Lab 1

STAT 131A

January 24, 2022

Welcome to the first lab of STAT 131A!

The goal of the computational part of the labs is to get you familiar with the data analysis you have already seen in lectures and produce them in R by yourself. Most of the exercises in the lab will be a simplified version of what is done in the lectures.

You may want to go ahead and compile this lab (the "knit" buttom). That will give you a pdf version to look at if you want (and verifies you aren't having any problem compiling the file before you make changes).

But generally you should go through the .Rmd and go through the code and make sure you understand it, then do the exercise. Interactively, you will probably just hit run for each chunk. But once you have successfully completed the exercise (i.e. ready to turn it in), reknit the document (in pdf) to make sure you did everything right. That is much better than doing the whole thing, and only at the end (when you want to turn it in!) discovering you have to go back because there was an error somewhere from earlier.

Submitting the lab

Instructions for code: We have already put in R chunks for where your code should go. For some questions, we will ask you to save your answer as a particular variable. For example, we might give you a code chunk that looks like this:

```
set.seed(4291)
# insert code here save the median of your simulated data as
# 'medx'
```

And you might complete it like so:

```
set.seed(4291)
# insert code here save the median of your simulated data as
# 'medx'
x <- rnorm(1000)
medx <- median(x)
medx
```

```
## [1] 0.01612433
```

Misspelling of the requested variable name may result in errors when answers are pulled for subsequent parts or for scoring purposes. In general objects that are not large datasets which take up a page or more should be called at the end of the code block in which they are asked for, for scoring purposes. Thus, it is a good idea to put the variable name at the bottom so it prints (assuming its not a huge object), and usually this should be already part of the provided code. It also helps you check your work.

Of note: Sometimes an exercise will ask for code AND posit a question. Make sure that if the answer to the question is not an output of the code, then you must answer it separately outside of the code block. For example the problem might ask you to make a plot and describe its prominent features. You would write the code to make the plot, but also write a sentence or two outside of the code block (plain text) to describe the features of the plot.

Submission: We are asking you to submit both the *PDF* and *.Rmd* to gradescope. **Do not modify your file name**. Otherwise, gradescope would fail to recognize it. For example, you will submit `LAB01.pdf` and `LAB01.Rmd` for this lab.

Exploratory Analysis – Rent Price in SF Bay Area

According to a U.S. News and World Report in 2016 in collaboration with Zillow, the off-campus housing price of Stanford and Berkeley have ranked No.1 and No.3 among top-ranked universities. Student renters in the Bay area expect to see the highest annual rental appreciation, which is anticipated to grow by more than 6 percent over the coming year. (Reference: College Students Can Expect Highest Off-Campus Housing Costs in Palo Alto and Princeton, Lowest in St. Louis) The rent price of Berkeley and its surrounding cities may vary. Many students choose to live in the neighborhoods such as Albany, Oakland, El Cerrito and Richmond. In this lab, we will explore a dataset scraped from Craigslist posting with the rent prices in Stanford and Berkeley, as well as their nearby cities. The dataset is a subset which only includes apartment/housings with less than or equal to 4 bedrooms.

The table **craigslist.csv** contains a Simple Random Sample (SRS) of the monthly rent price in 2016. Each posting record contains the following information:

- time: posting time
- price: apartment/housing monthly rent price
- size: apartment/housing size (ft^2)
- brs: number of bedrooms
- title: posting title
- link: posting link, add "https://sfbay.craigslist.org" to visit the posting page
- location: cities

Read in data.

```
craigslist <- read.csv("craigslist.csv",
                       header = TRUE, stringsAsFactors = FALSE)
#Check to see if everything looks as expected:
head(craigslist)
```

```
##                  time price size brs
## 1 2016-09-27 18:54:00  4100   NA   2
## 2 2016-09-27 18:53:00  1090  400   1
## 3 2016-09-27 17:29:00  3275  706   1
## 4 2016-09-27 17:29:00  4150  958   2
## 5 2016-09-27 17:25:00  2491  735   1
## 6 2016-09-27 17:25:00  2825  716   1
##                                                          title
## 1    Furnished Townhouse in the Elmwood  Available March 8 2017
## 2                   Private/small studio ALL UTILITIES INCLUDED
## 3     Luxury  Convenience all at Parker Movein special Hurry in
## 4 New luxury 2BR in Downtown Berkeley  1500 off 1st months rent
```

```
## 5                One Bedroom Ready For You 500 off Move In Cost
## 6                                  Ready Now for Immediate Movein
##                        link location
## 1 /eby/apa/5802541604.html berkeley
## 2 /eby/apa/5802540177.html berkeley
## 3 /eby/apa/5802443907.html berkeley
## 4 /eby/apa/5802443556.html berkeley
## 5 /eby/apa/5802429182.html berkeley
## 6 /eby/apa/5802414465.html berkeley
```

Cities included in the dataset.

```
unique(craigslist$location)
```

```
##  [1] "berkeley"            "oakland"             "albany / el cerrito"
##  [4] "richmond"            "emeryville"          "alameda"
##  [7] "palo alto"           "mountain view"       "sunnyvale"
## [10] "menlo park"          "redwood city"
```

A posting example (i.e. single observation).

```
craigslist[1, ]
```

```
##                  time price size brs
## 1 2016-09-27 18:54:00  4100   NA   2
##                                                    title
## 1 Furnished Townhouse in the Elmwood  Available March 8 2017
##                        link location
## 1 /eby/apa/5802541604.html berkeley
```

Summary statistics

Exercise 1.

Computing summary statistics is always the first step in the exploratory analysis. The summaries may include average, median, maximum, minimum, etc. One simple method is to use the `summary` function.

(a) Find out the number of postings in the dataset.

```
# insert code here
```

(b) Use `summary` function to get the mean, median, maximum and minimum of the monthly rent.

```
# Insert your code here:
```

(c) Use `table` function to get the number of postings in each city.

```
# Insert your code here:
```

(d) What percentage of entries were more than $3,000 per month?

```
# Insert code here:
```

Histograms and boxplots

Recall that many basic R plot functions accept the same options, such as:

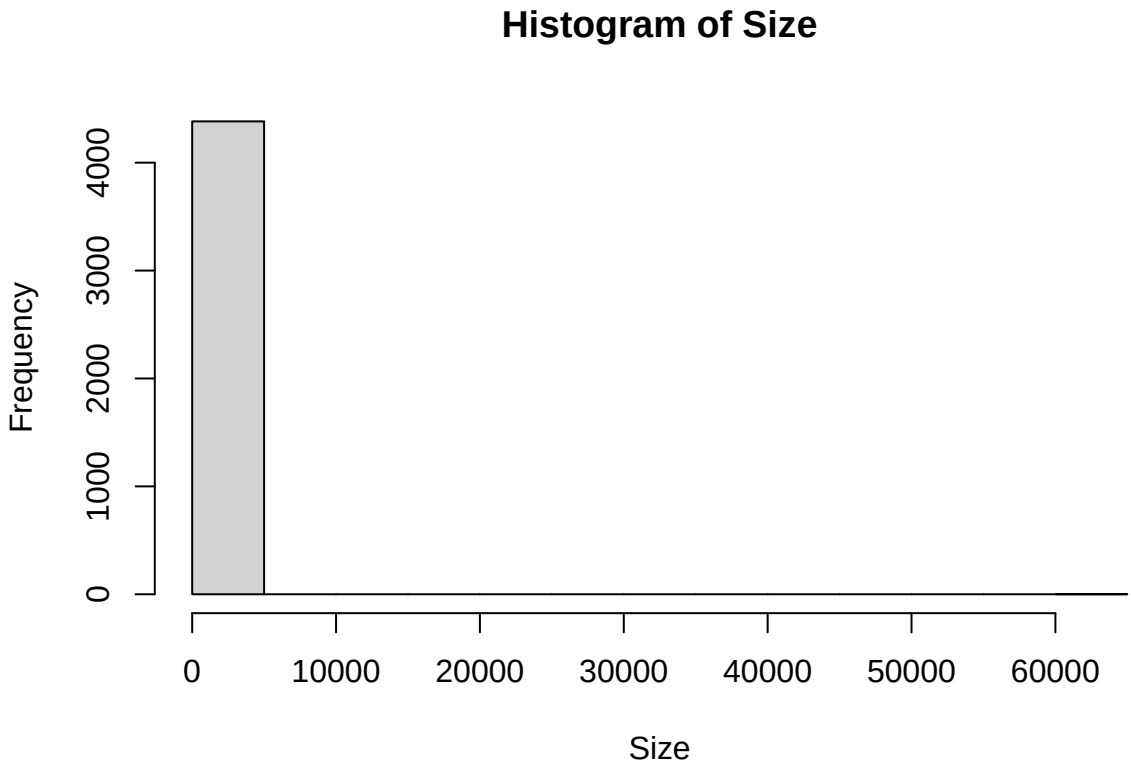
- `main`: Title of the plot.
- `xlab`/`ylab`: x/y axis label.
- `col`: plot color, the usuage of this variable may vary for different plot functions. Here the two color corresponds to female and male.
- `xlim`/`ylim`: the limits (starting and ending values) for the x/y axis.
- `legend`: the legend of the plot, the usuage of this variable may vary for different plot functions.

Always remember to add the title and axis label when you create a plot. You can check the help document (e.g `?hist`) to experience other arguments.

