LAB 2

STAT 131A

Welcome to the lab 2! In this lab, you will

1) review probability;
2) review normal distributions
3) Obtain some basic statistics by groups;

We will continue to use the rent price dataset from the lab 1. In the table **craigslist.csv**, each posting record (row) 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. Create `one.bedrooms` data frame of only postings for 1 bedroom.

```
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
```

```
one.bedrooms <- craigslist[craigslist$brs == 1,]
```

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:

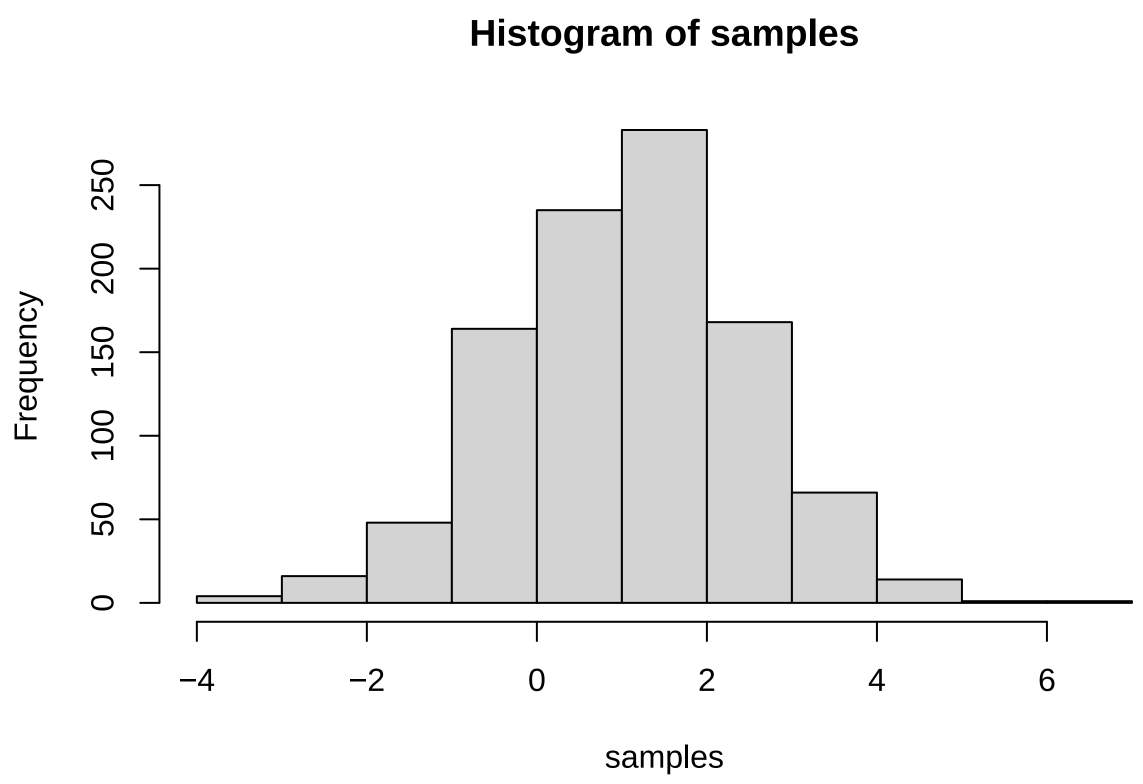
```
qtl <- qnorm(0.5, mean = 1, sd = sqrt(2))
qtl
```

```
## [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.

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



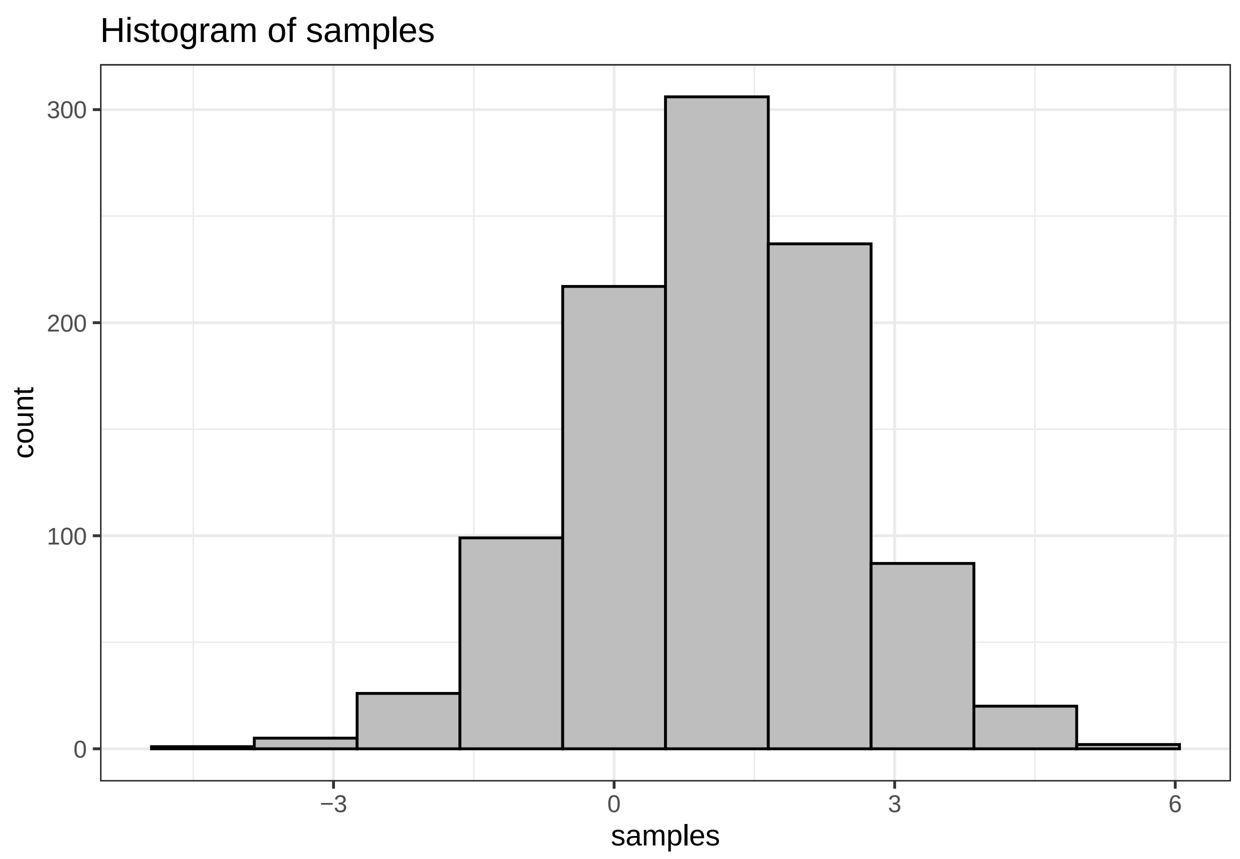
```r
class(craigslist)
```

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

