MSAN 593 - Homework 2

Andre Guimaraes Duarte

July 29, 2016

```
knitr::opts_chunk$set(echo = TRUE)
```

Question 1

1.1

1.1.1

```
myRunifVec <- runif(10000000, 4, 6)
hist(myRunifVec,
     main = paste("Histogram of ", length(myRunifVec), "\n random variables ~ U(4, 6)",
                  sep = ""), xlab = "x", freq = F)
```

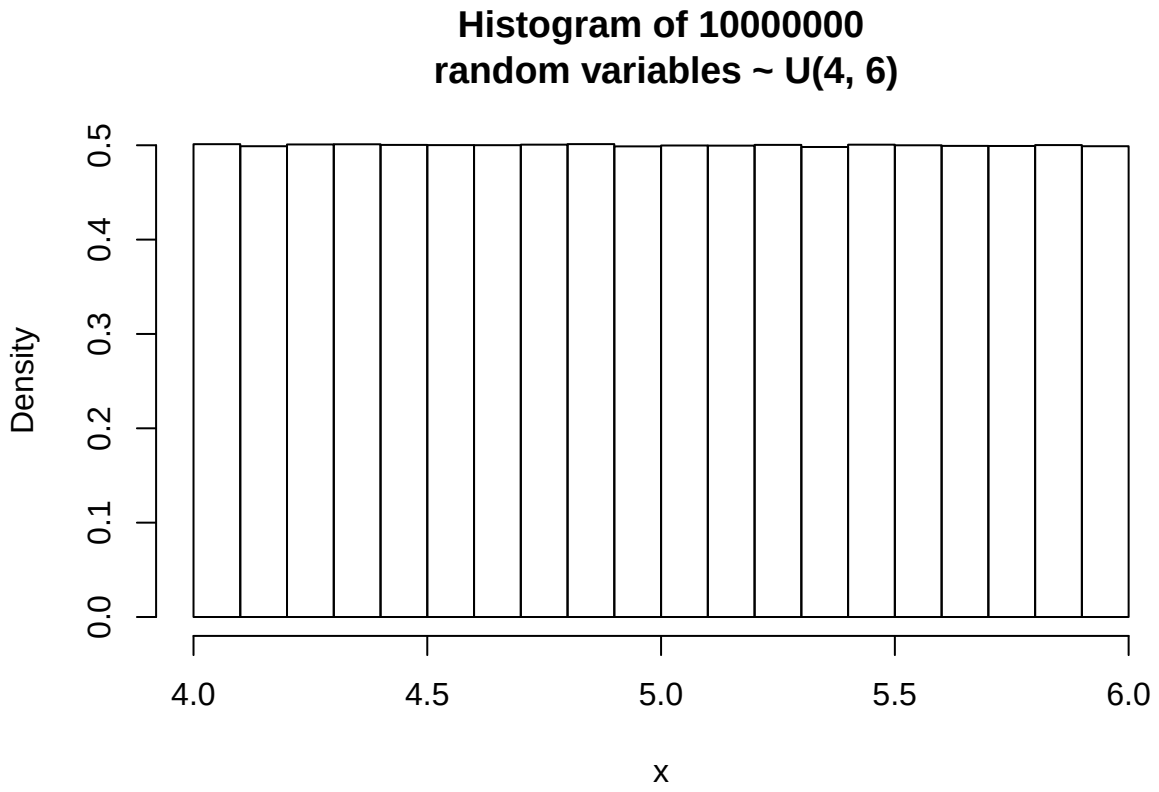

Figure 1: Histogram of 10,000,000 random variables ~ U(4, 6)

The histogram in figure 1 is the histogram of a $\sim U(4, 6)$ distribution with 10000000 random variables.

1.1.2

```
samples <- 100000
myRunifSample <- sample(myRunifVec, 100000)
hist(myRunifSample,
     main = paste("Histogram of ", length(myRunifSample),
                  "\n random variables sampled from a ~ U(4, 6)",
                  sep = ""), xlab = "x", freq = F)
```

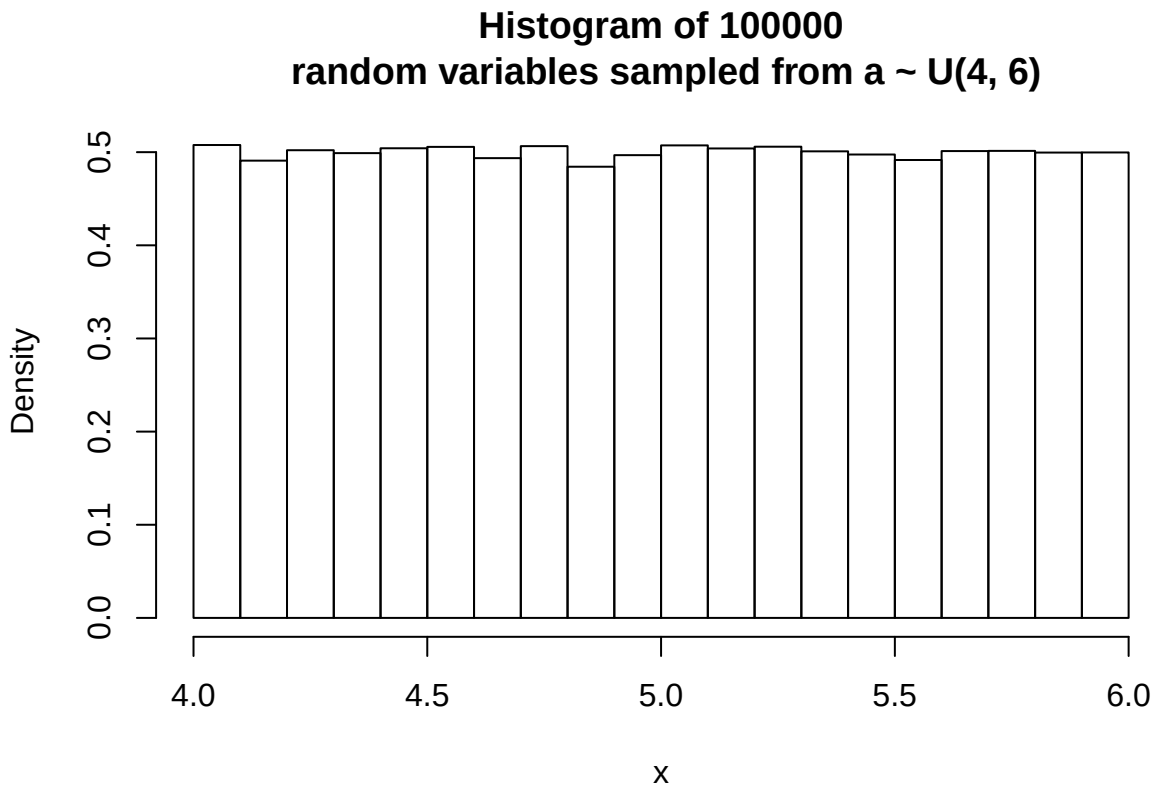

Figure 2: Histogram of a sample of 100,000 random elements from a ~U(4, 6) distribution

The histogram in figure 2 is very similar to the population distribution 1. Sampling elements from a uniform distribution maintains its original distribution.

1.1.3

Disclaimer: I first did this assignment in its entirety using `for` loops. My computer handled up to 1,000 samples fairly OK, but once the number of samples reached a few tens of thousands, it was unbearably slow to take the samples and means. I figured that `for` loops were not the way to go for this exercise, and did some research of some vectorized alternatives. I found that using the `apply` method was conveninent in this case, and indeed the results are much, **much** faster.

First, I define a data frame called `myRunifDataFrame` that has 30 columns. Each column will have `100,000` rows of single random samples from the vector `myRunifVec`. To do 30 repetitions of sampling 100,000 random variables, I use the `replicate` method. This does not simply copy the results from one column to the other, but indeed each column will have a separate sample of 100,000 random variables from the original distribution. `replicate` replicates the function, in this case `sample`.

Then, I define a second data frame, called `myRunifMeans` that contains the means of two, five, ten, and thirty elements from the original distribution. By using the `apply` method, I can define which columns (how many) of `myRunifDataFrame` to use in order to compute the mean. This takes only a couple of seconds to run, opposed to many minutes by using the previous method (`for` loops) and is much more efficient.

```
myRunifDataFrame <- data.frame(replicate(30, sample(myRunifVec, 100000)))
myRunifMeans <- data.frame("Means2"= apply(myRunifDataFrame[,1:2], 1, mean),
                           "Means5"= apply(myRunifDataFrame[,1:5], 1, mean),
                           "Means10"= apply(myRunifDataFrame[,1:10], 1, mean),
                           "Means30"= apply(myRunifDataFrame[,1:30], 1, mean))
```

```
hist(myRunifMeans$Means2, main="Histogram of the average of two elements from a ~Unif(4, 6)",
     xlab="x", freq = F)
```

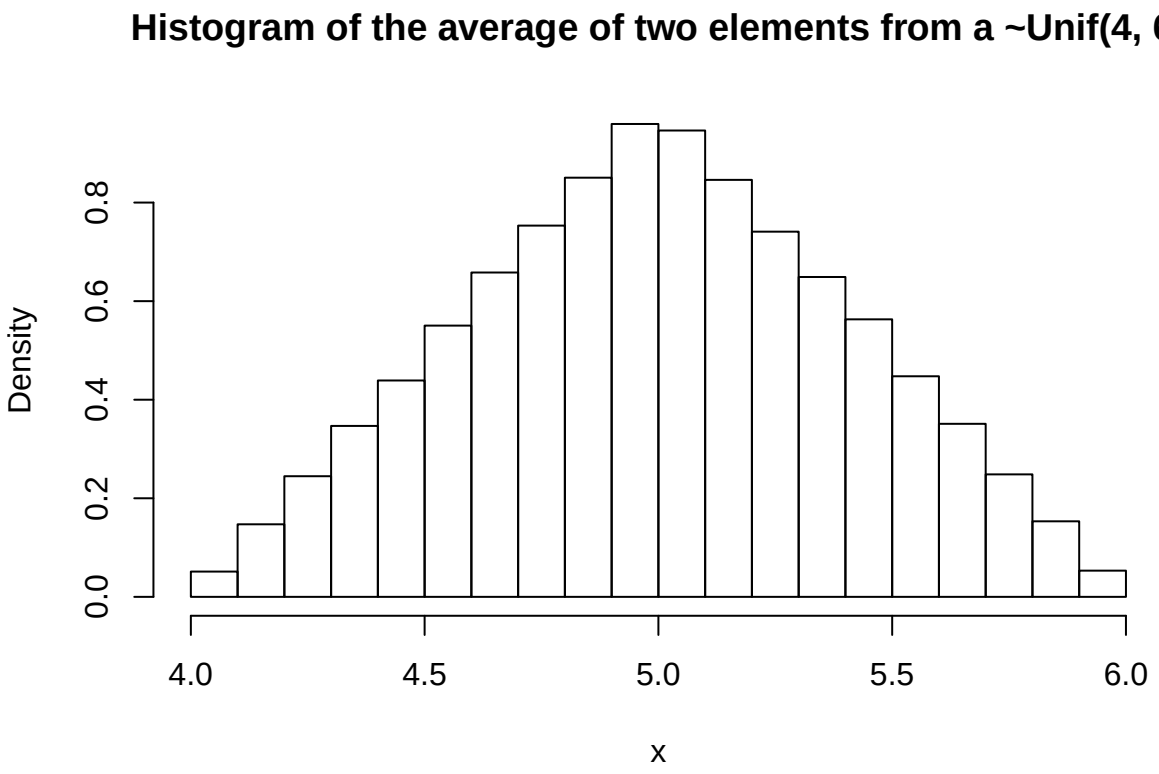

Figure 3: Histogram of 100,000 random means of two elements from a ~U(4, 6) distribution

The histogram in figure 3 is not similar to the population distribution shown in figure 1. In fact, the distribution seems very triangular.

