.288403
##                100% Sample
## SampleMin         1.600000
## SampleMax         5.100000
## SampleMean        3.487783
## SampleVariance    1.302728
```

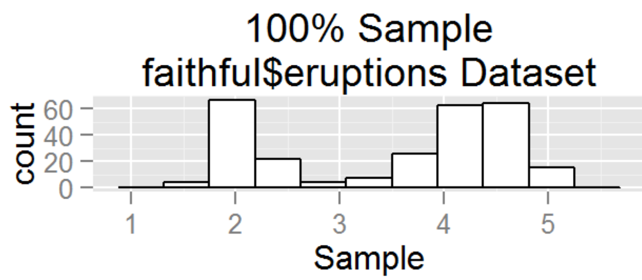

1% Sample
faithful$eruptions Dataset
count
1.00
0.75
0.50
0.25
0.00
2
3
4
5
Sample

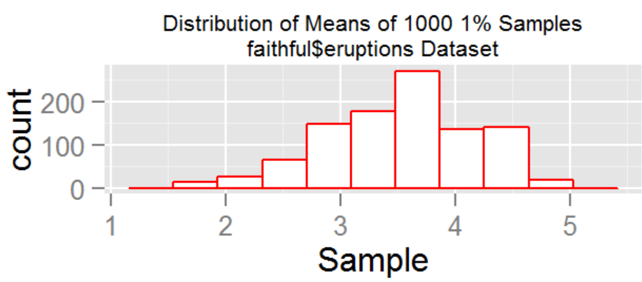

10% Sample
faithful$eruptions Dataset
count
8
6
4
2
0
1
2
3
4
5
Sample

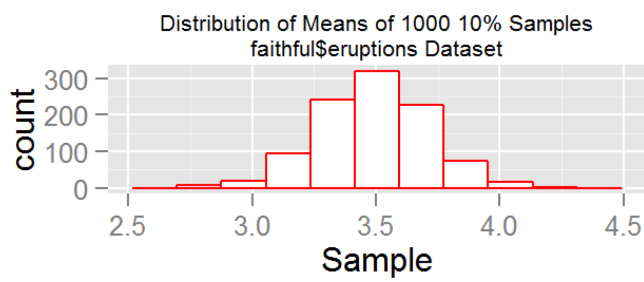

25% Sample
faithful$eruptions Dataset
count
20
15
10
5
0
1
2
3
4
5
Sample

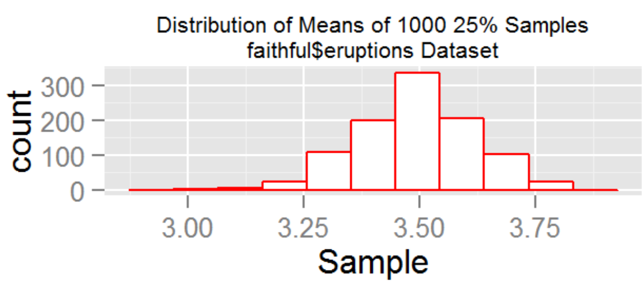

50% Sample
faithful$eruptions Dataset
count
30
20
10
0
1
2
3
4
5
Sample

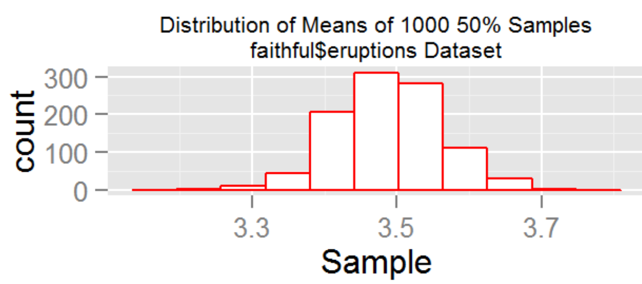

80% Sample
faithful$eruptions Dataset
count
50
40
30
20
10
0
1
2
3
4
5
Sample

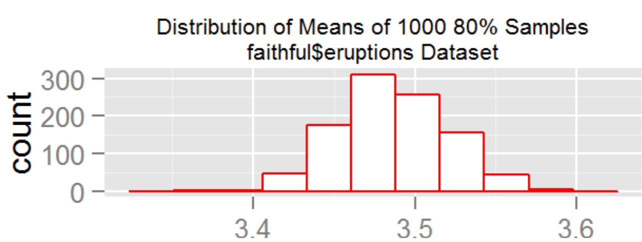

100% Sample
faithful$eruptions Dataset
count
60
40
20
0
1
2
3
4
5
Sample

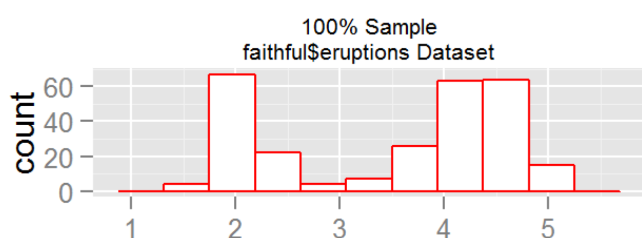