```r
class(samples)
```

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

```r
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):

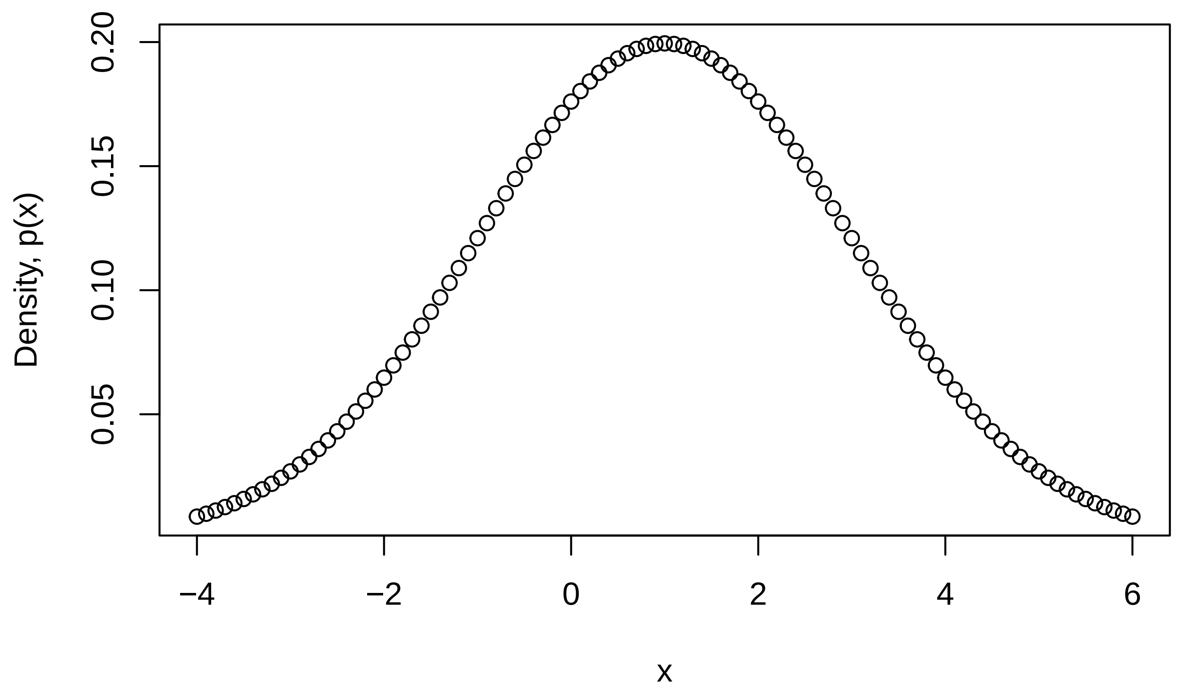
We will be manually recreating what `curve` does: we will create a sequence of x values and then evaluate f(x) to get the corresponding y vector of values, and then plot them with `plot`