Histograms

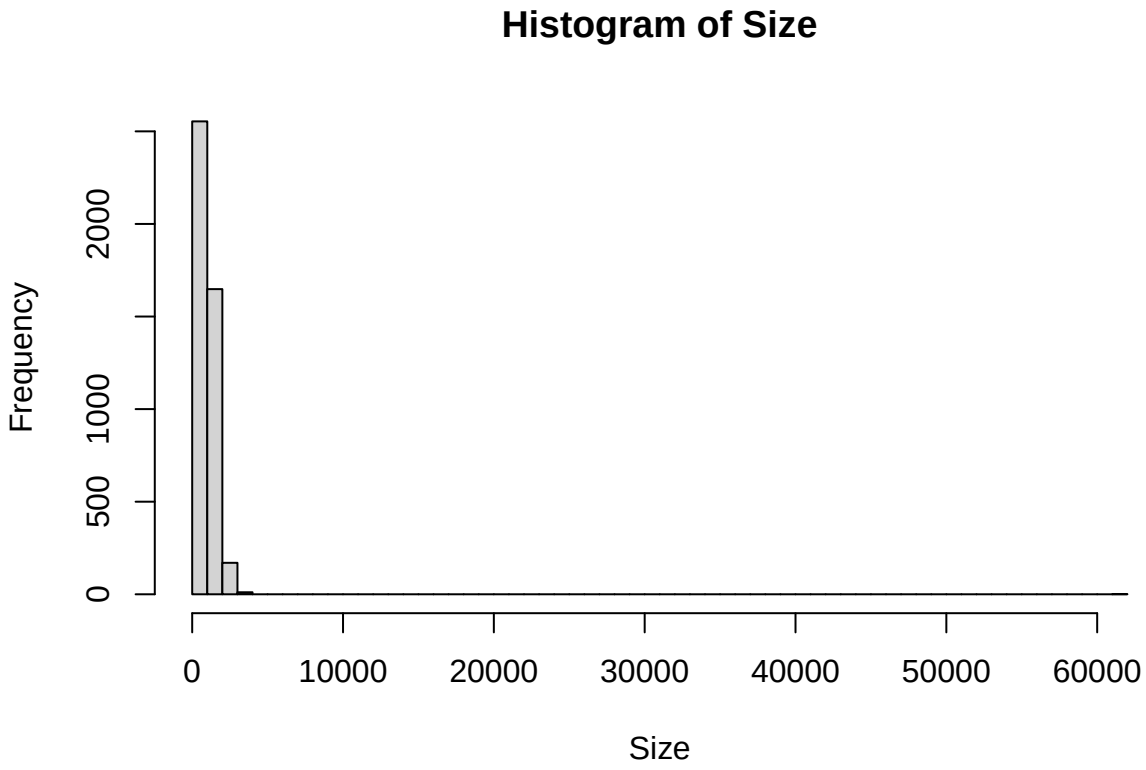
For example, to adjust plot of the histogram of our simulated `sample` in the previous probability section:

```
hist(craigslist$size,
     main = "Histogram of Size", # plot title
     xlab = "Size")
```



You can see that this is not very informative with just one big peak and then not much more. We can change the number of breaks:

```
hist(craigslist$size,
     main = "Histogram of Size", # plot title
     xlab = "Size", breaks=50)
```



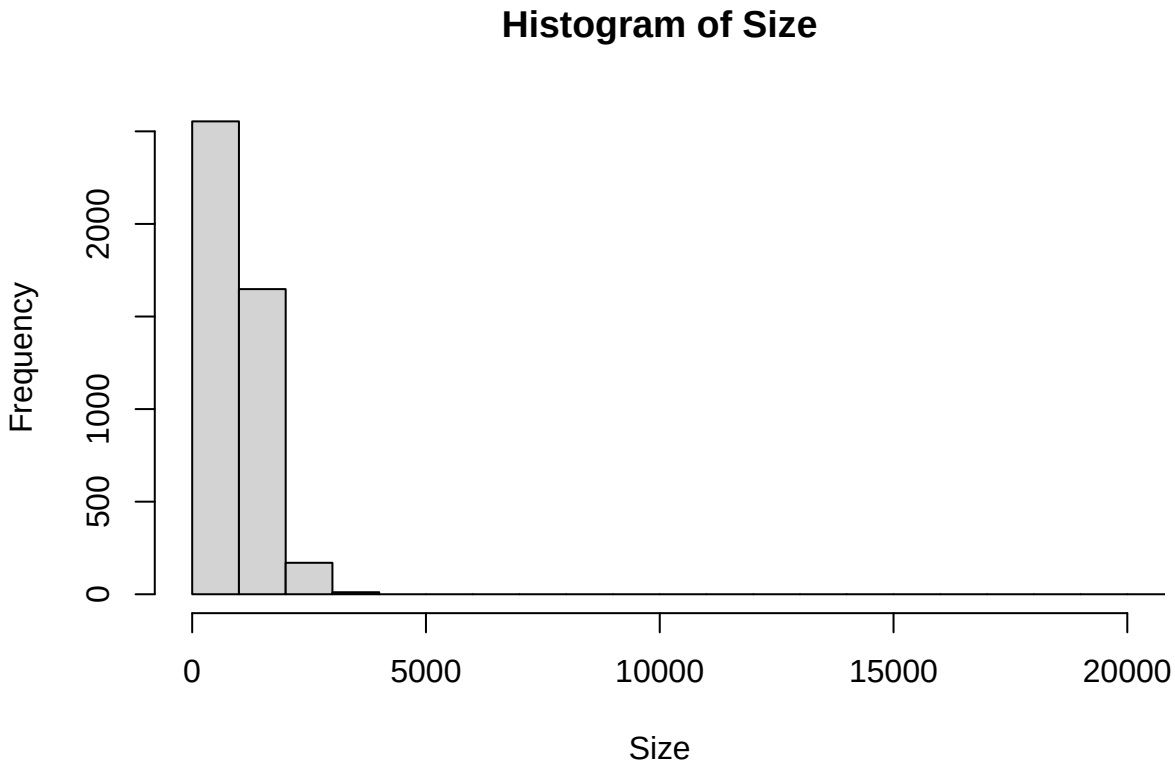
We see that most of the data is much smaller. Indeed if we do a summary, we see this:

```
summary(craigslist$size)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       1     735     947    1053    1200   62000    1492
```

We can choose to do a "zoom" on the section with the data:

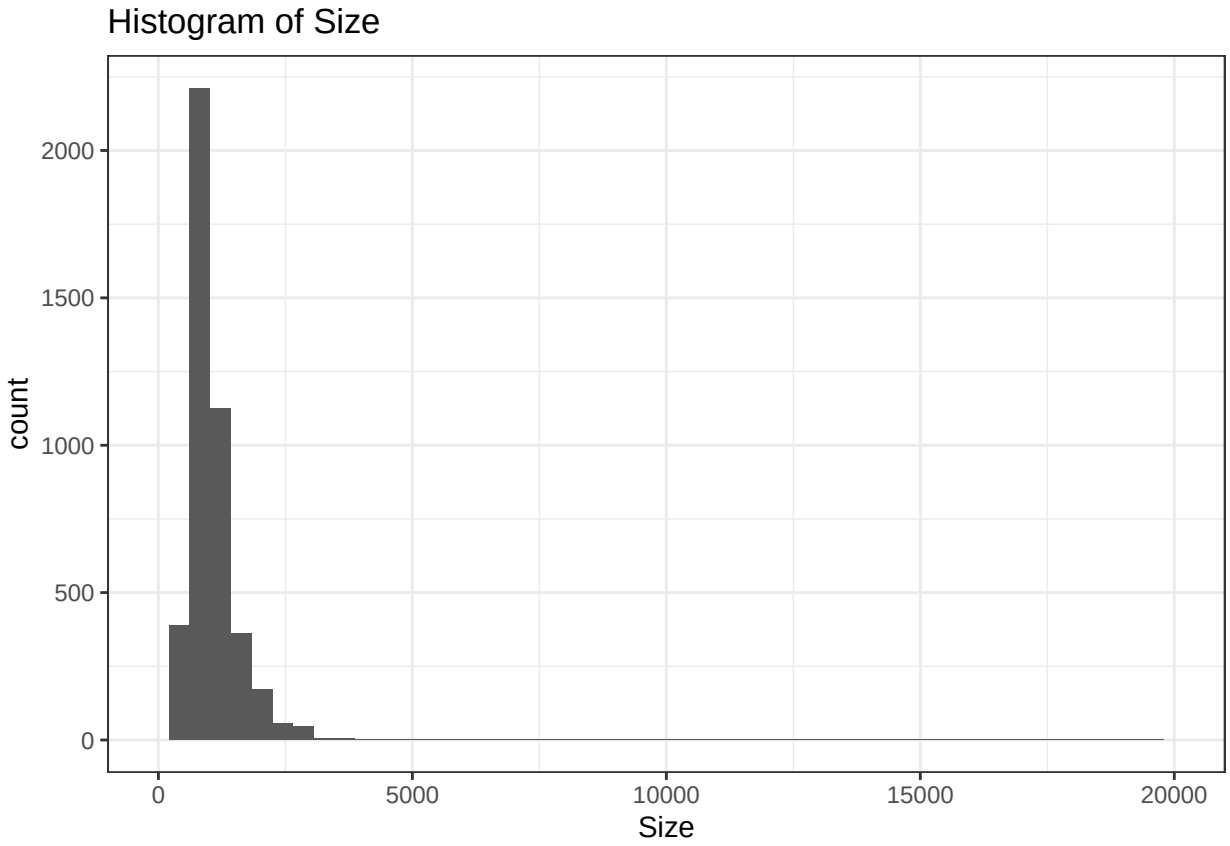
```
hist(craigslist$size,
     main = "Histogram of Size", # plot title
     xlab = "Size", breaks=50,
     xlim=c(0,20000))
```




```
# ggplot version
ggplot(data = craigslist, aes(x = size)) +
  geom_histogram(bins = 50) +
  labs(x = 'Size', title = 'Histogram of Size') +
  xlim(0,20000) + # zoom in
  theme_bw()
```

```
## Warning: Removed 1493 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



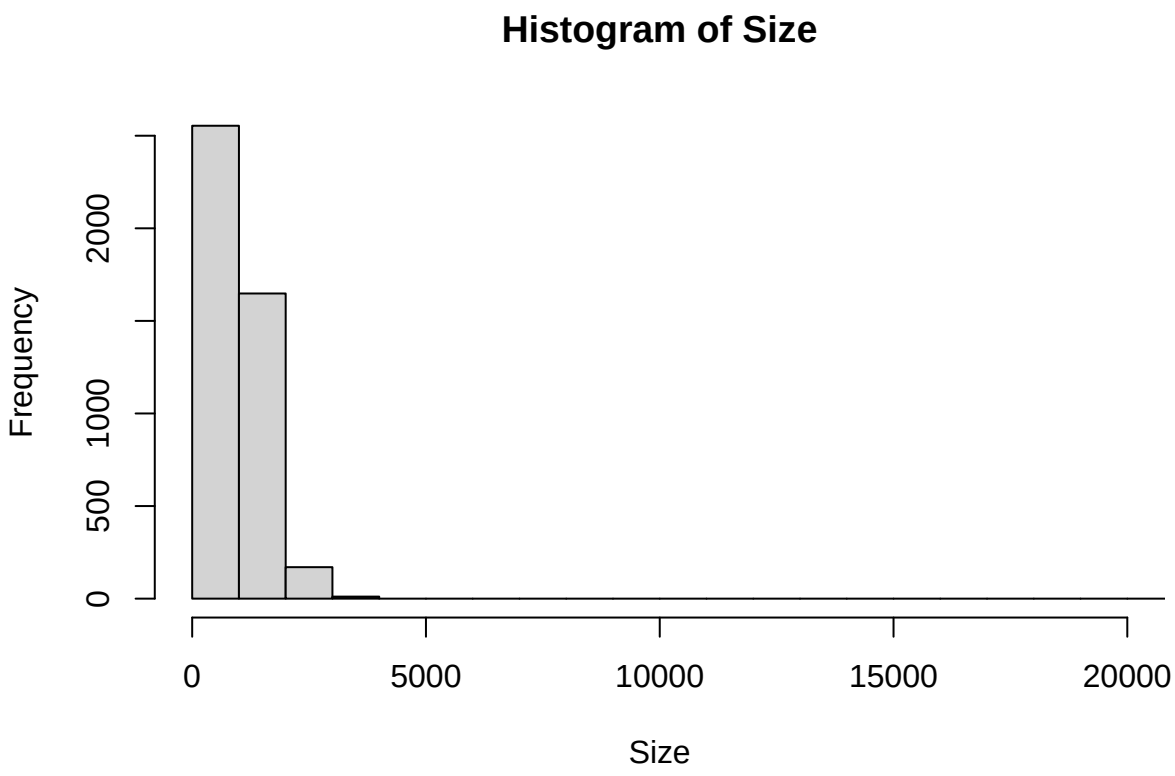