Distribution of Means of 1000 1% Samples
faithful$eruptions Dataset
count
200
100
0
1
2
3
4
5
Sample


Distribution of Means of 1000 10% Samples
faithful$eruptions Dataset
count
300
200
100
0
2.5
3.0
3.5
4.0
4.5
Sample

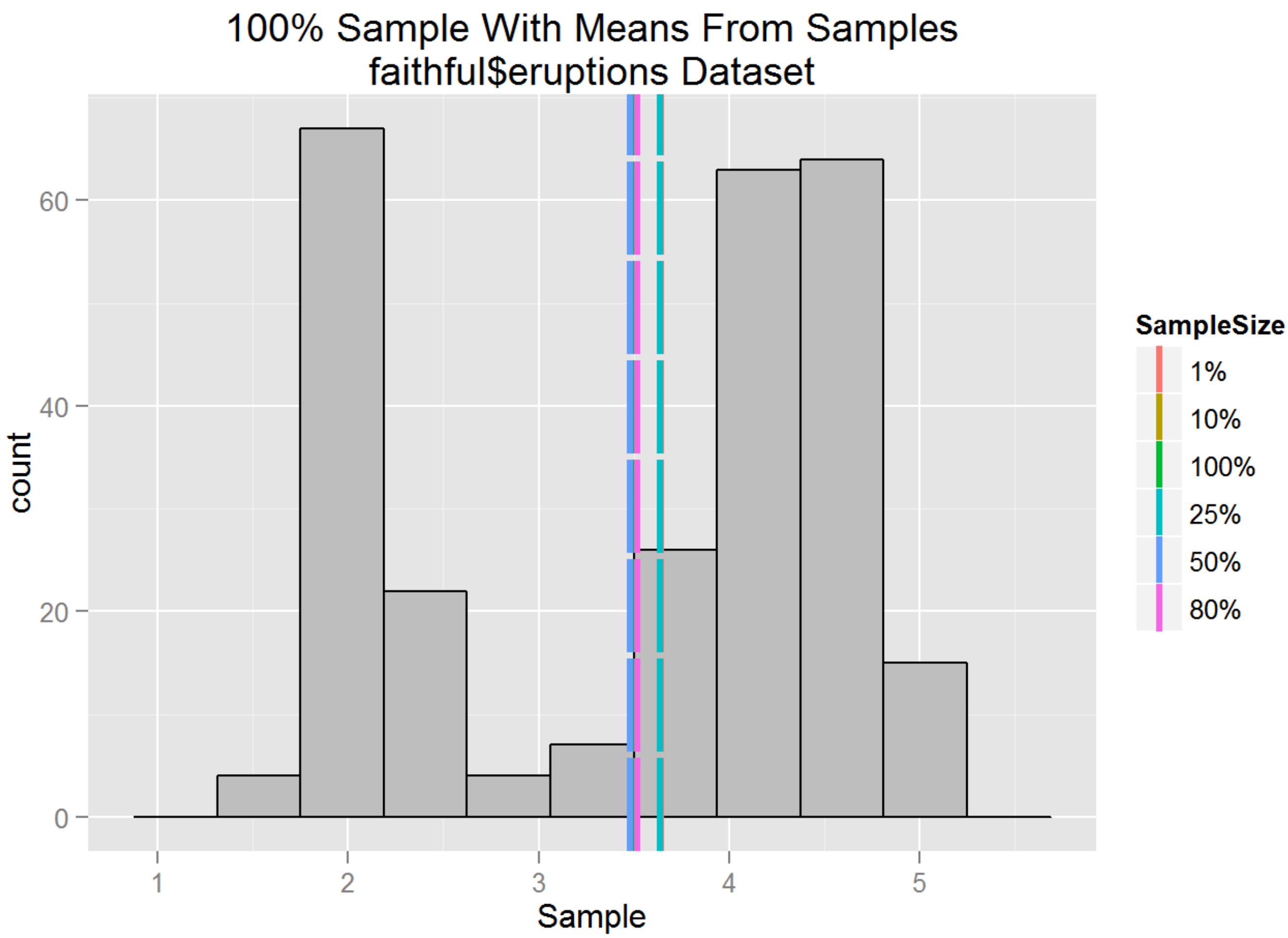

Distribution of Means of 1000 25% Samples
faithful$eruptions Dataset
count
300
200
100
0
3.00
3.25
3.50
3.75
Sample

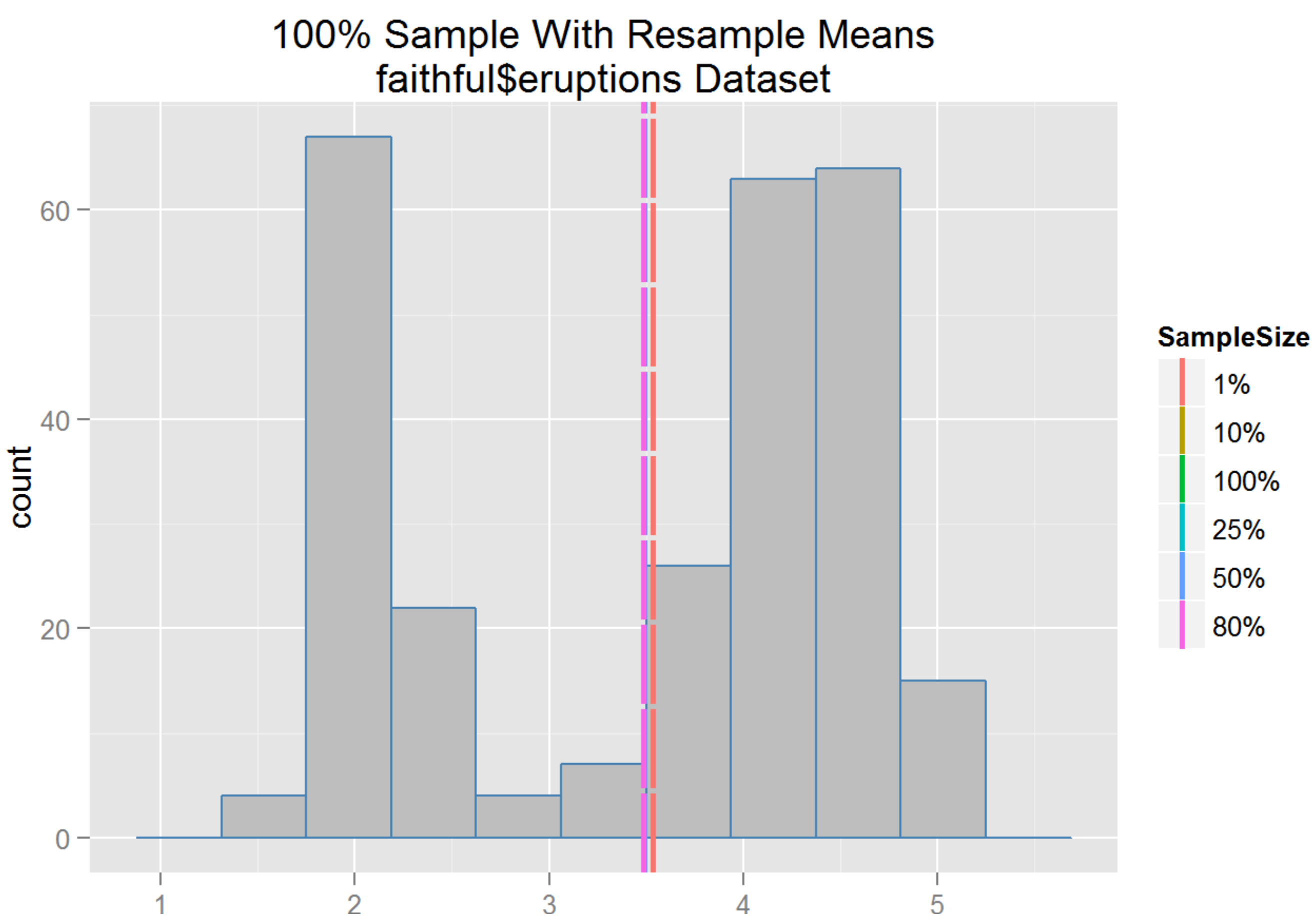

Distribution of Means of 1000 50% Samples
faithful$eruptions Dataset
count
300
200
100
0
3.3
3.5
3.7
Sample


Distribution of Means of 1000 80% Samples
faithful$eruptions Dataset
count
300
200
100
0
3.4
3.5
3.6


100% Sample
faithful$eruptions Dataset
count
60
40
20
0
1
2
3
4
5


```
##                  1% Sample 10% Sample 25% Sample  50% Sample  80% Sample
## SampleMin        1.8330000 1.83300000 1.73300000 1.600000000 1.600000000
## SampleMax        4.7330000 4.81700000 4.93300000 5.067000000 