1.1.4

```
hist(myRunifMeans$Means5, main="Histogram of the average of five elements from a ~Unif(4, 6)",
     xlab="x", freq = F)
```

The histogram in figure 4 is different from the population distribution in figure 1. There are more observations around 5, and less toward the tails. It is different than the previous histogram (figure 3): the distribution is less triangular.

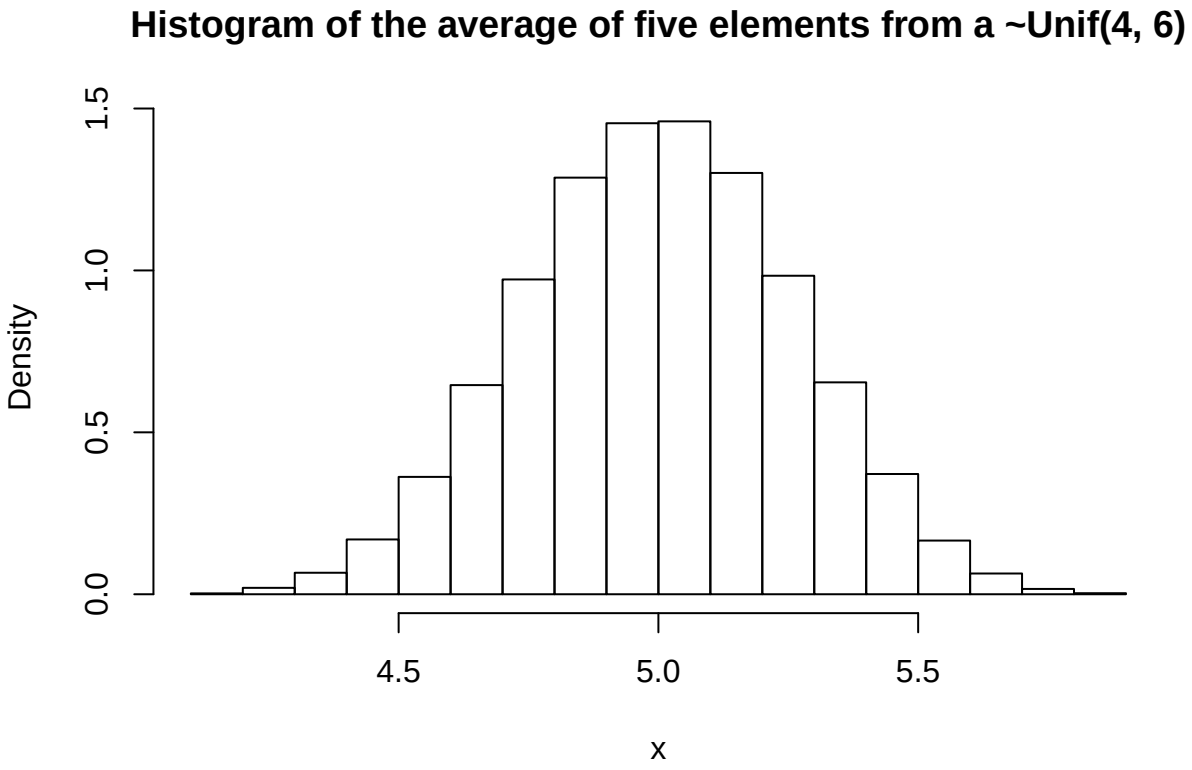

Figure 4: Histogram of 100,000 random means of five elements from a ~U(4, 6) distribution

1.1.5

```
hist(myRunifMeans$Means10, main="Histogram of the average of ten elements from a ~Unif(4, 6)",
     xlab="x", freq = F)
```

The histogram in figure 5 is different from the population distribution in figure 1. This one is bell-shaped and seems to be symmetrical around the value 5.

1.1.6

```
hist(myRunifMeans$Means30, main="Histogram of the average of thirty elements from a ~Unif(4, 6)",
     xlab="x", freq = F)
```

The histogram in figure 6 is different from the population distribution in figure 1. It looks a lot like a normal distribution centered around 5.

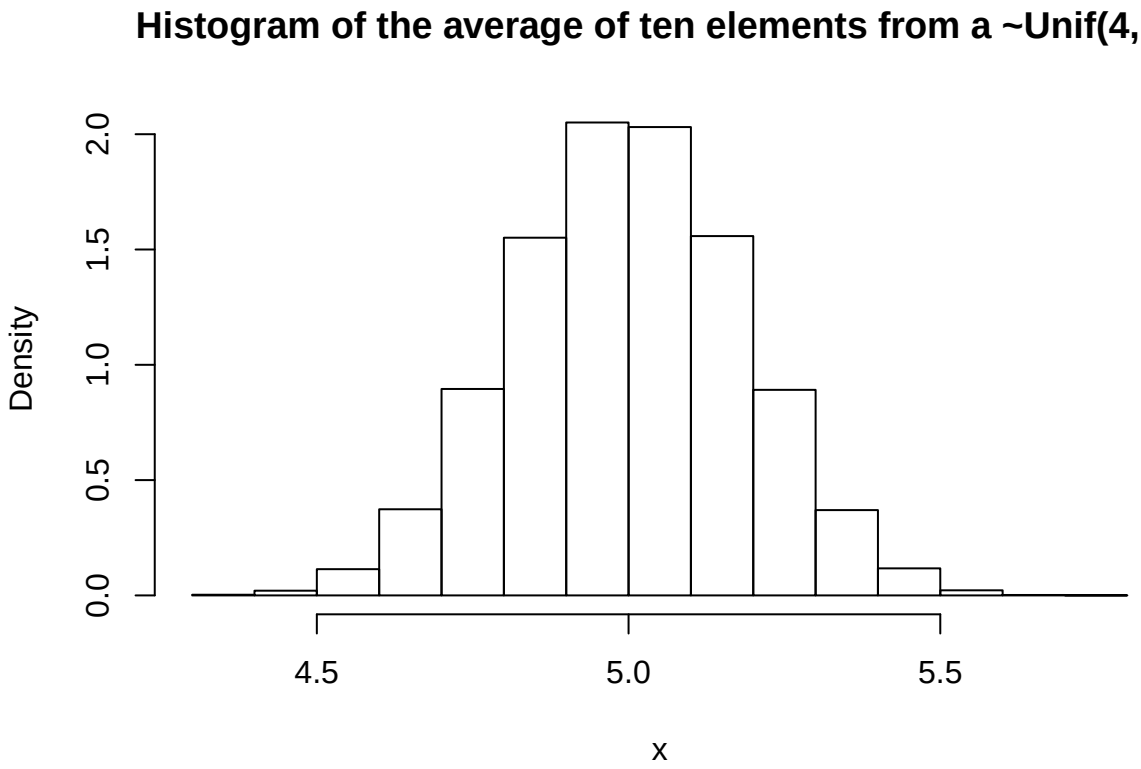

Figure 5: Histogram of 100,000 random means of ten elements from a ~U(4, 6) distribution

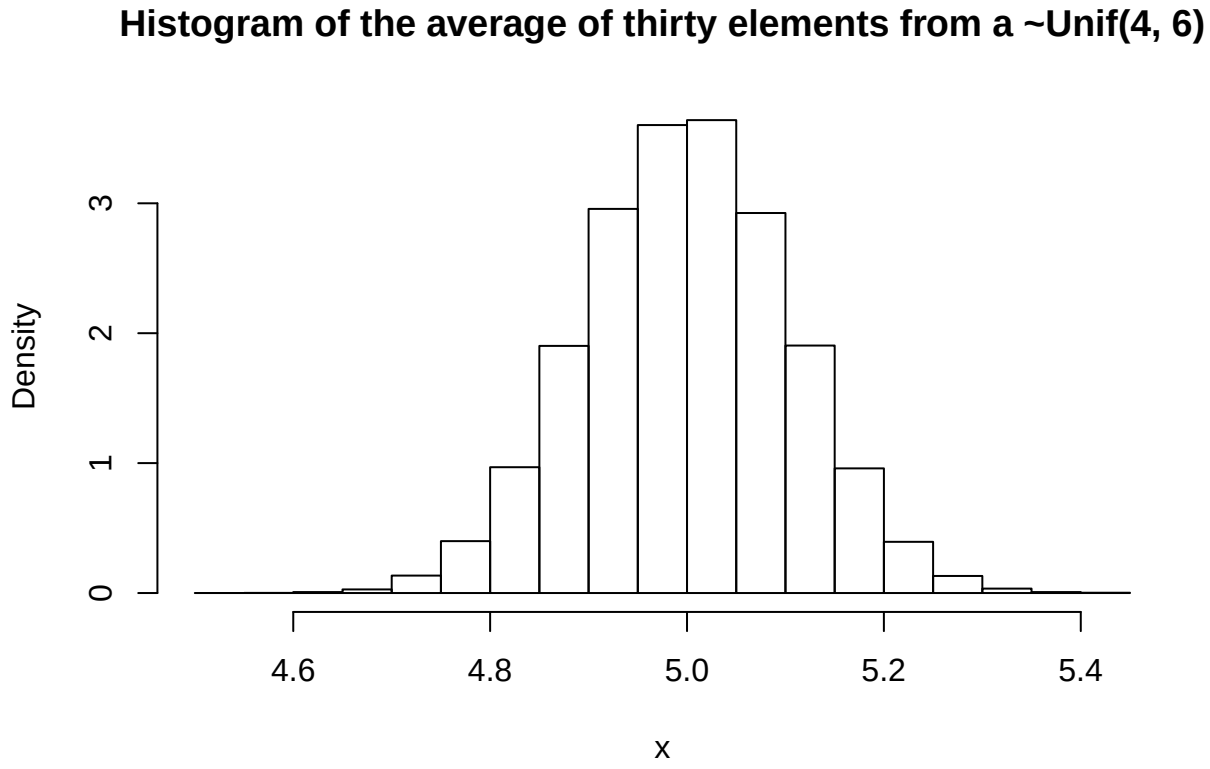

Figure 6: Histogram of 100,000 random means of thirty elements from a ~U(4, 6) distribution

1.2

1.2.1

```
myRexpVec <- rexp(10000000, 0.5)
hist(myRexpVec,
     main = paste("Histogram of ", length(myRexpVec), "\n random variables ~ Exp(0.5)",
                  sep = ""), xlab = "x", freq = F)
```

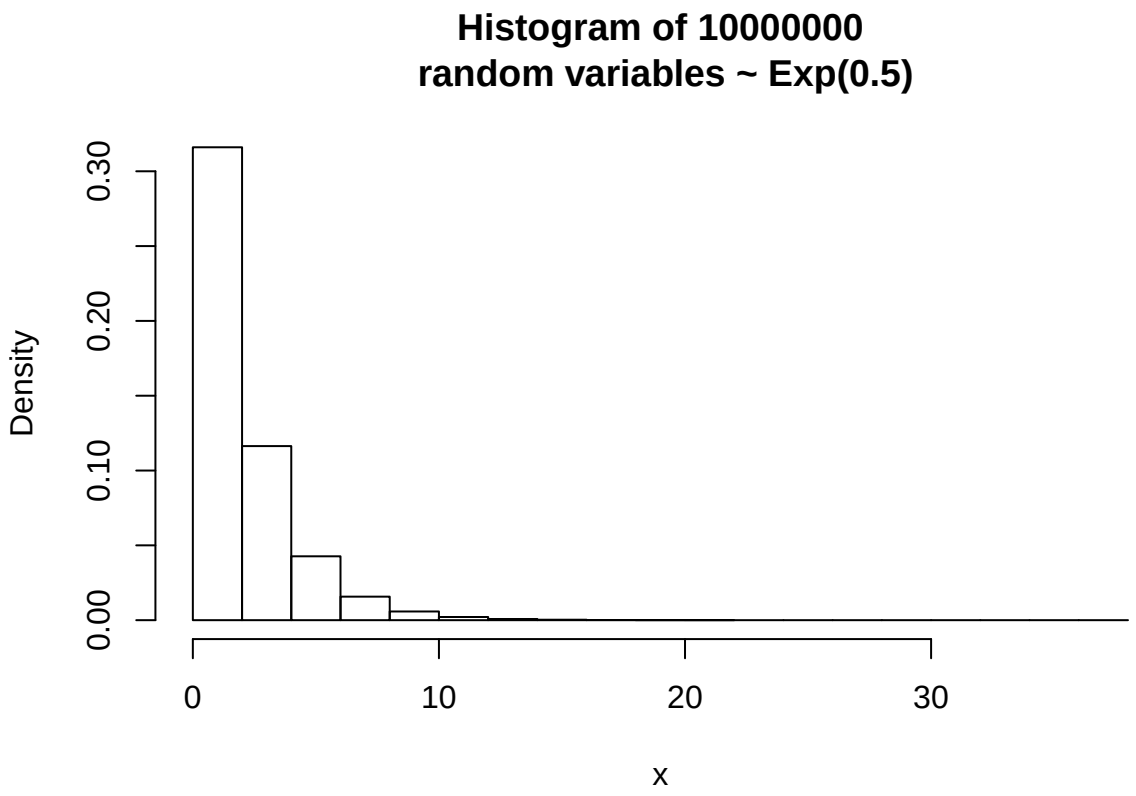

Figure 7: Histogram of 10,000,000 random variables ~Exp(0.5)

Figure 7 shows the histogram of a $\sim Exp(0.5)$ distribution with 10000000 random variables.