```
# warning "Removed 1492 rows containing non-finite values (stat_bin)." is about NA values
```

Notice that the `breaks` argument is based on the *whole range of the data*, and you are just zooming. So when you zoom, you need to often increase the breaks.

Exercise 2.

Fix the code above so that you have more reasonable number of breaks and zoom up further into the data. There is no singular right answer, just make it more or less reasonable than what it currently is.

```
# Fix this code:
hist(craigslist$size,
     main = "Histogram of Size", # plot title
     xlab = "Size", breaks=50,
     xlim=c(0,20000))
```



Exercise 3.

(a) Plot a histogram of the monthly rent. Choose a number of breaks you think best from 10, 50, 300. Not all choices will be given full credit.

```
#Insert code here:
```

(b) Plot another histogram with the argument `freq = FALSE`. What is the interpretation of y axis numbers? Hint, be careful, there are two common answers, one of which is wrong, but can be easily confused with the right answer.

Hint: It is helpful to check the help document using `?` if you are confused about one argument or usage. Always remember to add the title and axis label when you create a plot.

```
#Insert code here:
```

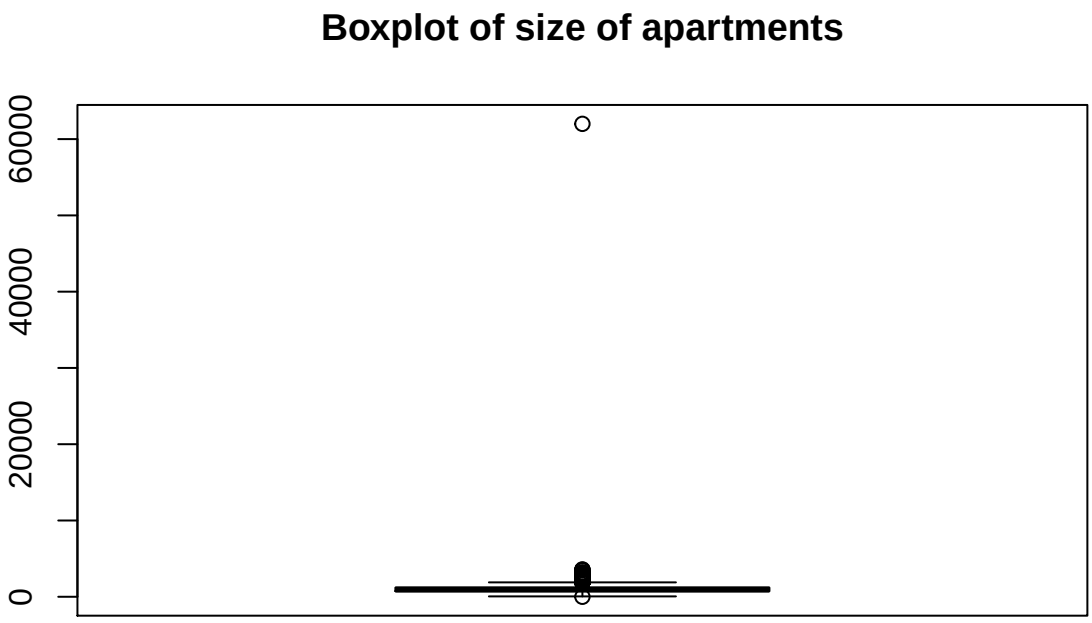
Your non-coding answer:

```
*Add text here*
```

Boxplots

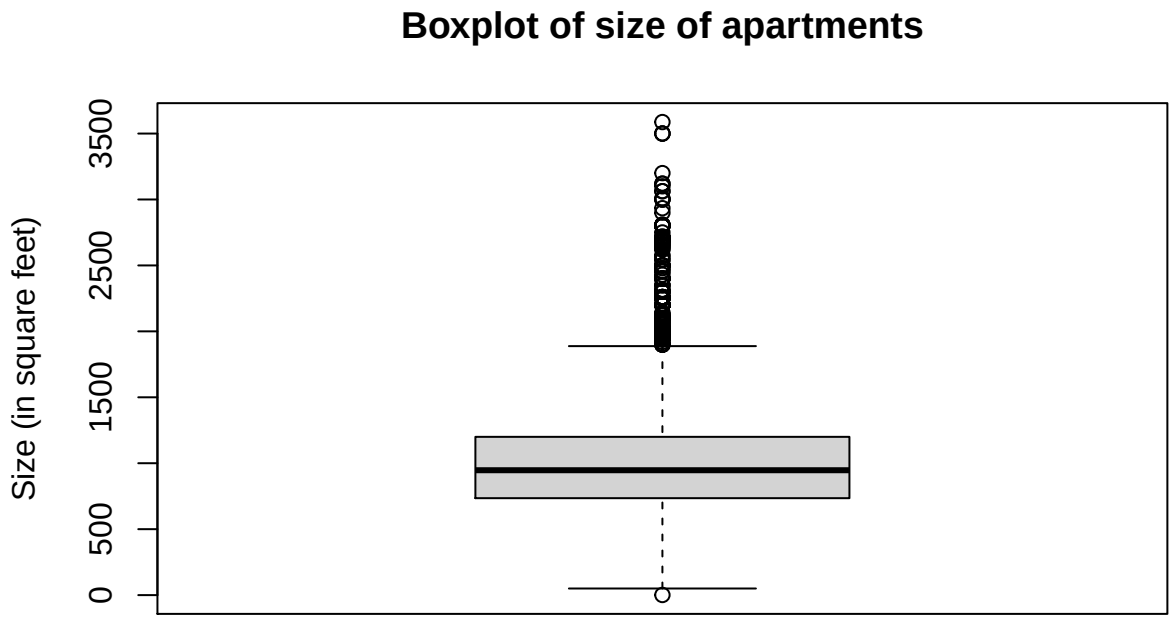
For a boxplot, we use the `boxplot` function.

```
boxplot(craigslist$size,
     main = "Boxplot of size of apartments"
)
```

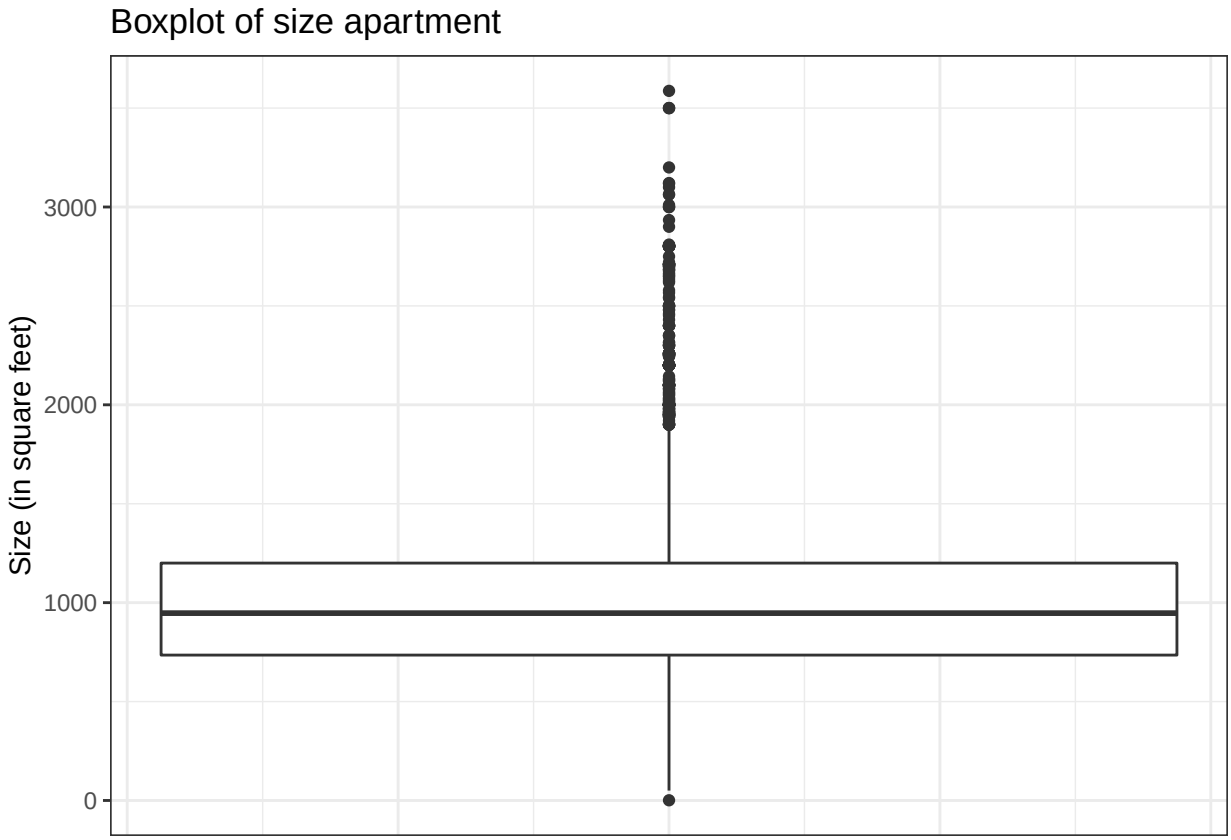


Notice how we can really see that one observation was messing up our histogram, and now our boxplot too. Let's just take that observation out