5.100000000
## SampleMean       3.6443333 3.64014815 3.63795588 3.479514706 3.518637615
## SampleVariance   2.4941853 1.14624544 1.19393079 1.390842829 1.288403237
## ResampleMin      1.8166667 2.70744444 3.06425000 3.205588235 3.371541284
## ResampleMax      4.9110000 4.14500000 3.83033824 3.697169118 3.591252294
## ResampleMean     3.5329643 3.48747378 3.49339681 3.489055522 3.486801821
## ResampleVariance 0.4225664 0.04699163 0.01489637 0.005223422 0.001142753
##                  100% Sample
## SampleMin           1.600000
## SampleMax           5.100000
## SampleMean          3.487783
## SampleVariance      1.302728
## ResampleMin         1.600000
## ResampleMax         5.100000
## ResampleMean        3.487783
## ResampleVariance    1.302728
```


100% Sample With Means From Samples
faithful$eruptions Dataset
count
Sample
SampleSize
1%
10%
100%
25%
50%
80%


100% Sample With Resample Means
faithful$eruptions Dataset
count
SampleSize
1%
10%
100%
25%
50%
80%

Sample

```
## NULL
```

Question (9) - The test asks us to provide a brief explanation of what we observed in the measures and histograms as it relates to the sample size and resampling.

For the first grid of histograms, where we were taking single samples of 1%,10%,25%, and 80%, we saw that the distriubtions of each sample became closer and closer to the actual distribution (shown by the 100% sample) as the sample size increased.

For the second second grid of histomgrams, where we were plotting the means of 1000 samples, we saw that as the sample size increased, the distributions of the means became increasing normal (as in the normal distribution).

Finally, on the histograms created in problems 6 and 7, we see that the means of the samples are kind of close to the actual mean but the Resample means are approxmimately equal to the actual mean.