1.2.2

```
myRexpSample <- sample(myRexpVec, 100000)
hist(myRexpSample,
     main = paste("Histogram of ", length(myRexpSample),
                  "\n random variables sampled from a ~ Exp(0.5)",
                  sep = ""), xlab = "x", freq = F)
```

The histogram in figure 8 is very similar to the population distribution 7. Sampling elements from a negative exponential distribution maintains its original distribution.

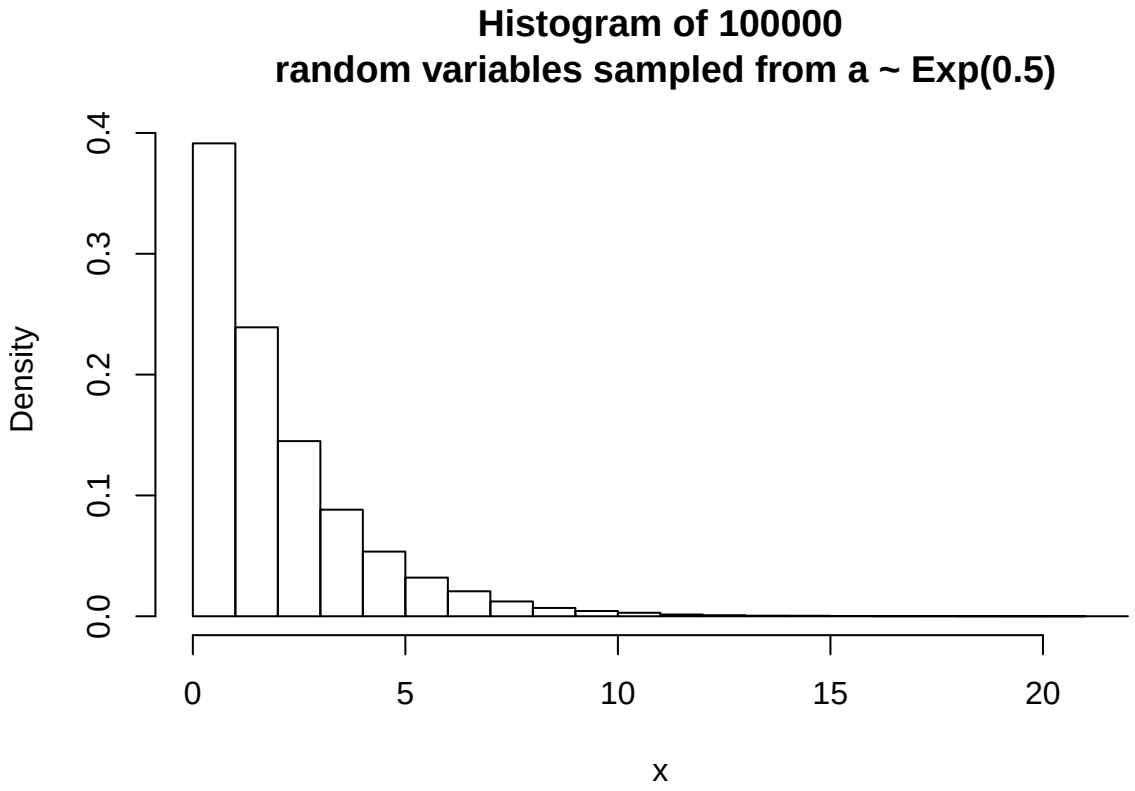

Figure 8: Histogram of 100,000 random elements from a ~Exp(0.5) distribution

1.2.3

```
myRexpDataFrame <- data.frame(replicate(30, sample(myRexpVec, 100000)))
myRexpMeans <- data.frame("Means2"= apply(myRexpDataFrame[,1:2], 1, mean),
                          "Means5"= apply(myRexpDataFrame[,1:5], 1, mean),
                          "Means10"= apply(myRexpDataFrame[,1:10], 1, mean),
                          "Means30"= apply(myRexpDataFrame[,1:30], 1, mean))
```

```
hist(myRexpMeans$Means2, main="Histogram of the average of two elements from a ~Exp(0.5)",
     xlab="x", freq = F)
```

Figure 9 shows the histogram is slightly more skewed to the right than the population distribution in figure 7. In fact, this is a Gamma distribution with parameter $\alpha = 2 * 0.5 = 1$.

1.2.4

```
hist(myRexpMeans$Means5, main="Histogram of the average of five elements from a ~Exp(0.5)",
     xlab="x", freq = F)
```

The histogram when we take the mean of five elements from the original population, shown in figure 10, is very different from the histogram of the population distribution in figure 7. It looks like the values are slowly distributing themselves around 2, but with a longer tail to the right.

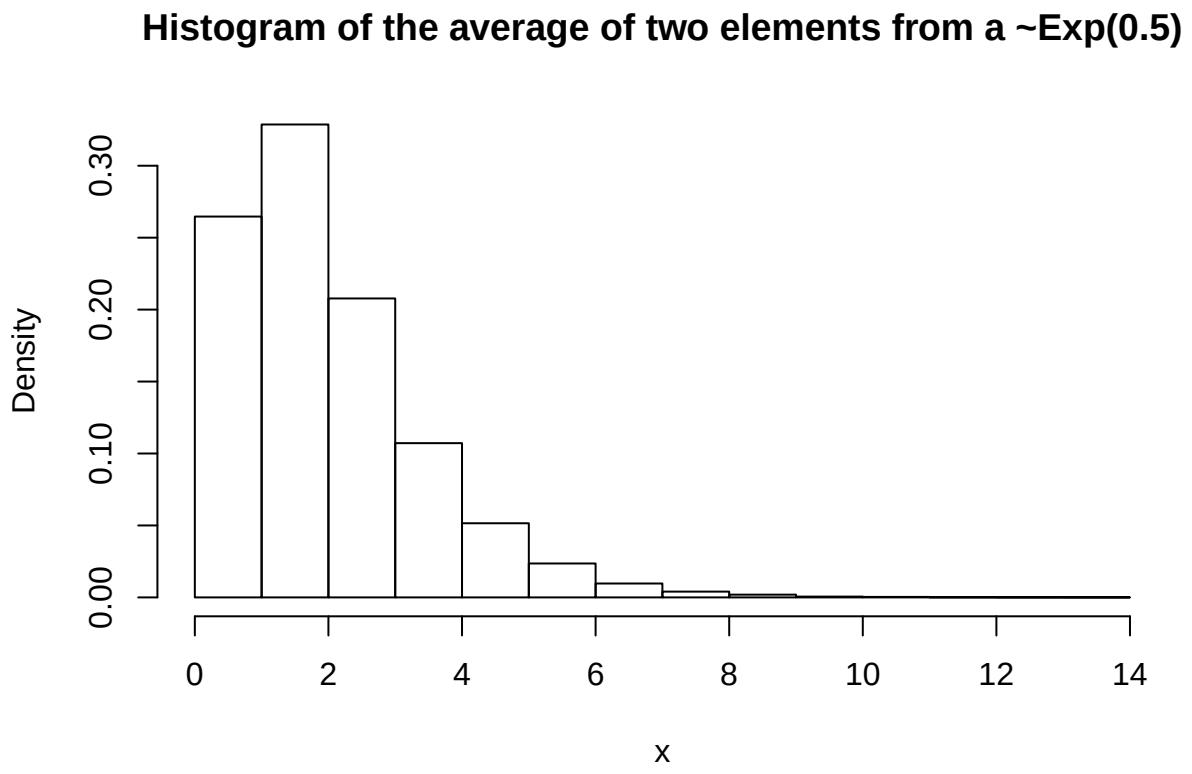

Figure 9: Histogram of 100,000 random means of two elements from a ~Exp(0.5) distribution

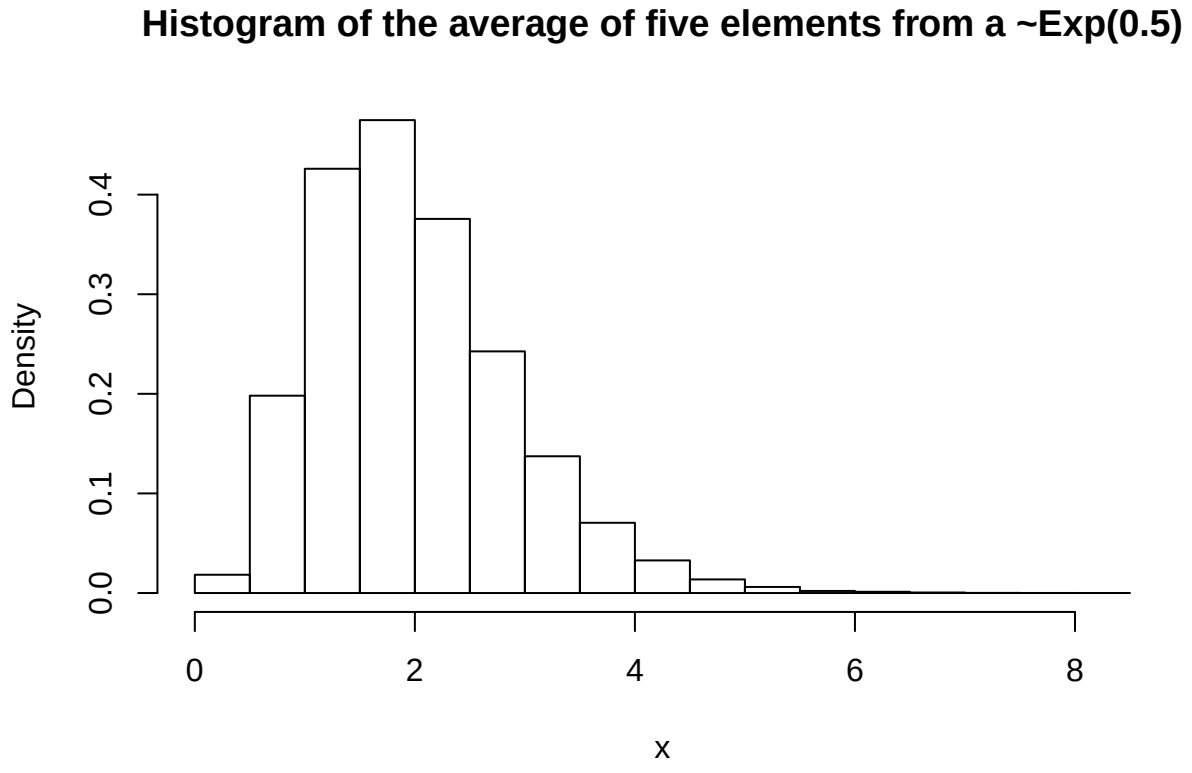

Figure 10: Histogram of 100,000 random means of five elements from a ~Exp(0.5) distribution