```
boxplot(craigslist$size[craigslist$size<60000],
     main = "Boxplot of size of apartments",
     ylab="Size (in square feet)"
)
```

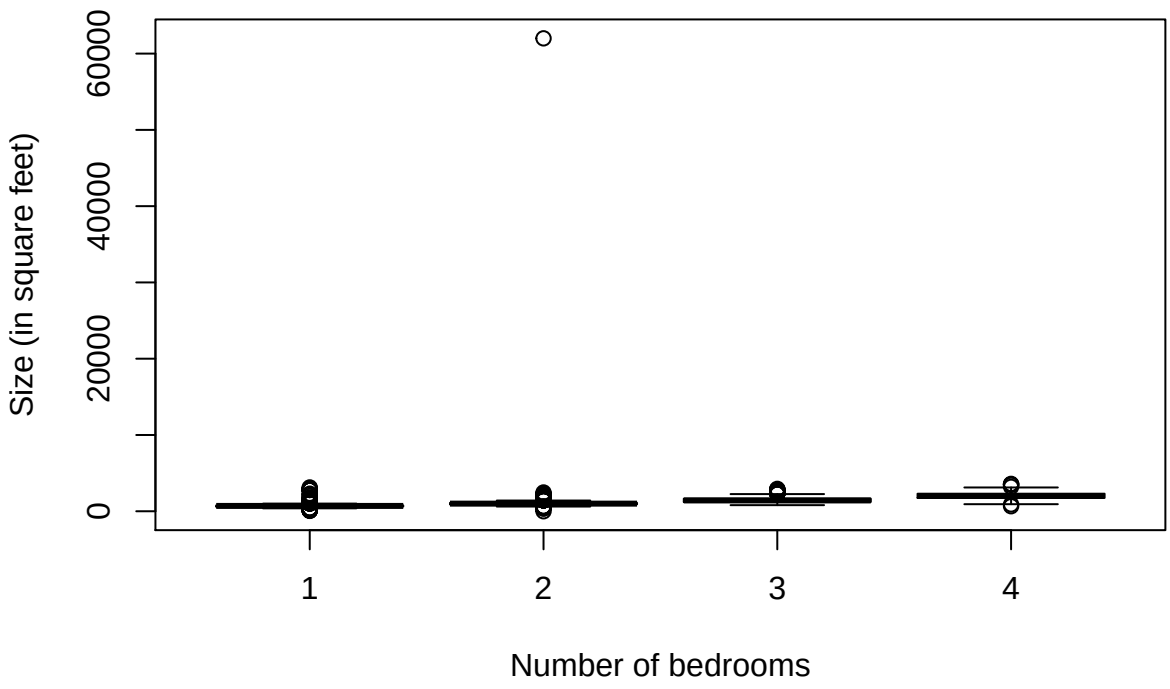



```
ggplot(craigslist %>% filter(size < 60000), aes(y = size)) +
  geom_boxplot() +
  labs(y = 'Size (in square feet)', title = 'Boxplot of size apartment') +
  theme_bw() + # theme_bw needs to be before theme
  theme(axis.ticks.x = element_blank(), axis.text.x = element_blank()) # remove x axis ticks and labels
```



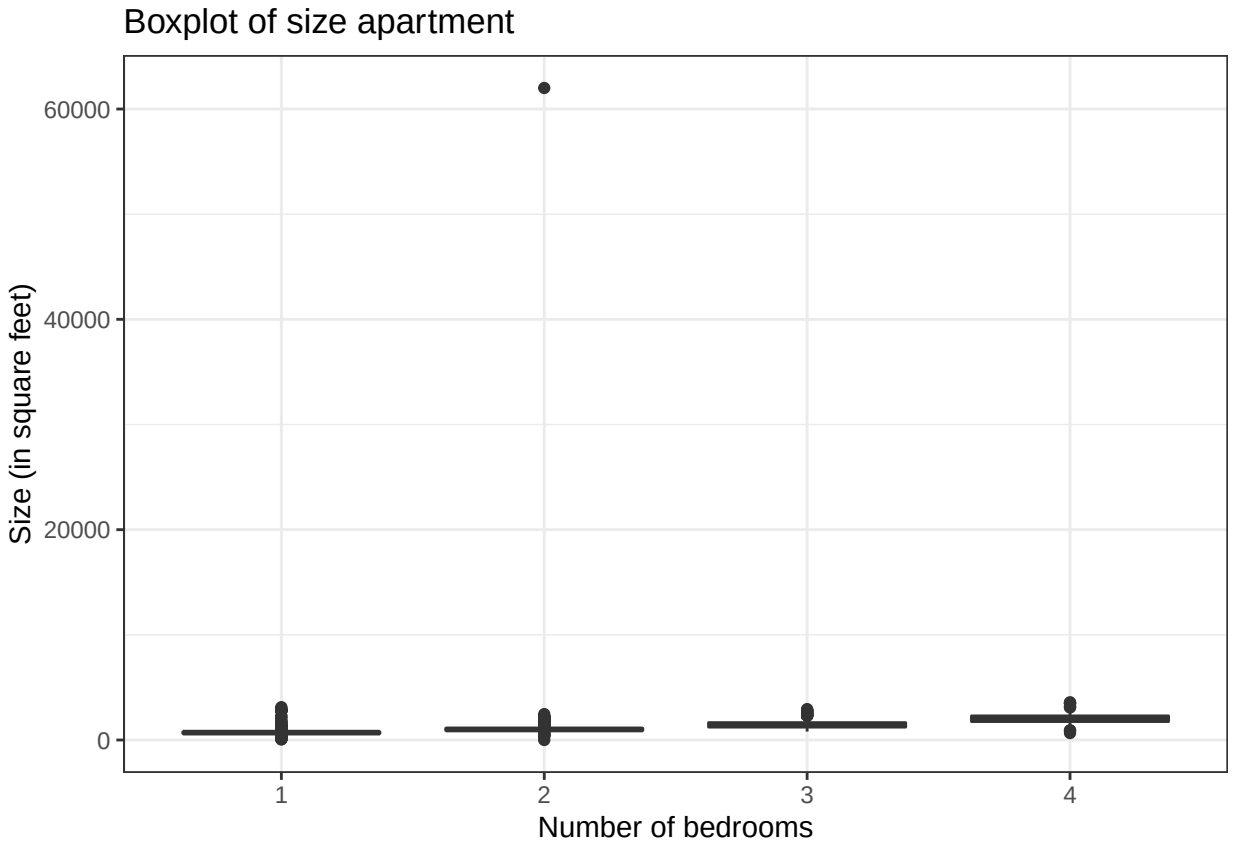
If we have a factor vector that separates the observations into different categories, we can create side-by-side boxplots. You do this by the formulation `y ~ fac`.

```
boxplot( craigslist$size ~ craigslist$brs ,
     main = "Boxplot of size of apartments",
     ylab="Size (in square feet)",
     xlab="Number of bedrooms"
)
```




```
ggplot(craigslist, aes(x = factor(brs), # brs is a vector of numbers, we need to convert it to
                                        # factor datatype using factor()
                       y = size)) +
  geom_boxplot() +
  labs(x = 'Number of bedrooms',
       y = 'Size (in square feet)',
       title = 'Boxplot of size apartment') +
  theme_bw() # theme_bw needs to be before theme
```

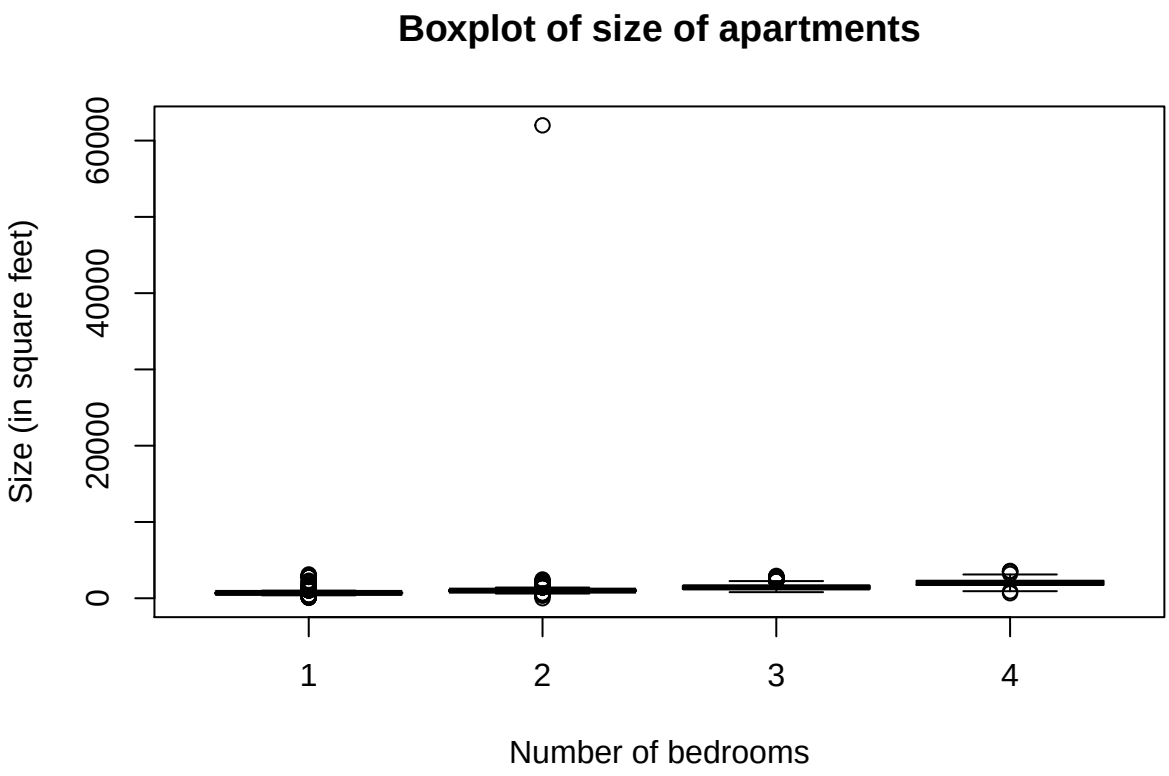
```
## Warning: Removed 1492 rows containing non-finite values (stat_boxplot).
```