1.2.5

```
hist(myRexpMeans$Means10, main="Histogram of the average of ten elements from a ~Exp(0.5)",
     xlab="x", freq = F)
```

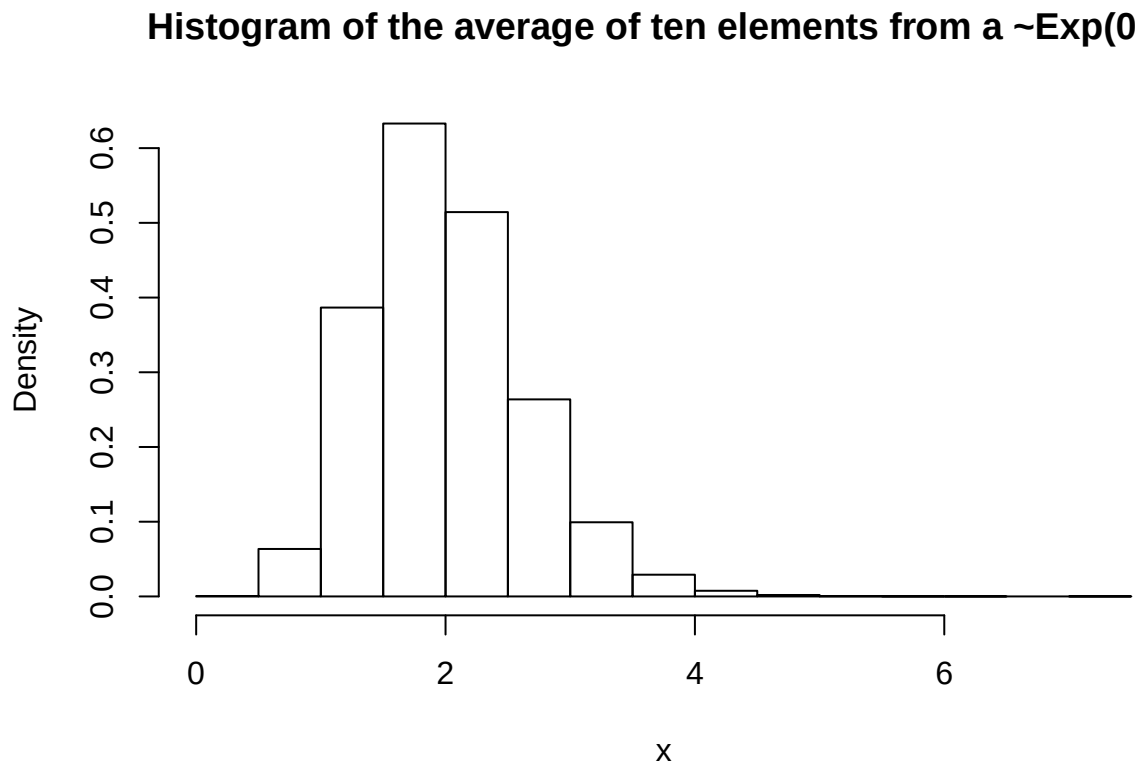

Figure 11: Histogram of 100,000 random means of ten elements from a ~Exp(0.5) distribution

By averaging ten elements, we can see in figure 11 that the distribution continues to "shift" to the right, although the skew is still present.

1.2.6

```
hist(myRexpMeans$Means30, main="Histogram of the average of thirty elements from a ~Exp(0.5)",
     xlab="x", freq = F)
```

In figure 12, we see the histogram when we take the average of 30 elements from the original negative exponential population shown in figure 7. The distribution is now very different, and almost looks like a normal distribution centered around 2. But we can still distinguish a longer tail to the right of the histogram.

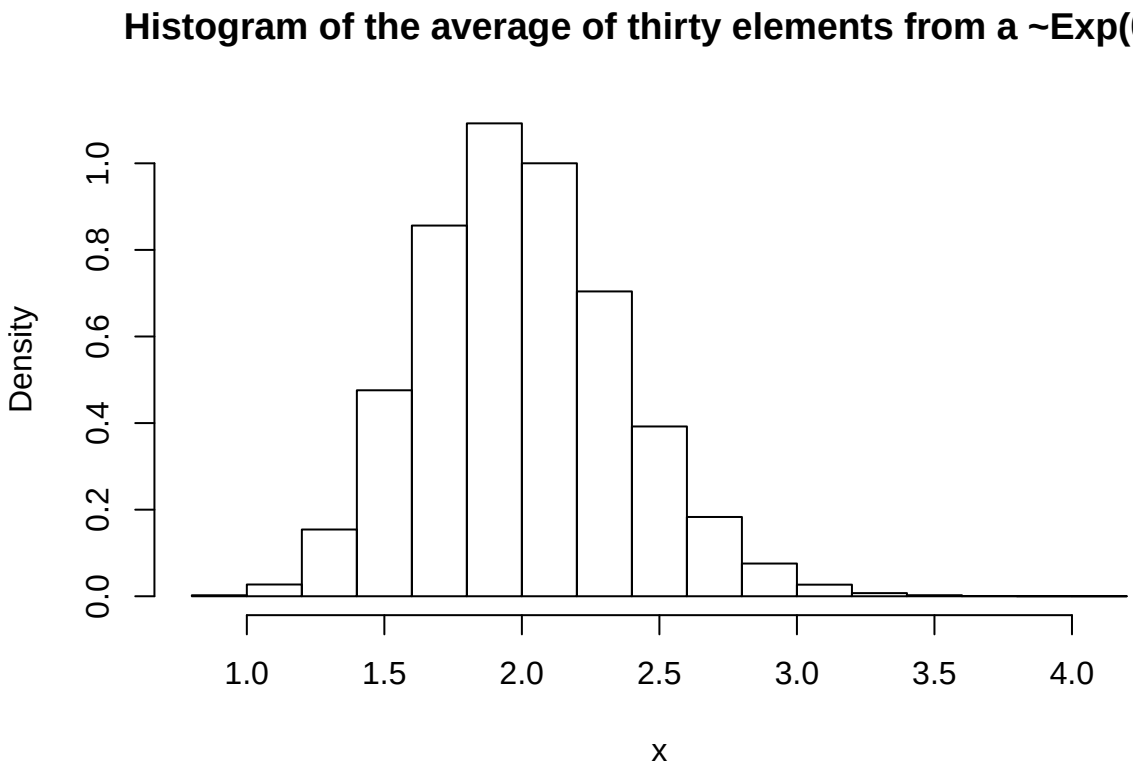

Figure 12: Histogram of 100,000 random means of thirty elements from a ~Exp(0.5) distribution

1.3

1.3.1

```
myBdist <- c(rnorm(5000000, -3, 1), rnorm(5000000, 3, 1))
hist(myBdist, main="Bimodal histogram of ~N(-3, 1) and ~N(3, 1)", xlab="x", freq = F)
```

We can see from the histogram in figure 13 that the distribution is bimodal. In fact, we have random variables from two distinct normal distributions, one centered around -3 and the other around 3. Both variances are equal to 1.

1.3.2

```
myBdistDataFrame <- data.frame(replicate(30, sample(myBdist, 100000)))
myBdistMeans <- data.frame("Means5"= apply(myBdistDataFrame[,1:5], 1, mean),
                           "Means10"= apply(myBdistDataFrame[,1:10], 1, mean),
                           "Means20"= apply(myBdistDataFrame[,1:20], 1, mean),
                           "Means30"= apply(myBdistDataFrame[,1:30], 1, mean))
```

```
hist(myBdistMeans$Means5, main="Histogram of the average of five elements
     from a ~N(-3, 1) and a ~N(3, 1)", xlab="x", freq = F)
```

We can gather from the histogram in image 14 that the bimodality seen in figure 13 is lost. Indeed, this histogram has only one "bump", and has a lot of values between -2 and 2.

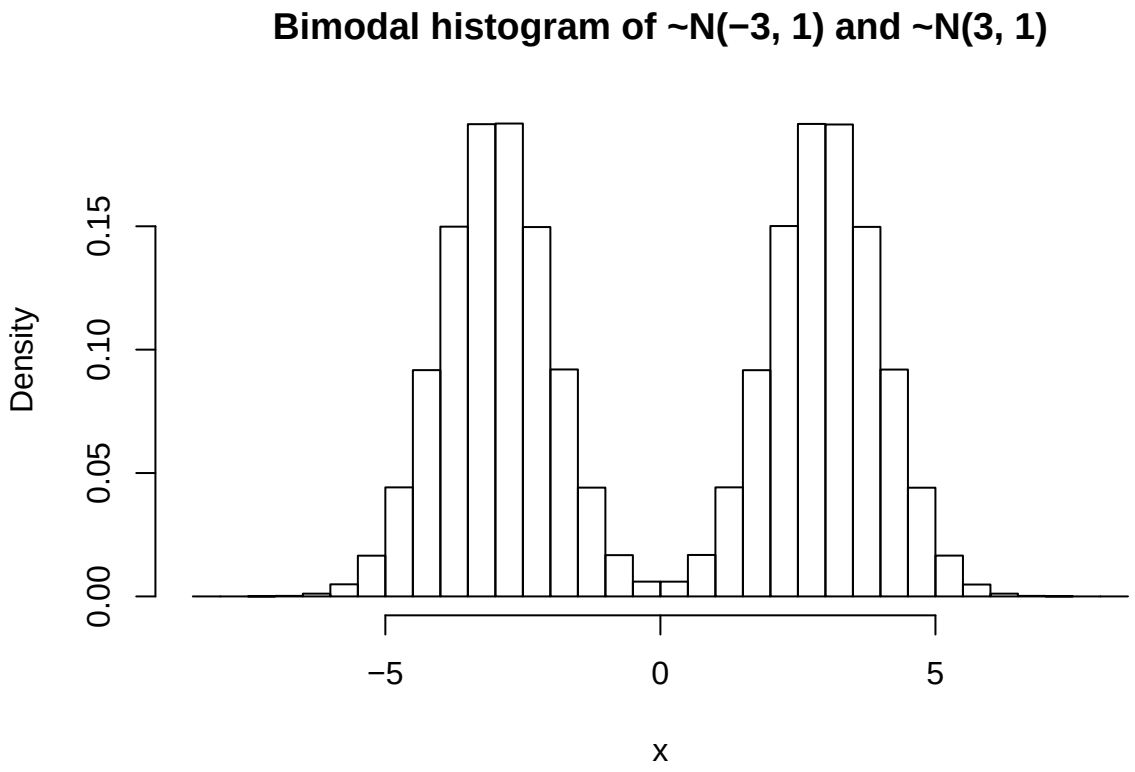

Figure 13: Histogram of 5,000,000 random variables from a ~N(-3, 1) and 5,000,000 random variables from a ~N(3, 1)