```
# warning "Removed 1492 rows containing non-finite values (stat_boxplot)." is about NA values
```

When both variables are stored in the same data.frame, we can also do the following (more elegant) method:

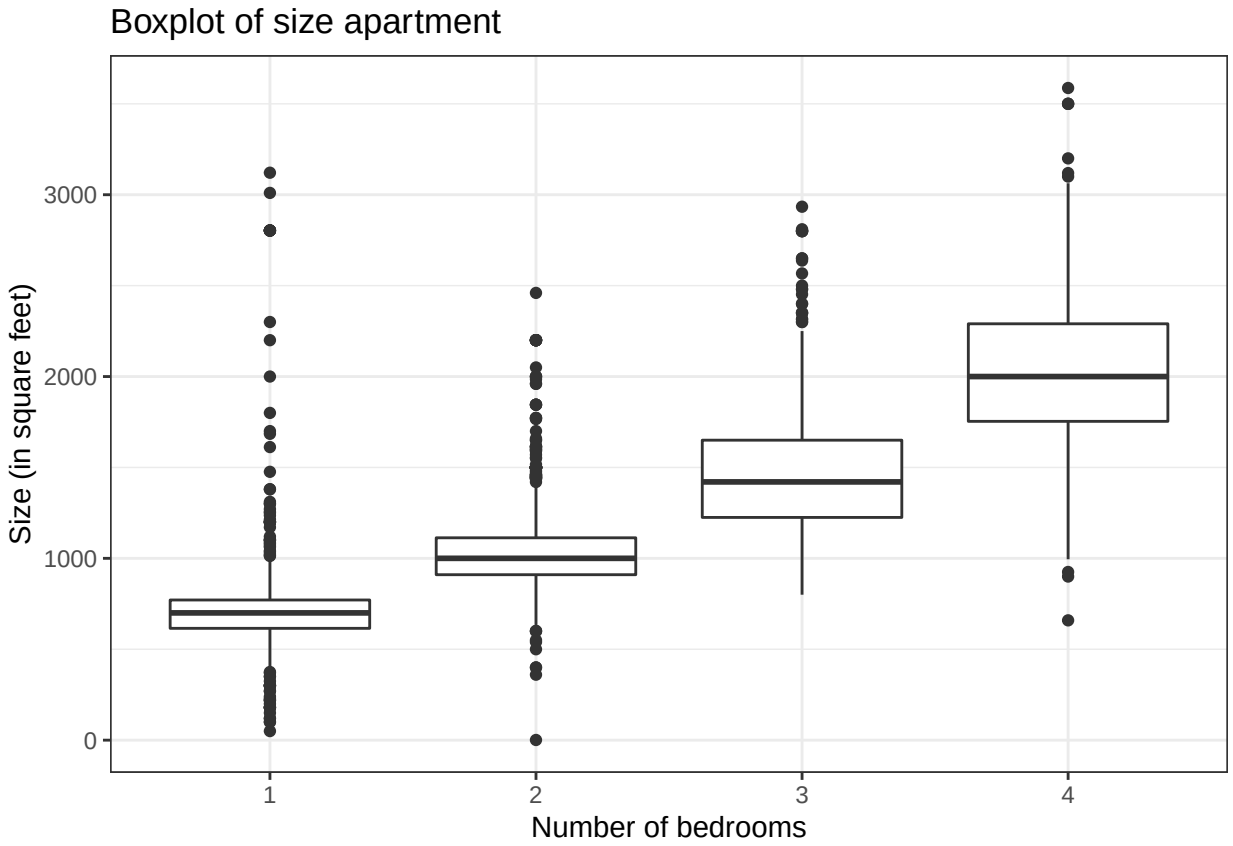
```
boxplot(size ~ brs, data=craigslist,
     main = "Boxplot of size of apartments",
     ylab="Size (in square feet)",
     xlab="Number of bedrooms"
)
```



Notice how this makes it easy to repeat the plot without that outlier by just subsetting the dataset given as input:

```
boxplot(size ~ brs, data=craigslist[craigslist$size<60000,],
     main = "Boxplot of size of apartments",
     ylab="Size (in square feet)",
     xlab="Number of bedrooms"
)
```


```
ggplot(data = craigslist %>% filter(size < 60000), # use dplyr to filter
       aes(x = factor(brs), y = size)) +
  geom_boxplot() +
    labs(x = 'Number of bedrooms',
       y = 'Size (in square feet)',
       title = 'Boxplot of size apartment') +
  theme_bw()  # theme_bw needs to be before theme
```


Boxplot of size apartment
Size (in square feet)
3000
2000
1000
0
1
2
3
4
Number of bedrooms

Exercise 4.

(a) Subset to only the one bedroom postings.

```
# insert code here
```

(b) Draw the boxplot of one bedroom rent price for each cities side by side.

```
#Insert code here
```

(c) Can you find the link of the outlier with monthly rent larger than 5000 in the boxplot?

```
# insert code here
```

Probability Distributions

There are a large number of standard parametric distributions available in R (nearly every common distribution!). To get a list of them, you can do:

```
?Distributions
```

You are probably only familiar with the normal distribution. But each of them is immensely useful in statistics. You will see Chi-squared distribution, student-t distribution later in this course. Every distribution has four functions associated with it.

Name	Explanation
d	density: Probability Density Functions (pdfs)
p	probability: Cumulative Distribution Functions (cdfs)
q	quantile: the inverse of Probability Density Functions (pdfs)
r	random: Generate random numbers from the distribution

Density (pdf) Take normal distribution $N(\mu, \sigma^2)$ for example. Let's use the `dnorm` function to calculate the density of $N(1, 2)$ at $x = 0$:

```
dnorm(0, mean = 1, sd = sqrt(2))
```

```
## [1] 0.2196956
```

```
# Notice here the parameter in dnorm is sd,
# which represents the standard deviation (sigma instead of sigma^2).
```

Looks familiar? It is exactly same the function that you wrote in lab 0! This function can accept vectors and calculate their densities as well.

Cummulative Distribution Functions (cdf) The Cummulative Distribution Functions (cdfs) gives you the probability that the random variable is less than or equal to value:

$$P(X \leq z)$$

where z is a constant and X is the random variable. For example, in the above example. $N(1, 2)$ is symmetric about 1. Then what would be the probability that it is less than 1?

```
probs <- pnorm(1, mean = 1, sd = sqrt(2))
probs
```

```
## [1] 0.5
```

Quantiles `qnorm` is the inverse function of `pnorm`. It accepts value p (probability) from 0 to 1, and returns a value q (quantile) satisfies the following condition:

$$P(X \leq q) = p$$

where X is the random variable. A toy example would be:

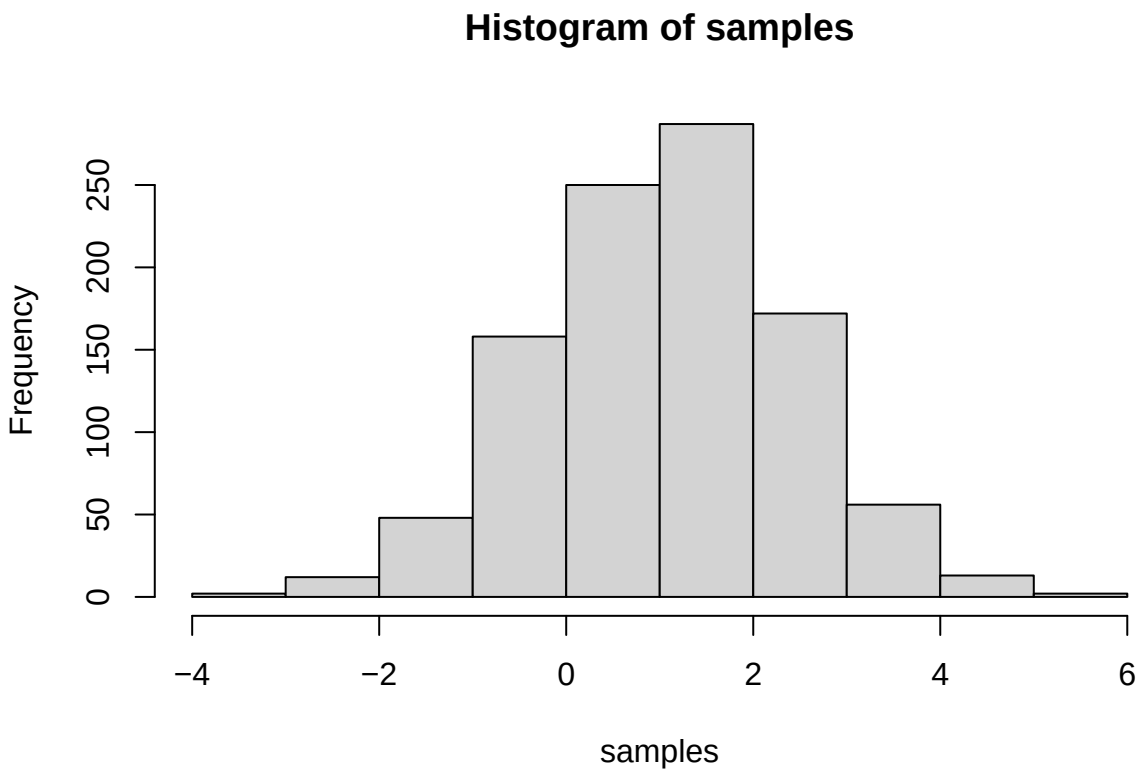
```
qtl <- qnorm(0.5, mean = 1, sd = sqrt(2))
qtl
```

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

Quantiles are very important in hypothesis testing, for example if you want to know what value of a test-statistic would give you a certain p-value.

Random number generation The last function is very important when you simulate stuff. We use it to generate numbers from a distribution.