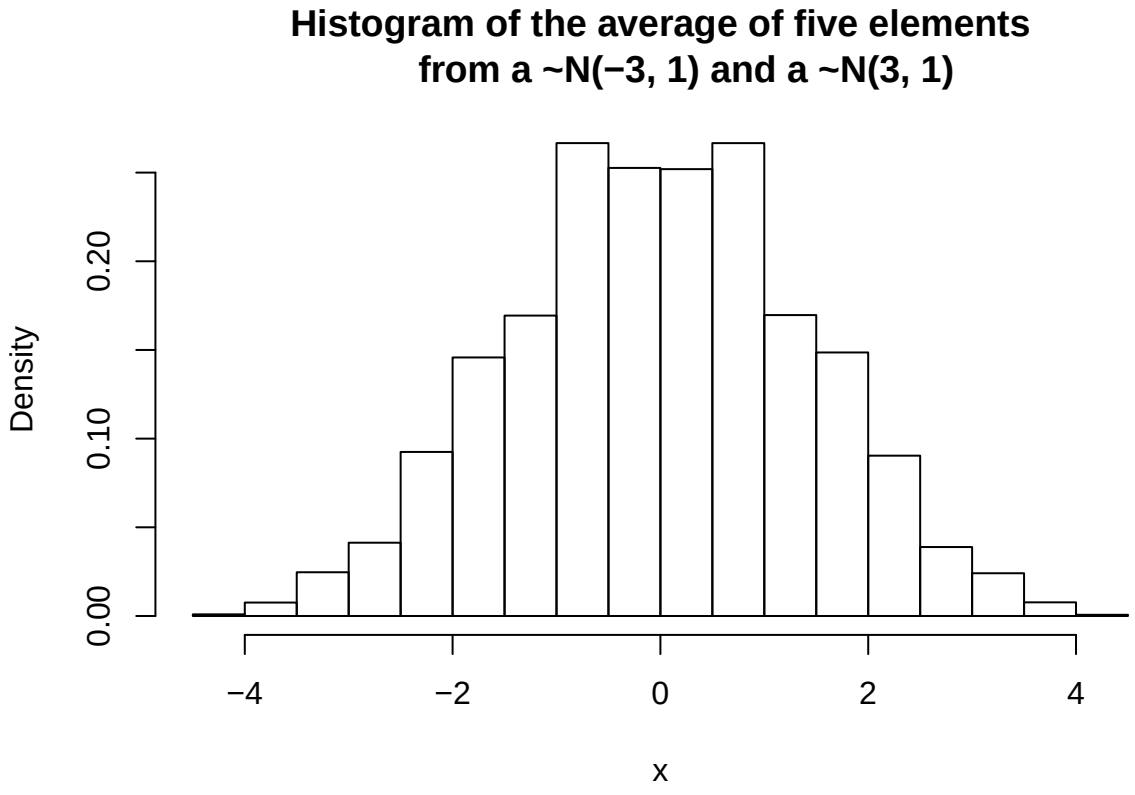

Figure 14: Histogram of 100,000 random means of five elements from a ~N(-3, 1) and a ~N(3, 1) distributions

1.3.3

```
hist(myBdistMeans$Means10, main="Histogram of the average of ten elements
     from a ~N(-3, 1) and a ~N(3, 1)", xlab="x", freq = F)
```

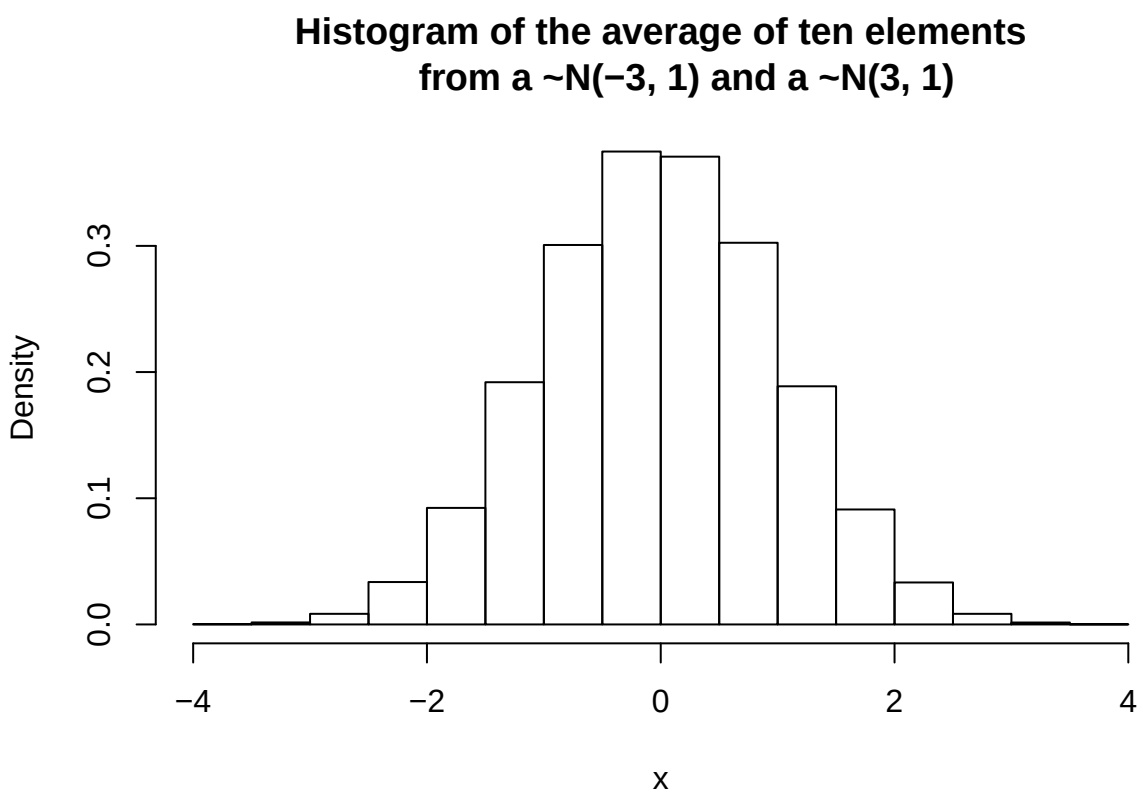

Figure 15: Histogram of 100,000 random means of ten elements from a ~N(-3, 1) and a ~N(3, 1) distributions

The histogram in figure 15 is similar to the previous one (figure 14), but less spread out. It is starting to look like a normal distribution.

```
hist(myBdistMeans$Means20, main="Histogram of the average of twenty elements from
     a ~N(-3, 1) and a ~N(3, 1)", xlab="x", freq = F)
```

By averaging over 20 elements from the original pouplation, the histogram seen in figure 16 is looking a lot like a normal distribution centered around 0. The spread of the distribution is becoming smaller the more elements we average.

```
hist(myBdistMeans$Means30, main="Histogram of the average of thirty elements from
     a ~N(-3, 1) and a ~N(3, 1)", xlab="x", freq = F)
```

The histogram in figure 17 does not look like the original population distribution in figure 13. In fact, the two modes that were present at first have completely disappeared. In its place, we seems to have a single normal distribution, centered around the average of the original two means: $\frac{-3+3}{2} = 0$.

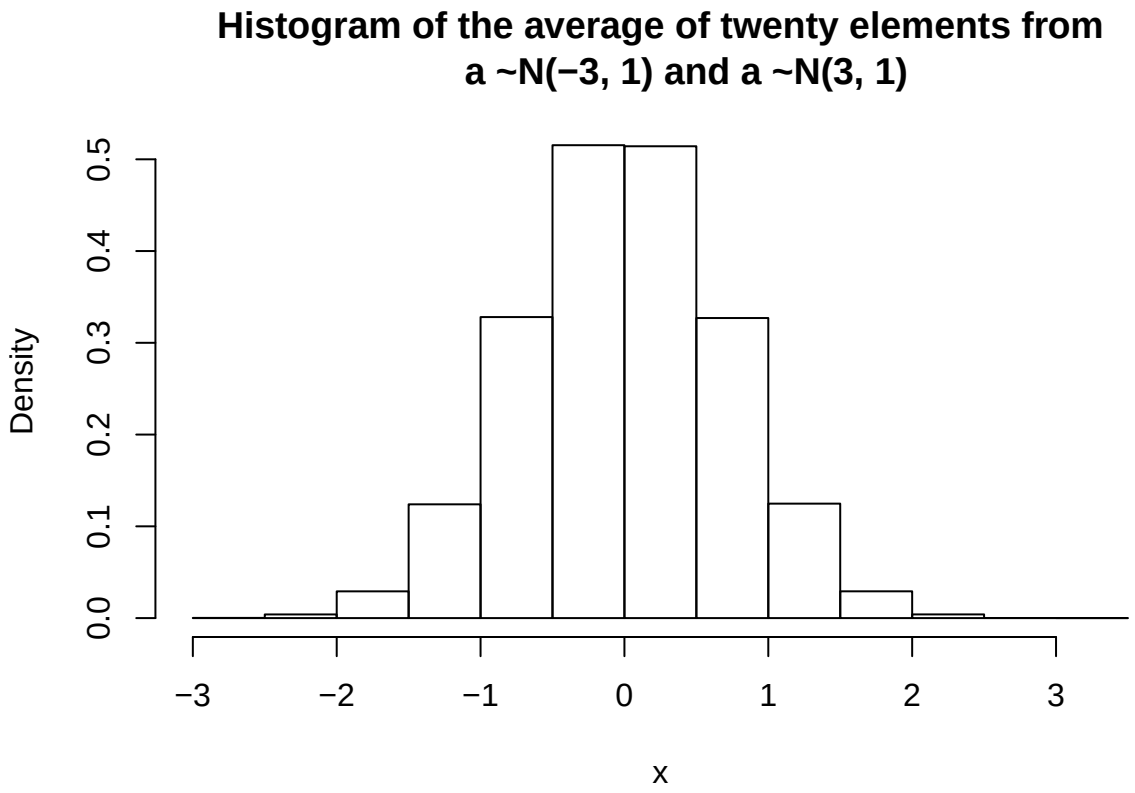

Figure 16: Histogram of 100,000 random means of twenty elements from a ~N(-3, 1) and a ~N(3, 1) distributions

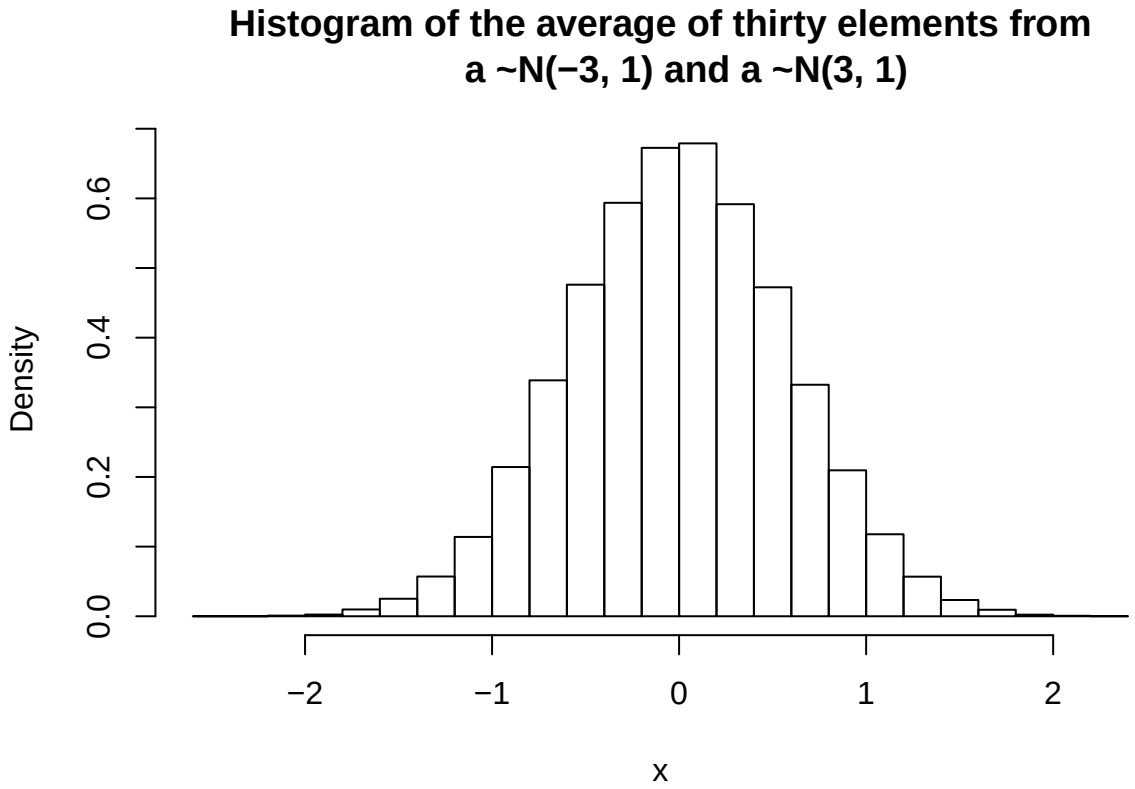

Figure 17: Histogram of 100,000 random means of thirty elements from a ~N(-3, 1) and a ~N(3, 1) distributions

1.3.4

Throughout this problem, we have seen how sampling and averaging several random variables from a random distribution (here we used a uniform, an exponential, and a combination of two normal distributions) leads to new distributions. This phenomenon is called a *convolution of probability distributions*. Convolutions have many uses in various fields, such as signal processing (with regards to both time and space), probability theory, cristallography, and even quantum mechanics!

In this exercise, we have seen that summing $n = 2$ independent random uniformely distributed random variables (with parameters a and b) leads to a triangular distribution (this one actually has a special name and is called *Irwin-Hall distribution*). As n tends to infinity, the distribution of the convolution tends to that of a normally distributed random variable with parameters $\mu = \frac{a+b}{2}$ and $\sigma^2 = \frac{(b-a)^2}{12}$.

When doing the convolution of n exponentially distributed random variables (with parameter θ), the resulting distribution will tend to resemble that of a Gamma distribution (with parameters (n, θ)).

When doing the convolution of n random variables samples from two distinct normal distributions (with parameters $\mu_1; \sigma_1^2$ and $\mu_2; \sigma_2^2$), the resulting distribution tends to be that of a normal distribution with parameters $\mu = \mu_1 + \mu_2$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$.

Question 2

For this question, we are given a data set that we need to clean up. The only instructions we have is that entries created before September 01 2015 do not interest us, and can therefore be discarded.

Loading the data and first visualization

The first step is loading the data onto R. I call `names` to get a first look at the variables we are dealing with, `str` to get a summary of the structure of the data frame, and `head` to see what the data actually looks like. Note: in importing the data, I used the flag `na.strings = c("NA", "")`. This allows for empty strings to be imported as `NA`s.

```
myData <- read.csv("hw2.csv", header = T, stringsAsFactors = F, na.strings = c("NA", ""))
names(myData)
```

```
##  [1] "Project.Name"
##  [2] "Created.Date"
##  [3] "Project.Status"
##  [4] "Opportunity..Purchased.Thru"
##  [5] "Agreement.Type"
##  [6] "Install.Branch"
##  [7] "Opportunity..Utility.Company"
##  [8] "Opportunity..Jurisdiction..Jurisdiction.Name"
##  [9] "Proposal..System.Size.STC.DC"
## [10] "Service.Contract..Service.Contract.Event..Using.Build.Partner."
## [11] "Opportunity..Service.Panel.Upgrade"
## [12] "Opportunity..Reroof.under.Array"
## [13] "Opportunity..HOA."
## [14] "Opportunity..PE.Stamp.Required."
```

```
str(myData, strict.width = "w")
```

```
## 'data.frame':    49880 obs. of  14 variables:
## $ Project.Name : chr "PR-1631525512" "PR-1727346069" "PR-1321701825"
##    "PR-1416881773" ...
## $ Created.Date : chr "11/19/15" "5/31/15" "1/25/16" "11/19/15" ...
## $ Project.Status : chr "Open" "Cancelled" "Open" "Open" ...
## $ Opportunity..Purchased.Thru : chr "Costco" NA NA "Costco" ...
## $ Agreement.Type : chr "Customer Owned - Full Upfront" "Custom PPA Fixed"
##    "Custom PPA Fixed" "Customer Owned - Bank Financed" ...
## $ Install.Branch : chr "Sacramento" "Las Vegas" "MD - Columbia" "Inland
##    Empire" ...
## $ Opportunity..Utility.Company : chr "PG&E" "NV Energy South" "BG&E" "SCE"
##    ...
## $ Opportunity..Jurisdiction..Jurisdiction.Name : chr "CA-CITY CHICO"
##    "NV-COUNTY CLARK" "MD-COUNTY BALTIMORE" "CA-CITY NORCO" ...
## $ Proposal..System.Size.STC.DC : num 5.57 5.72 8.48 5.83 7.54 ...
## $ Service.Contract..Service.Contract.Event..Using.Build.Partner.: int 0 0
##    1 0 0 0 0 0 0 1 ...
## $ Opportunity..Service.Panel.Upgrade : int 0 0 0 0 0 0 0 0 0 0 ...
## $ Opportunity..Reroof.under.Array : int 0 0 0 0 0 0 0 0 0 0 ...
## $ Opportunity..HOA.  : chr "No" "Yes" "Yes" "No" ...
## $ Opportunity..PE.Stamp.Required.  : int 0 0 1 1 0 1 0 0 0 1 ...
```

```
head(myData, 5)
```

```
##    Project.Name Created.Date Project.Status Opportunity..Purchased.Thru
## 1 PR-1631525512     11/19/15           Open                       Costco
## 2 PR-1727346069      5/31/15      Cancelled                         <NA>
## 3 PR-1321701825      1/25/16           Open                         <NA>
## 4 PR-1416881773     11/19/15           Open                       Costco
## 5 PR-1767873453      5/31/15       Complete                         <NA>
##                    Agreement.Type Install.Branch
## 1  Customer Owned - Full Upfront     Sacramento
## 2               Custom PPA Fixed      Las Vegas
## 3               Custom PPA Fixed  MD - Columbia
## 4 Customer Owned - Bank Financed  Inland Empire
## 5               Custom PPA Fixed      Las Vegas
##   Opportunity..Utility.Company
## 1                        PG&E
## 2             NV Energy South
## 3                        BG&E
## 4                         SCE
## 5             NV Energy South
##   Opportunity..Jurisdiction..Jurisdiction.Name
## 1                              CA-CITY CHICO
## 2                             NV-COUNTY CLARK
## 3                          MD-COUNTY BALTIMORE
## 4                              CA-CITY NORCO
## 5                             NV-COUNTY CLARK
##   Proposal..System.Size.STC.DC
## 1                        5.565
## 2                        5.720
## 3                        8.480
## 4                        5.830
## 5                        7.540
##   Service.Contract..Service.Contract.Event..Using.Build.Partner.
## 1                                                              0
## 2                                                              0
## 3                                                              1
## 4                                                              0
## 5                                                              0
##   Opportunity..Service.Panel.Upgrade Opportunity..Reroof.under.Array
## 1                                  0                               0
## 2                                  0                               0
## 3                                  0                               0
## 4                                  0                               0
## 5                                  0                               0
##   Opportunity..HOA. Opportunity..PE.Stamp.Required.
## 1                No                               0
## 2               Yes                               0
## 3               Yes                               1
## 4                No                               1
## 5              <NA>                               0
```

We can see that we have 49880 rows (entries) and 14 columns (variables). We can see two things from this information: first, there is a column `myData$Created.Date`. We will use this one to filter out observations

before our specified date. Second, the columns need to be renamed for simplicity sake. These are my first priorities. We can also see that dates are imported as strings in the format `mm/dd/yy`. This is important, since we will be converting these strings into a `Date` format, so that we can sort them and keep only the ones that matter to us here.

Removing entries that we don't need (prior to September 01 2015)

To convert the dates into a `Date` format, I use the code below.

```
myData$Created.Date <- as.Date( as.character(myData$Created.Date), "%m/%d/%y")
```

Now, we can perform mathematic operations on this row. For instance, to see only entries created on or after September 01 2015, we can use `myData$Created.Date[myData$Created.Date >= "2015-09-01"]`. Now, to subset our data frame to only this period, we use the following code.

```
myData <- subset(myData, myData$Created.Date >= "2015-09-01")
```

We now have 32320 rows in our data frame.

Renaming the variables

Now, let's rename the variables.

```
colnames(myData) <- c("PName",
                      "CDate",
                      "PStatus",
                      "PurchasedThru",
                      "AgrType",
                      "InstBranch",
                      "UtilityCompany",
                      "Jurisdiction",
                      "SysSize",
                      "UsingPartner",
                      "PanelUpgrade",
                      "ReroofArray",
                      "HOA",
                      "PEStampReq")
str(myData, strict.width = "w")
```

```
## 'data.frame':    32320 obs. of  14 variables:
## $ PName : chr "PR-1631525512" "PR-1321701825" "PR-1416881773"
##    "PR-1991119351" ...
## $ CDate : Date, format: "2015-11-19" "2016-01-25" ...
## $ PStatus : chr "Open" "Open" "Open" "Open" ...
## $ PurchasedThru : chr "Costco" NA "Costco" "Standard (Non-Retail)" ...
## $ AgrType : chr "Customer Owned - Full Upfront" "Custom PPA Fixed"
##    "Customer Owned - Bank Financed" "Custom Lease" ...
## $ InstBranch : chr "Sacramento" "MD - Columbia" "Inland Empire" "NYC
##    North" ...
## $ UtilityCompany: chr "PG&E" "BG&E" "SCE" "ConEdison" ...
## $ Jurisdiction : chr "CA-CITY CHICO" "MD-COUNTY BALTIMORE" "CA-CITY NORCO"
```

```
##    "NYC-NEW YORK CITY" ...
## $ SysSize : num 5.57 8.48 5.83 4.24 7.7 ...
## $ UsingPartner : int 0 1 0 0 0 0 1 0 0 0 ...
## $ PanelUpgrade : int 0 0 0 0 0 0 0 0 0 0 ...
## $ ReroofArray : int 0 0 0 0 0 0 0 0 0 0 ...
## $ HOA : chr "No" "Yes" "No" "No" ...
## $ PEStampReq : int 0 1 1 1 0 0 1 0 0 1 ...
```

Now the variables have names that are not only shorter, but easier to understand as well. The next step is coercing the data into "better" types (`factor`, `logical`, etc).

Modifying the data types

- `PName`, the project name/code has type `character` and class `character`. No changes are needed.
- `CDate`, the project creation date, has type `double` and class `Date`. No changes are needed.
- `PStatus`, the project status, has type `character` and class `character`. This variable can be transformed into a `factor` with `myData$PStatus <- factor(myData$PStatus, levels = unique(myData$PStatus))`.

```
myData$PStatus <- factor(myData$PStatus, levels = unique(myData$PStatus))
summary(myData$PStatus)
```

```
##      Open  Complete   Blocked Cancelled      NA's
##     18506      3297      3865      6637        15
```

We can also plot a histogram of this data:

```
plot(myData$PStatus, main="Histogram of project status", ylab="Frequency")
```

We can see in figure 18 that there are considerably more open projects than completed, blocked, and cancelled.

- `PurchasedThru`, the store where the equipment was bought, has type `character` and class `character`. This variable can be transformed into a `factor`. But first, we need to change one name, and group two entries together ("Standard (Non-Retail)" simplified to "Non-Retail", and "Costco (Dept 44)" is the same as "Costco").

```
myData$PurchasedThru[myData$PurchasedThru == "Costco (Dept 44)"] <- "Costco"
myData$PurchasedThru[myData$PurchasedThru == "Standard (Non-Retail)"] <- "Non-Retail"
myData$PurchasedThru <- factor(myData$PurchasedThru, levels = unique(myData$PurchasedThru))
summary(myData$PurchasedThru)
```

```
##     Costco Non-Retail Home Depot    Comcast   Best Buy       NA's
##       6173      16180        677        310        135       8845
```

- `AgrType`, the agreement type, has type `character` and class `character`. This variable can be converted into a `factor` with `myData$AgrType <- factor(myData$AgrType, levels = unique(myData$AgrType))`.