```
# generate 1000 samples from Normal(1, 2) distribution.
samples <- rnorm(n= 1000, mean = 1, sd = sqrt(2))
# plot the histogram
hist(samples)
```



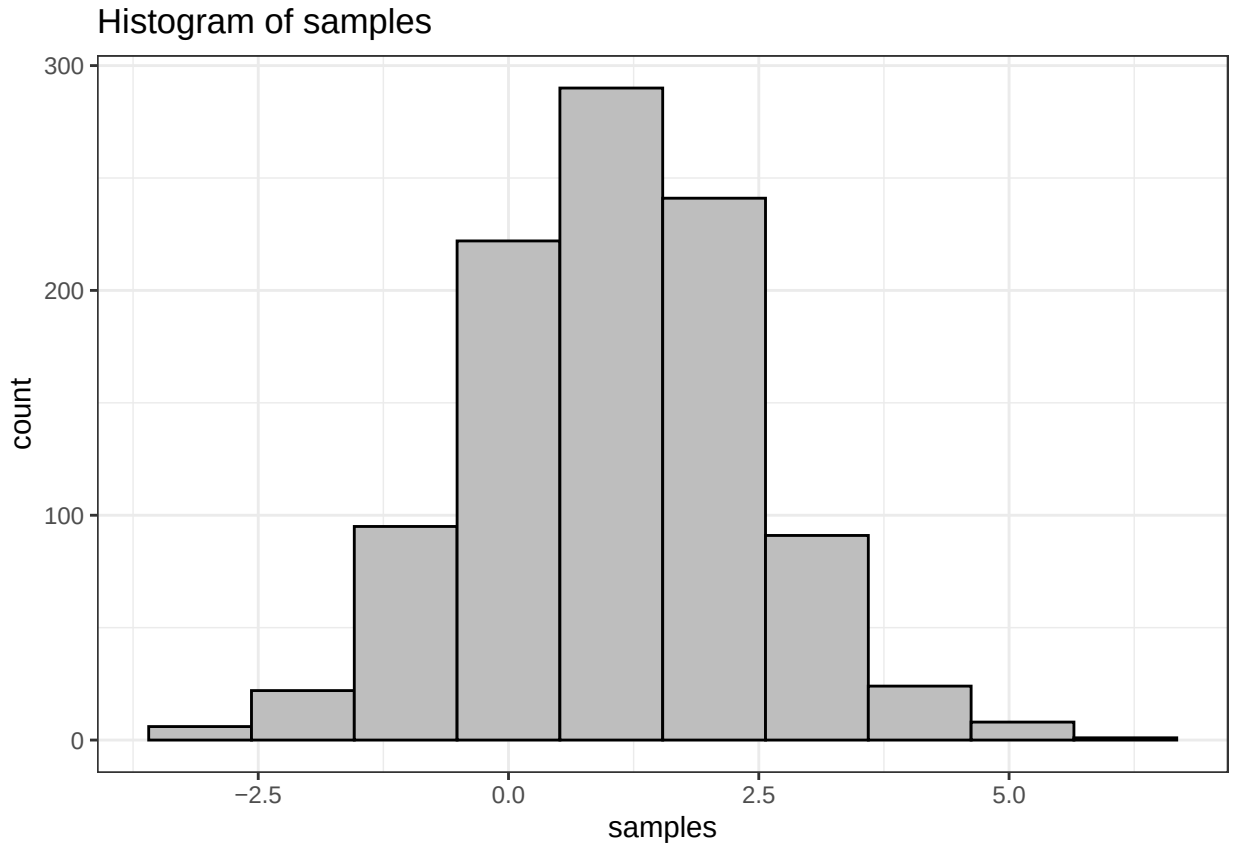
```
class(craigslist)
```

```
## [1] "data.frame"
```

```
class(samples)
```

```
## [1] "numeric"
```

```
sample_df = data.frame(sample = samples)
ggplot(sample_df, aes(x = samples)) +
  geom_histogram(bins = 10,
                 fill = 'grey', # fill in color
                 color = 'black') + # border color
  theme_bw() +
  labs(title = 'Histogram of samples')
```



Plotting a function

Let's plot the line of this function (you can also use the `curve` function that the professor used for the lecture notes to draw functions):

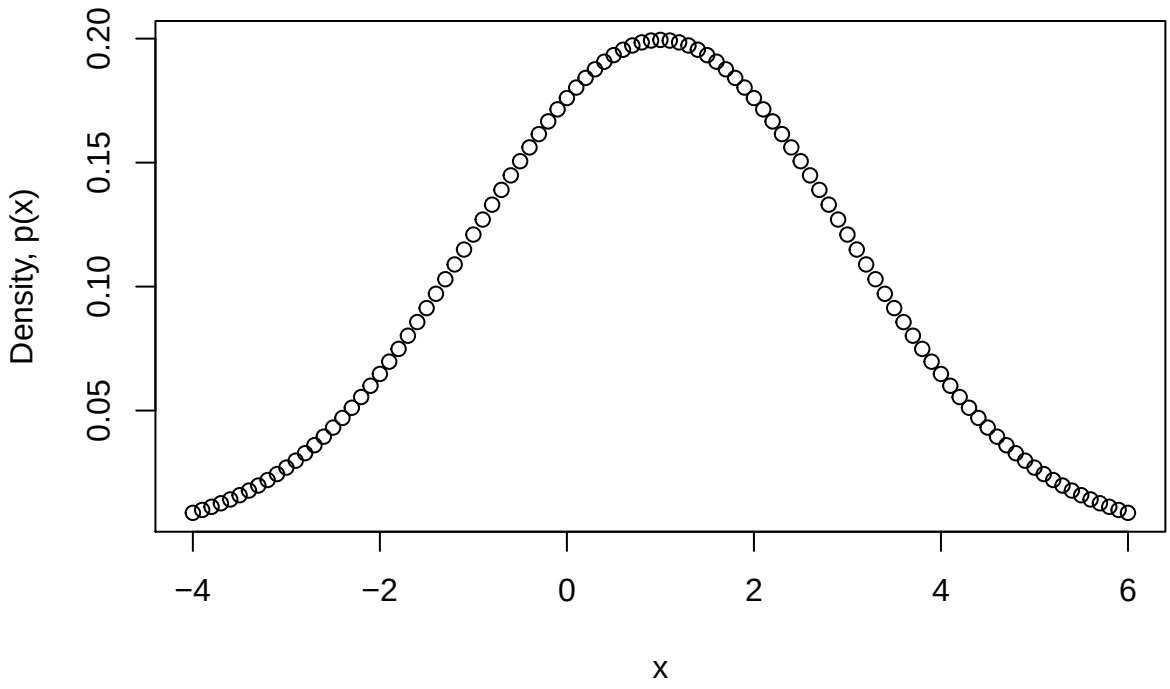
We will be manually recreating what `curve` does: we will create a sequence of x values and then evaluate f(x) to get the corresponding y vector of values, and then plot them with `plot`

First we create the sequences

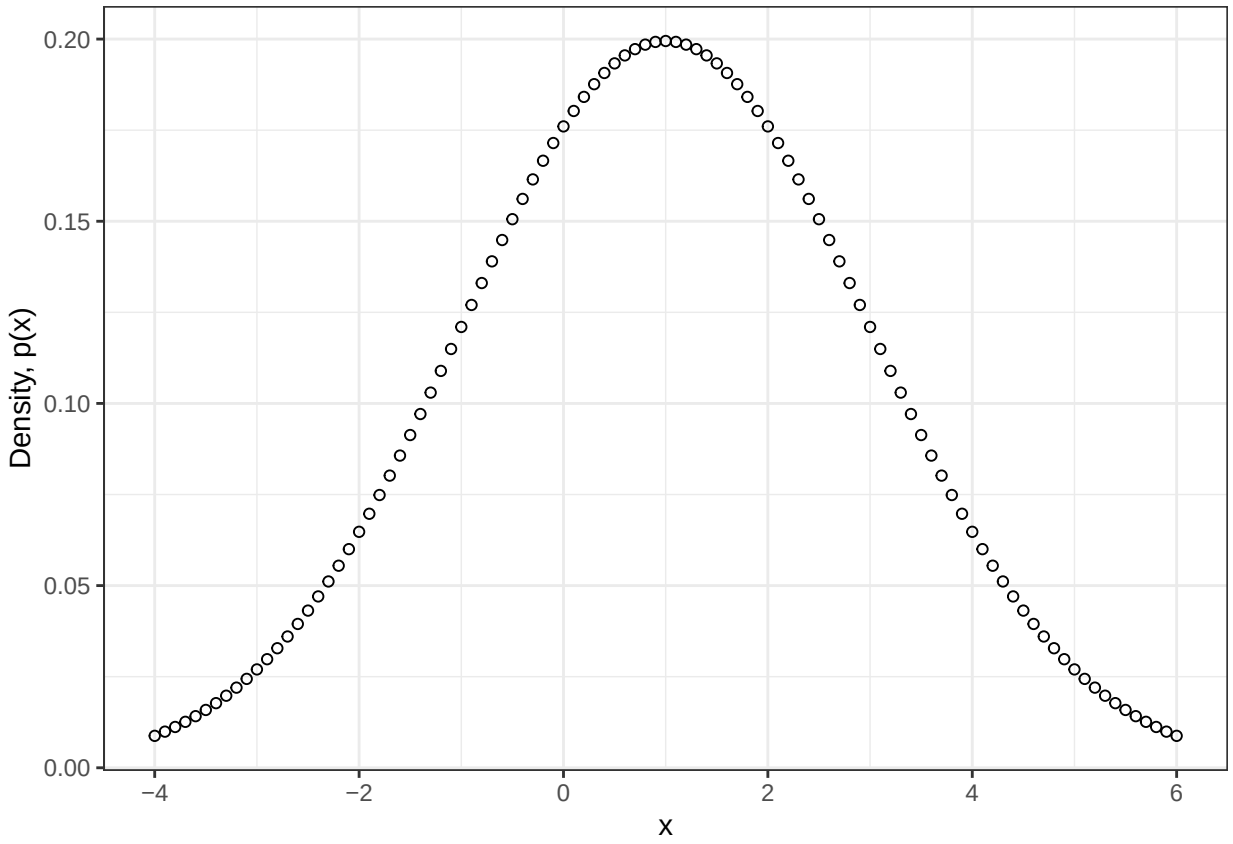
```r
# create a vector, spaced by 0.1
x <- seq(from = -4, to = 6, by = 0.1)
# calculate their densities
x.dens <- dnorm(x, mean = 1, sd = 2)
```

Now we will plot them