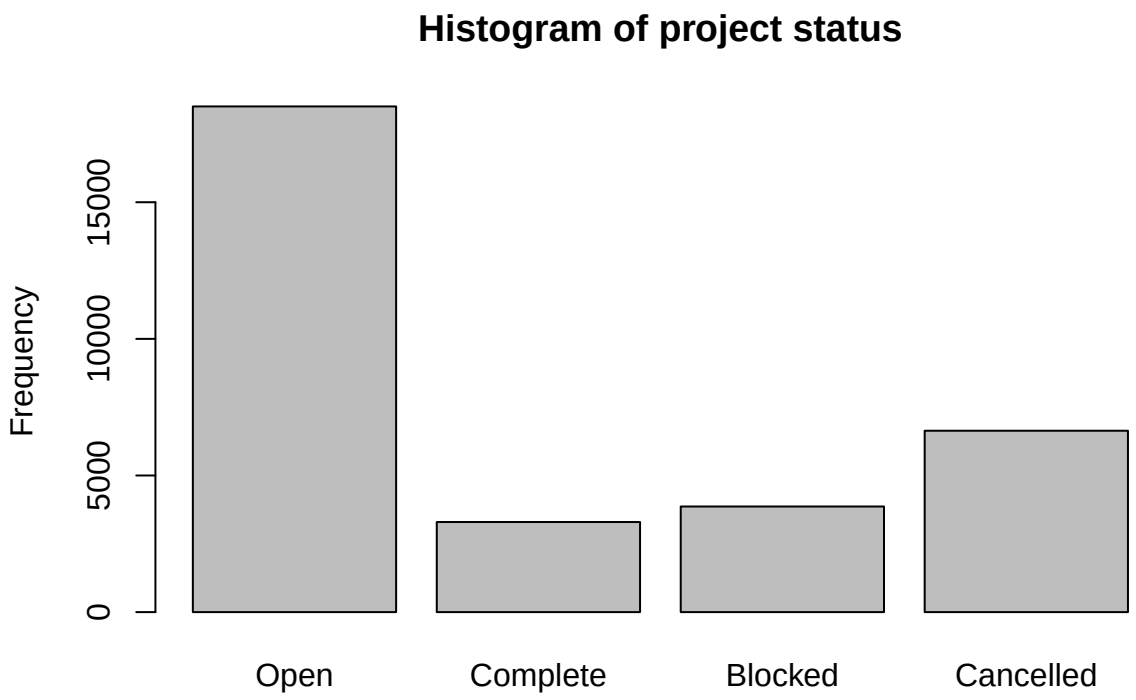

Figure 18: Histogram of project status

```
myData$AgrType <- factor(myData$AgrType, levels = unique(myData$AgrType))
summary(myData$AgrType)
```

```
##  Customer Owned - Full Upfront              Custom PPA Fixed
##                           3882                         11455
## Customer Owned - Bank Financed                 Custom Lease
##                           2864                          7739
##                     Prepaid PPA                Prepaid Lease
##                            615                           107
##          Low Upfront PPA Fixed                      Prepaid
##                              4                             2
##                         Custom     Sun Run Service + Pre-pay
##                              2                             1
##                           NA's
##                           5649
```

- `InstBranch`, the install branch, has type `character` and class `character`. This vector contains names of cities. There are 38 distinct cities. It would be nice if all the entries followed a certain pattern, such as `STATE - City`, but they don't. We can use regular expressions in order to filter them out and change them manually, but won't be doing so in this assignment.
- `UtilityCompany`, the utility company, has type `character` and class `character`. This variable can be converted into a `factor` with `myData$UtilityCompany <- factor(myData$UtilityCompany, levels = unique(myData$UtilityCompany))`.

```
myData$UtilityCompany <- factor(myData$UtilityCompany, levels = unique(myData$UtilityCompany))
```

Note: There are 55 levels for this one, so we are not printing the result of `summary(myData$UtilityCompany)`.

- `Jurisdiction`, the jurisdcition, has type `character` and class `character`. This variable contains names of cities. They all follow a certain pattern: `STATE - CITY`, except for one where the city is in lowercase. This specificity was found by looking at `unique(myData$Jurisdiction)` (there are 1220 unique entries!), and is `NY-City of Longbeach`. To find the rows, we can use the query `which(myData$Jurisdiction == "NY-City of Longbeach")`, which returns 25671, 28118, 30096. In order to update them, we can then do:

```
indices_Jurisdiction_toUpper <- which(myData$Jurisdiction == "NY-City of Longbeach")
myData$Jurisdiction[indices_Jurisdiction_toUpper] <- toupper("NY-City of Longbeach")
```

We can do some interesting stuff with this. For example, we can have a look at how many entries have a jurisdiction in the state of California:

```
table(grepl("CA.*", myData$Jurisdiction))
```

```
##
## FALSE  TRUE
## 21410 10910
```

We see that there are 10910 entries that correspond to this query. About a third of our dataset (approximately 30,000 rows) has the jurisdiction in California.

- `SysSize`, the system size/area, has type `double` and class `numeric`. No changes are needed. We can, however, plot a histogram of these values:

```
hist(myData$SysSize, main = "Histogram of system sizes",
     xlab = "System size", freq = F, ylim = c(0, 0.18))
x <- seq(0, 30, 0.1)
y <- dgamma(x, shape = mean(myData$SysSize, na.rm = T))
lines(y~x, col = "blue", legend(12, 0.1,
     paste("Gamma distribution (alpha=", round(mean(myData$SysSize, na.rm = T), 3), ")"),
     col = "blue", lty = 1, cex = 0.9))
```

Figure 19 shows a very nice distribution (it looks like a Poisson distribution).

- `UsingPartner`, whether the owner is using a partner, has type `integer` and class `integer`. It can be converted into a `logical`: with `myData$UsingPartner <- as.logical(myData$UsingPartner)`.

```
myData$UsingPartner <- as.logical(myData$UsingPartner)
summary(myData$UsingPartner)
```

```
##    Mode   FALSE    TRUE    NA's
## logical   30789    1531       0
```

- `PanelUpgrade`, whether the owner needs/has had to upgrade, has type `integer` and class `integer`. It can be converted into a `logical` with `myData$PanelUpgrade <- as.logical(myData$PanelUpgrade)`.

```
myData$PanelUpgrade <- as.logical(myData$PanelUpgrade)
summary(myData$PanelUpgrade)
```

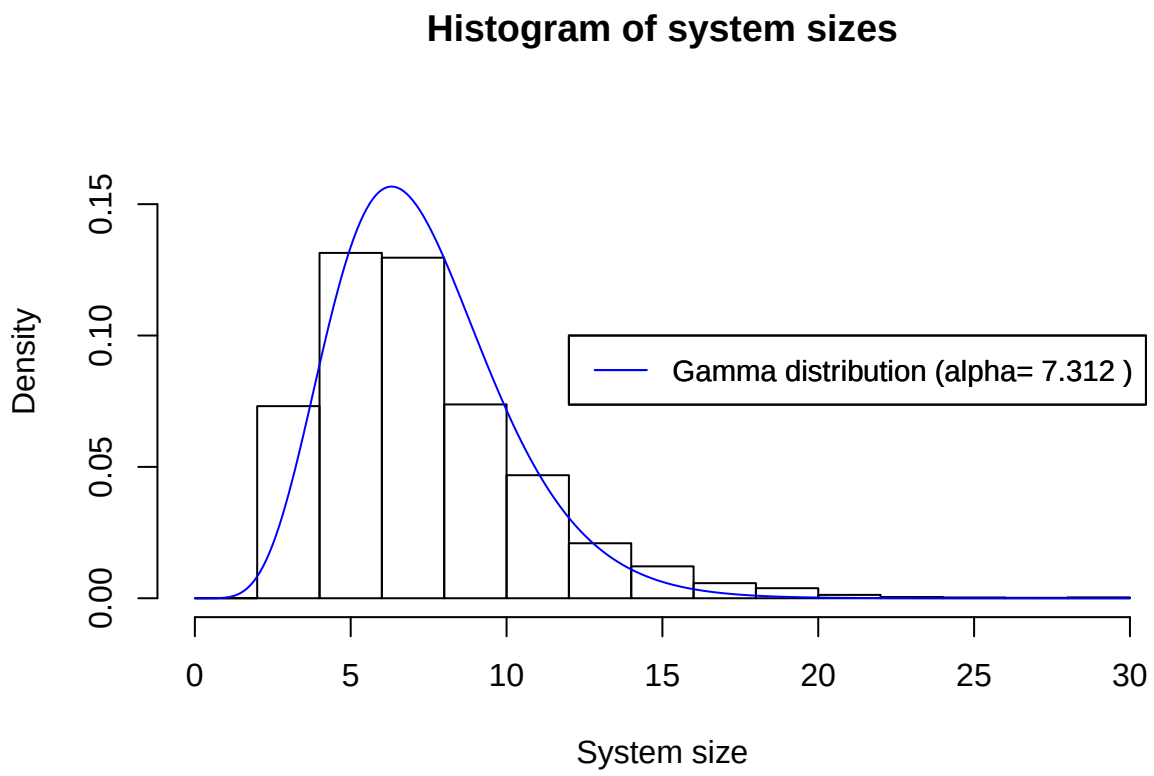

Figure 19: Histogram of system sizes

```
##    Mode   FALSE    TRUE    NA's
## logical   29043    3277       0
```

- `ReroofArray`, whether the owner needs/has had a reroof array, has type `integer` and class `integer`. It can be converted into a `logical` with `myData$ReroofArray <- as.logical(myData$ReroofArray)`.

```
myData$ReroofArray <- as.logical(myData$ReroofArray)
summary(myData$ReroofArray)
```

```
##    Mode   FALSE    TRUE    NA's
## logical   29328    2992       0
```

- `HOA`, whether the owner is part of the home-owners association, has type `character` and class `character`. We have two options here: either convert this variable into a `logical` or a `factor`. I decided to convert it into a logical. First, we need to convert "No"s to "0"s and "Yes"s to "1"s. Another choice that I made is to convert entries with "Customer did not know" to `NA`s. This all is done below:

```
myData$HOA[myData$HOA == "No"] <- "0"
myData$HOA[myData$HOA == "Yes"] <- "1"
myData$HOA[myData$HOA == "Customer did not know"] <- NA
myData$HOA <- as.logical(myData$HOA == "1")
summary(myData$HOA)
```

```
##    Mode   FALSE    TRUE    NA's
## logical   23954    6523    1843
```

- `PEStampReq`, whether a PE stamp is required, has type `integer` and class `integer`. It can be converted into a `logical`with `myData$PEStampReq <- as.logical(myData$PEStampReq)`.

```
myData$PEStampReq <- as.logical(myData$PEStampReq)
summary(myData$PEStampReq)
```

```
##    Mode   FALSE    TRUE    NA's
## logical   22466    9854       0
```

To visualize the changes we made, we can have another look at the structure of our data frame:

```
str(myData, strict.width = "w")
```

```
## 'data.frame':    32320 obs. of  14 variables:
## $ PName : chr "PR-1631525512" "PR-1321701825" "PR-1416881773"
##    "PR-1991119351" ...
## $ CDate : Date, format: "2015-11-19" "2016-01-25" ...
## $ PStatus : Factor w/ 4 levels "Open","Complete",..: 1 1 1 1 2 3 1 3 4 1
##    ...
## $ PurchasedThru : Factor w/ 5 levels "Costco","Non-Retail",..: 1 NA 1 2 2
##    NA NA 1 2 2 ...
## $ AgrType : Factor w/ 10 levels "Customer Owned - Full Upfront",..: 1 2 3
##    4 2 NA 2 3 NA 4 ...
## $ InstBranch : chr "Sacramento" "MD - Columbia" "Inland Empire" "NYC
##    North" ...
## $ UtilityCompany: Factor w/ 55 levels "PG&E","BG&E",..: 1 2 3 4 5 6 7 1 8
##    9 ...
## $ Jurisdiction : chr "CA-CITY CHICO" "MD-COUNTY BALTIMORE" "CA-CITY NORCO"
##    "NYC-NEW YORK CITY" ...
## $ SysSize : num 5.57 8.48 5.83 4.24 7.7 ...
## $ UsingPartner : logi FALSE TRUE FALSE FALSE FALSE FALSE ...
## $ PanelUpgrade : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## $ ReroofArray : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
## $ HOA : logi FALSE TRUE FALSE FALSE FALSE FALSE ...
## $ PEStampReq : logi FALSE TRUE TRUE TRUE FALSE FALSE ...
```

Now our data frame is much better organized, and we can start delving deeper into its contents.

Cleaning up NAs

`PStatus` contains the status of the project. It can be of only 4 types: Open, Complete, Blocked, Cancelled. However, there are 15 `NA`s in this vector. If we print out these rows of the data frame, we get the following:

```
myData[is.na(myData$PStatus),]
```

```
##                                PName      CDate PStatus PurchasedThru
## 879                             test 2015-10-30    <NA>          <NA>
## 11252                           Dmas 2015-12-23    <NA>          <NA>
## 12076           202R-043KONA/ Konadu 2015-12-17    <NA>          <NA>
## 15855                     2143773610 2015-10-27    <NA>          <NA>
## 19798                Rubina Needles 2015-11-10    <NA>          <NA>
```

```
## 23024                          austin 2016-01-06    <NA>          <NA>
## 40205                     Laura ford 2015-09-02    <NA>          <NA>
## 41383 gonzalez maria 115r441gonz 0915 2015-09-02    <NA>          <NA>
## 42024              PK1VALVR9K3C:002-H 2015-09-03    <NA>          <NA>
## 42533 gonzalez maria 115r441gonz 0915 2015-09-02    <NA>          <NA>
## 42534 gonzalez maria 115r441gonz 0915 2015-09-02    <NA>          <NA>
## 42537 gonzalez maria 115r441gonz 0915 2015-09-02    <NA>          <NA>
## 47827                    231R-024HARW 2015-10-12    <NA>          <NA>
## 48997                    pr1732531655 2015-10-27    <NA>          <NA>
## 49080                   PR-2132921988 2015-10-26    <NA>          <NA>
##       AgrType InstBranch UtilityCompany Jurisdiction SysSize UsingPartner
## 879      <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 11252    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 12076    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 15855    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 19798    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 23024    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 40205    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 41383    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 42024    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 42533    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 42534    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 42537    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 47827    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 48997    <NA>       <NA>           <NA>         <NA>      NA        FALSE
## 49080    <NA>       <NA>           <NA>         <NA>      NA        FALSE
##       PanelUpgrade ReroofArray HOA PEStampReq
## 879          FALSE       FALSE  NA      FALSE
## 11252        FALSE       FALSE  NA      FALSE
## 12076        FALSE       FALSE  NA      FALSE
## 15855        FALSE       FALSE  NA      FALSE
## 19798        FALSE       FALSE  NA      FALSE
## 23024        FALSE       FALSE  NA      FALSE
## 40205        FALSE       FALSE  NA      FALSE
## 41383        FALSE       FALSE  NA      FALSE
## 42024        FALSE       FALSE  NA      FALSE
## 42533        FALSE       FALSE  NA      FALSE
## 42534        FALSE       FALSE  NA      FALSE
## 42537        FALSE       FALSE  NA      FALSE
## 47827        FALSE       FALSE  NA      FALSE
## 48997        FALSE       FALSE  NA      FALSE
## 49080        FALSE       FALSE  NA      FALSE
```

We can *clearly* see that these entries are all gibberish (one of them is literally a `test` entry!), and have `NA`s in all columns except for the creation date and project name. Therefore, we can remove them from the data frame.

```
myData <- myData[!is.na(myData$PStatus),]
```

We now have 32305 rows in the data frame.

However, there are still lots of entries with many `NA`s that we can deal with. The function `rowSums` is very useful in this case. Indeed, with a single statement, we can find the rows with more than a specific number of `NA`s. In this case, we can conservatively say that if an entry has more than five `NA`s, it can probably be

removed from the data set. Looking at those entries (with command `myData[rowSums(is.na(myData)) >= 5,]`) shows that there are many entries with basically only `NA`s, except for a couple of them, where the project status was "Cancelled". Therefore, we can adapt the above code to include this condition. Then we update `myData` to keep everything but the rows that satisfy both conditions.

```
myData <- myData[!(rowSums(is.na(myData)) >= 5 & myData$PStatus != "Cancelled"), ]
```

We now have 32181 rows in the data frame.

`PName` contains the project name. The general form of the values is `PR-<project number>`. We can use regular expressions to see if any entries do not follow this concensus for the naming of the project:

```
myData$PName[!grepl("^PR-.*", myData$PName)]
```

```
## [1] "5403 Mount Vernon Ave-91710-3540" "Sweeney,Merrilee-"
```

We see here that there are two entries for which the project name does not follow the pattern `PR-<project number>`. In this case, we will remove these two entries.

```
indices_PName_toRemove <- c(which(myData$PName == myData$PName[!grepl("^PR-.*",
                                                                      myData$PName)][1]),
                            which(myData$PName == myData$PName[!grepl("^PR-.*",
                                                                      myData$PName)][2]))
myData <- myData[-indices_PName_toRemove,]
```

We now have 32179 rows in the data frame.

Organizing the data by monthly activity

We can create a new column `CDateMonth` in order to organize the data monthly. First, we use a series of nested `ifelse` commands to sort the data according to month. Then, we coerce that data into a `factor`. Note: The order here is important, and goes *from September to April*. Indeed, since we are inbetween two years, in this case January **does not** come before September. We can then plot the new column and see a histogram of the number of projects created per month.

```
myData$CDateMonths <-
  ifelse(myData$CDate >= "2015-09-01" & myData$CDate <= "2015-09-30", "September",
  ifelse(myData$CDate >= "2015-10-01" & myData$CDate <= "2015-10-31", "October",
  ifelse(myData$CDate >= "2015-11-01" & myData$CDate <= "2015-11-30", "November",
  ifelse(myData$CDate >= "2015-12-01" & myData$CDate <= "2015-12-31", "December",
  ifelse(myData$CDate >= "2016-01-01" & myData$CDate <= "2016-01-31", "January",
  ifelse(myData$CDate >= "2016-02-01" & myData$CDate <= "2016-02-29", "February",
  ifelse(myData$CDate >= "2016-03-01" & myData$CDate <= "2016-03-31", "March",
  ifelse(myData$CDate >= "2016-04-01" & myData$CDate <= "2016-04-30", "April", NA)))))))
myData$CDateMonths <- factor(myData$CDateMonths,
                             levels = c("September", "October", "November", "December",
                                        "January", "February", "March", "April"),
                             ordered = T)
plot(myData$CDateMonths, main = "Histogram of monthly activity",
                         ylab = "Number of projects created",
                         xlab = "Month")
```

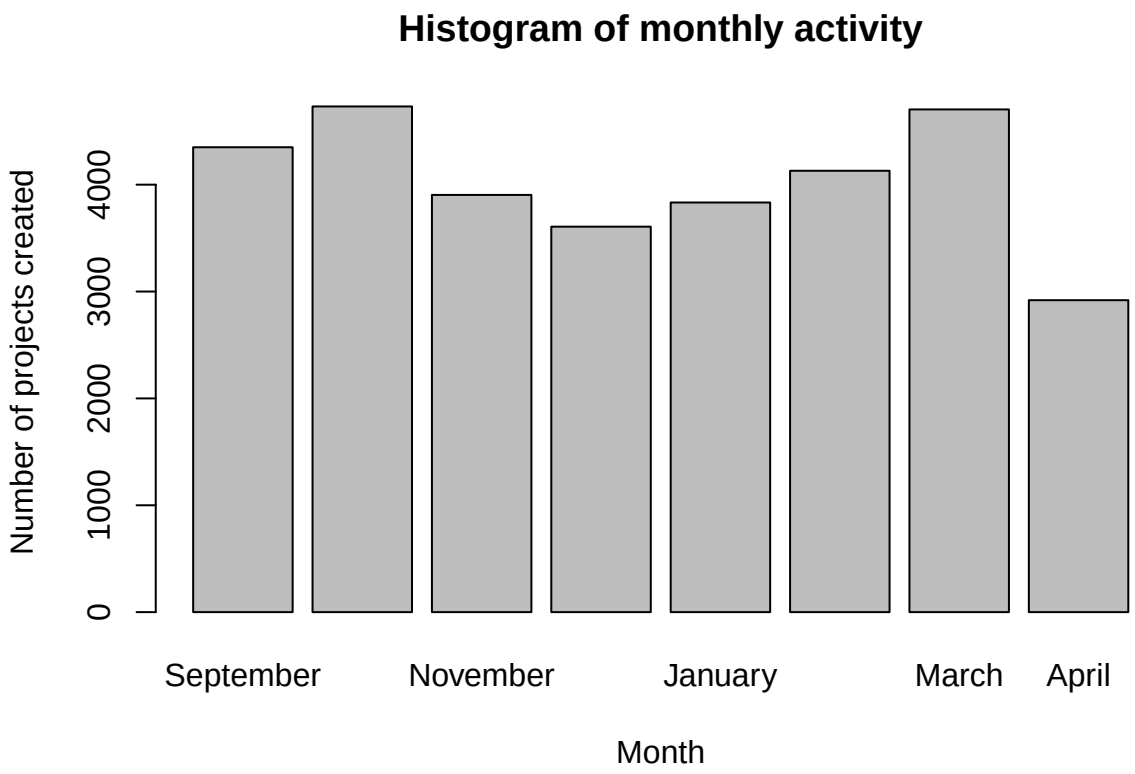

Figure 20: Histogram of the total number of projects created per month

As we can see in figure 20, the number of projects created per month has been over 3000 since September 2015. There was less demand during the winter months. Since December, there has been a steady growth in projects created. Note: the last data entry is from 2016-04-22, hence the "shorter" column for the month of April.

Conclusion

After all this janitorial exercise, we have managed to clean up the data somewhat well. Columns now have cleaner (shorter) and more intuitive names, the data is uniform across the columns (each variable has its very specific type), we have defined factors and logical vectors, and even corrected a jurisdiction name. We removed rows that corresponded to old data, those that had "too many" empty cells ("too many" is merely my interpretation, and we could have chosen a higher or smaller number of `NA`s for an entry to be considered discardable), and those whose project name didn't follow the usual pattern that we would expect. We have shown tables and plots summarizing the data (and even a potential statistical distribution for system sizes!). Finally, we have created a whole new variable that collects the data in monthly blocks, which is useful in order to visualize business in a per-month basis.

The conclusion that I take from this problem is that a lot of work can be done with a data set, even without knowing what the data refers to. There are many different ways to approach a dataset-cleaning problem, and my first take at it is probably still very amateurish, but I think I learned a lot already by doing this exercise. One thing I think I could have explored more is how to use regular expressions to further separate the data by state and/or city. For example, I would have liked to create a histogram with the number of projects per state. I managed to create tables for each state (using the `Jurisdiction` column and the state of California as an example in this write-up), but not an entire plot that would get each state automatically. The built-in dataset `state.abb`, which is a vector with all US states two-letter abbreviations, could be a good idea of where to start in order to accomplish this goal, but I unfortunately did not manage to do it for this assignment.