```r
# plot the line
plot(x, x.dens, ylab="Density, p(x)")
```

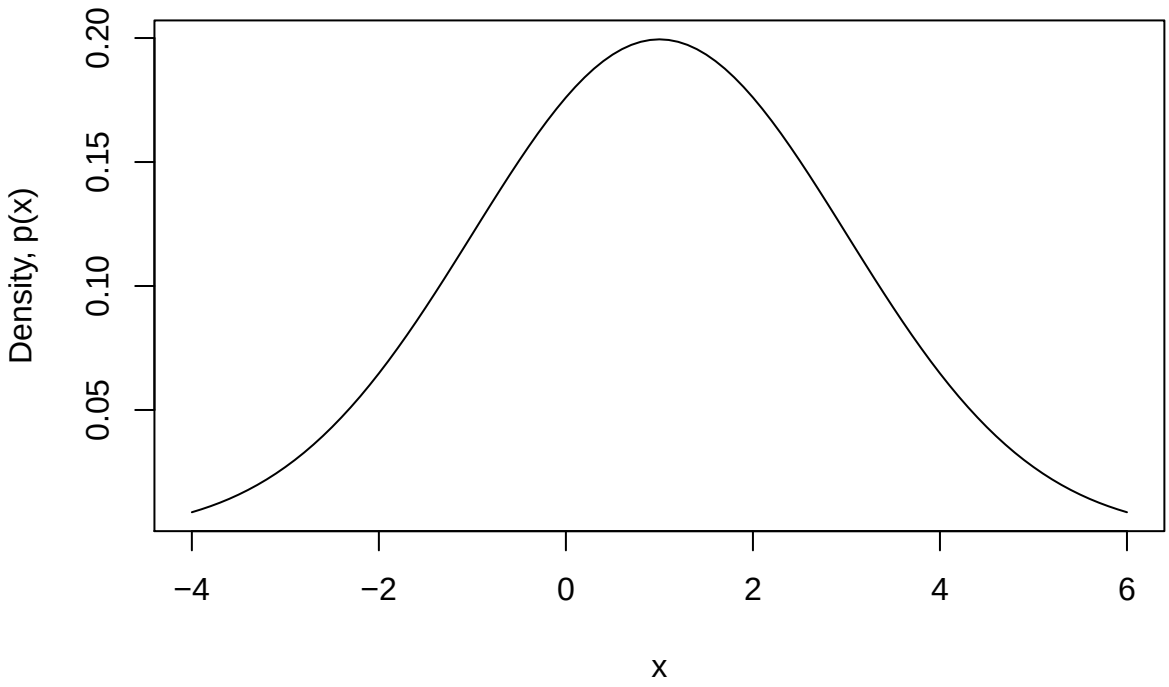



```r
x_df = data.frame(x = x, x.dens = x.dens)
ggplot(x_df, aes(x, x.dens)) +
  geom_point(shape = 1) +
  labs(y = 'Density, p(x)') +
  theme_bw()
```

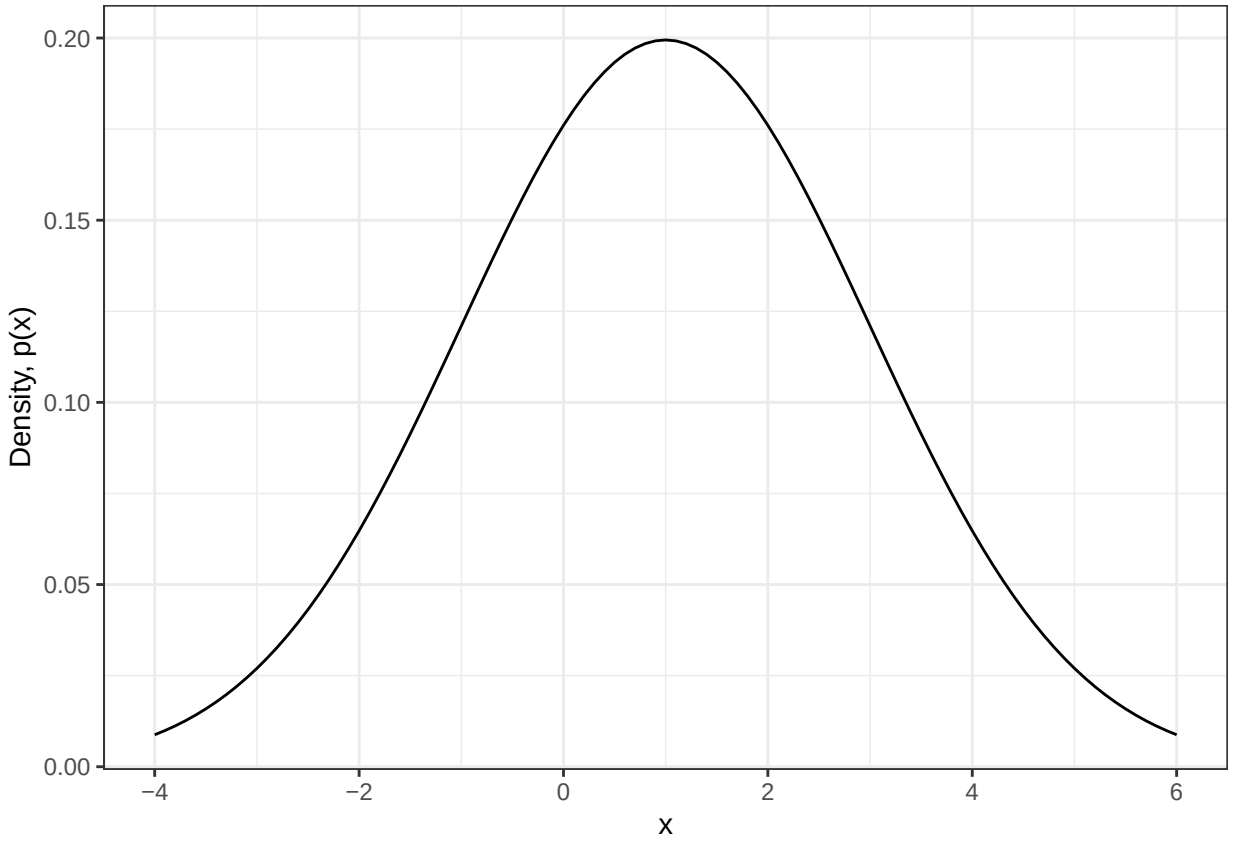


We'd really rather plot a curve, than these points. We can do this by setting `type="l"`; this makes `plot` act like the function `lines`, with the difference that it doesn't have to add to an existing plot.

```
# plot the line
plot(x, x.dens,type="l", ylab="Density, p(x)",xlim=c(-4,6))
```

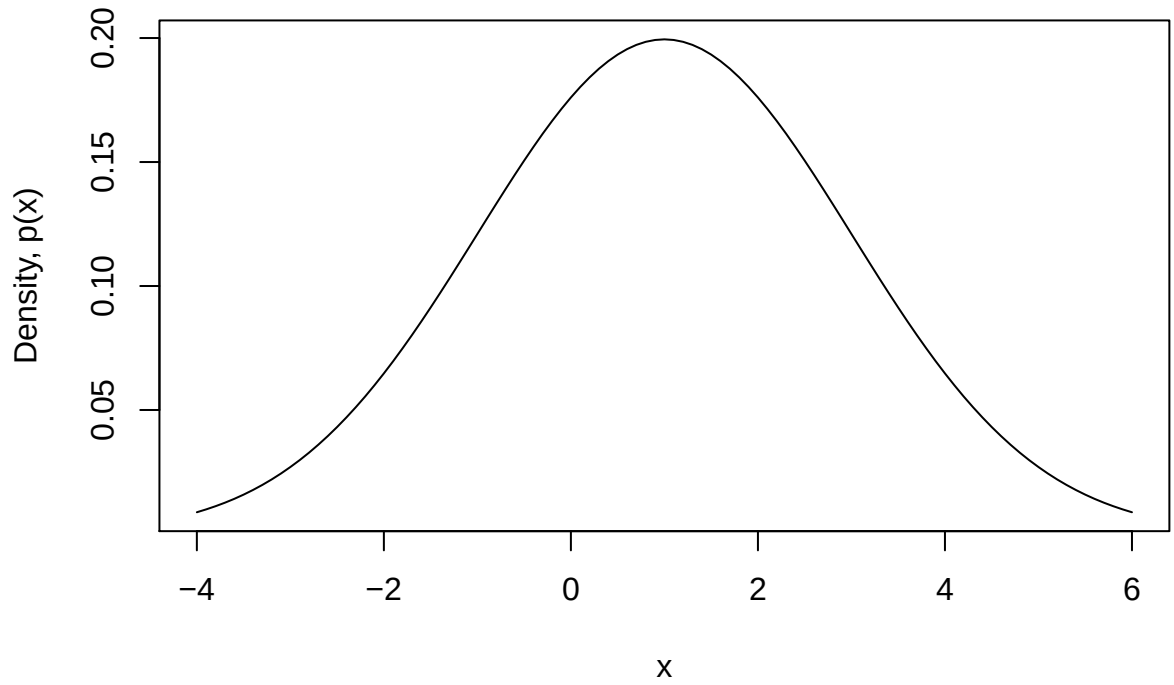


```
ggplot(x_df, aes(x, x.dens)) +
  geom_line() +
  labs(y = 'Density, p(x)') +
  theme_bw()
```



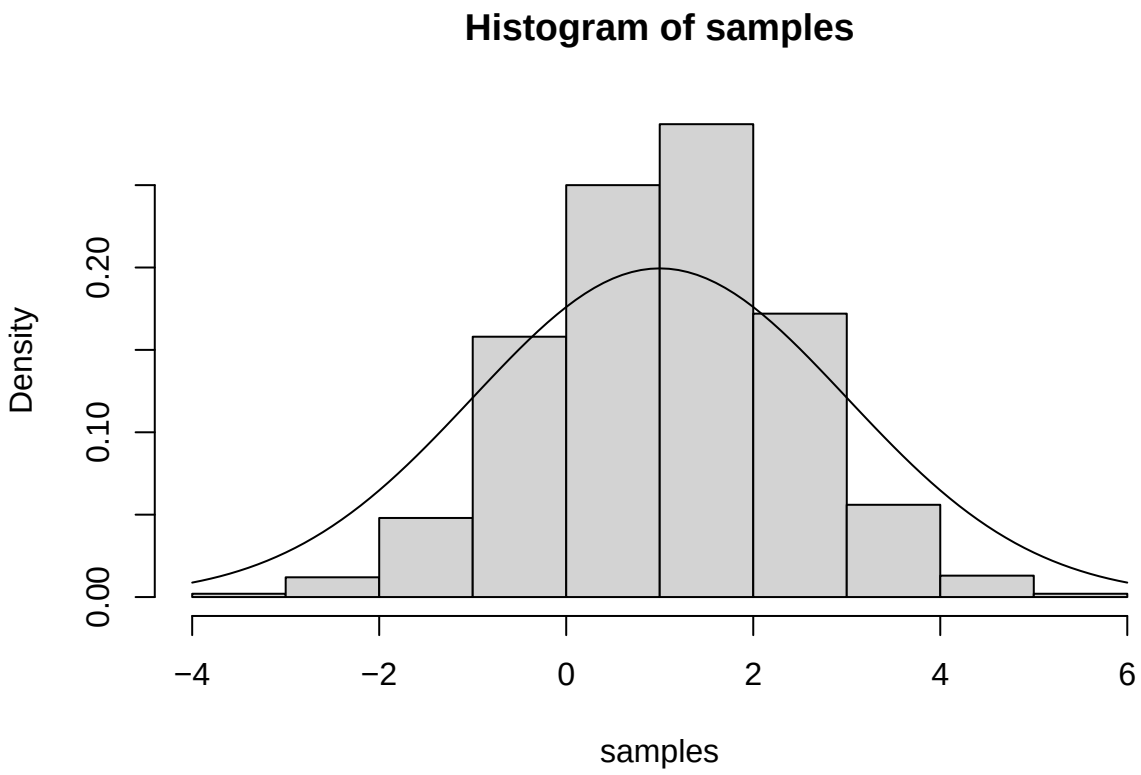
It's useful to know how to manually plot function as described above, but for simple functions `curve` can be clearer

```
# plot the line
curve(dnorm(x, mean = 1, sd = 2),from=-4,to=6,
      ylab="Density, p(x)")
```

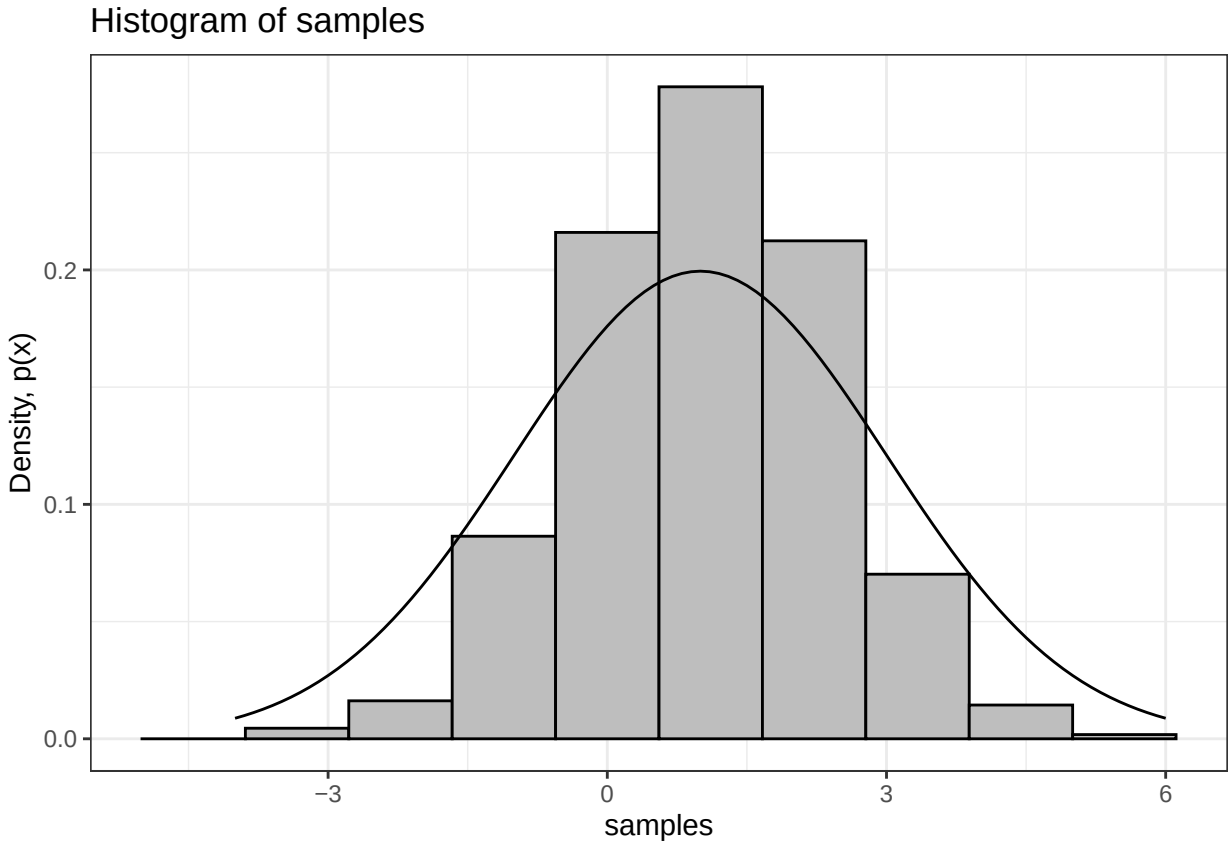


We can also overlay the density on top of the histogram, but we need to make sure the histogram is on the density scale:

```
hist(samples, freq=FALSE,xlim=c(-4,6))
curve(dnorm(x, mean = 1, sd = 2),
      ylab="Density, p(x)", add=TRUE)
```



```
ggplot(sample_df, aes(samples, y = ..density..)) +
  geom_histogram(bins = 10, fill = 'grey', color = 'black') +
  geom_line(aes(x = x, y = x.dens),data = x_df) +
  labs(y = 'Density, p(x)', title = 'Histogram of samples') +
  theme_bw()
```



Exercise 5.

(a) X_1 follows Normal Distribution $N(3.5, 9)$. What is the probability that $-2.5 < X \leq 9.5$?

```
# insert code here
```

(b) X_2 follows Normal Distribution $N(3.5, 9)$. Theoretically, what is the expected value of the interquartile range (IQR) if we plot samples from X_2? (HINT: IQR = 0.75 quantile - 0.25 quantile.)

```
# insert code here save
```

Set Seed

The random numbers and random samples generated in R are produced by a random number generator. The process is not really "random", but mimic the results of what we would get from the random process. Thus, unlike a truly random process, the random numbers can be tracked and be exactly reproduce. For example, if you run a permutation test function for two times, you would get two very close but different p-values. But if you set the seed to be the same number before you run the permutation test, you would obtain the exact same p-values. Throughout the rest of the course where random number generation is involved, we would set seed of the random number generator such that the results are fully reproducible (important for grading purposes!). However, in the real application, you would generally change it.

The following chunk illustrate how `set.seed` influence the random number generation.

```
set.seed(201728)
sample(x = 1:5, size = 3) # after calling this function, the seed will be updated
```

```
## [1] 4 3 2
```

```
sample(x = 1:5, size = 3) # the seed has changed thus we would get a different number
```

```
## [1] 1 3 4
```

```
set.seed(201728) # set the seed back to 201728
sample(x = 1:5, size = 3)
```

```
## [1] 4 3 2
```

It is the same with `rnorm`:

```
set.seed(20170126)
rnorm(5, mean = 0, sd = 1)
```

```
## [1] -1.1609512  0.6712777 -0.2232760 -2.2804950  0.1142109
```

```
rnorm(5, mean = 0, sd = 1)
```

```
## [1]  0.1279252  0.6195922  2.4062442 -0.4081882  0.3705725
```

```
set.seed(20170126)
rnorm(5, mean = 0, sd = 1)
```

```
## [1] -1.1609512  0.6712777 -0.2232760 -2.2804950  0.1142109
```