First we create the sequences

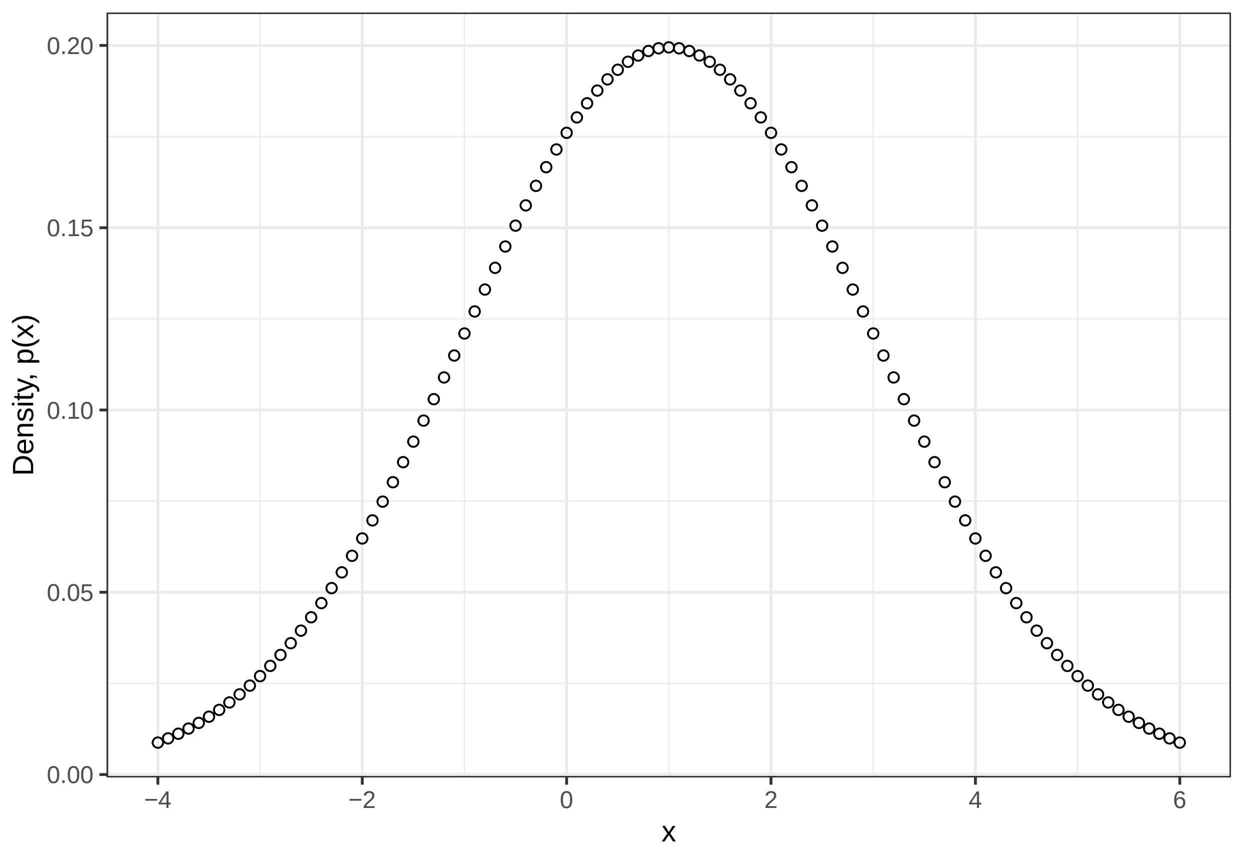
```r
# create a vector, spaced by 0.1
x <- seq(from = -4, to = 6, by = 0.1)
# calculate their densities
x.dens <- dnorm(x, mean = 1, sd = 2)
```

Now we will plot them

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

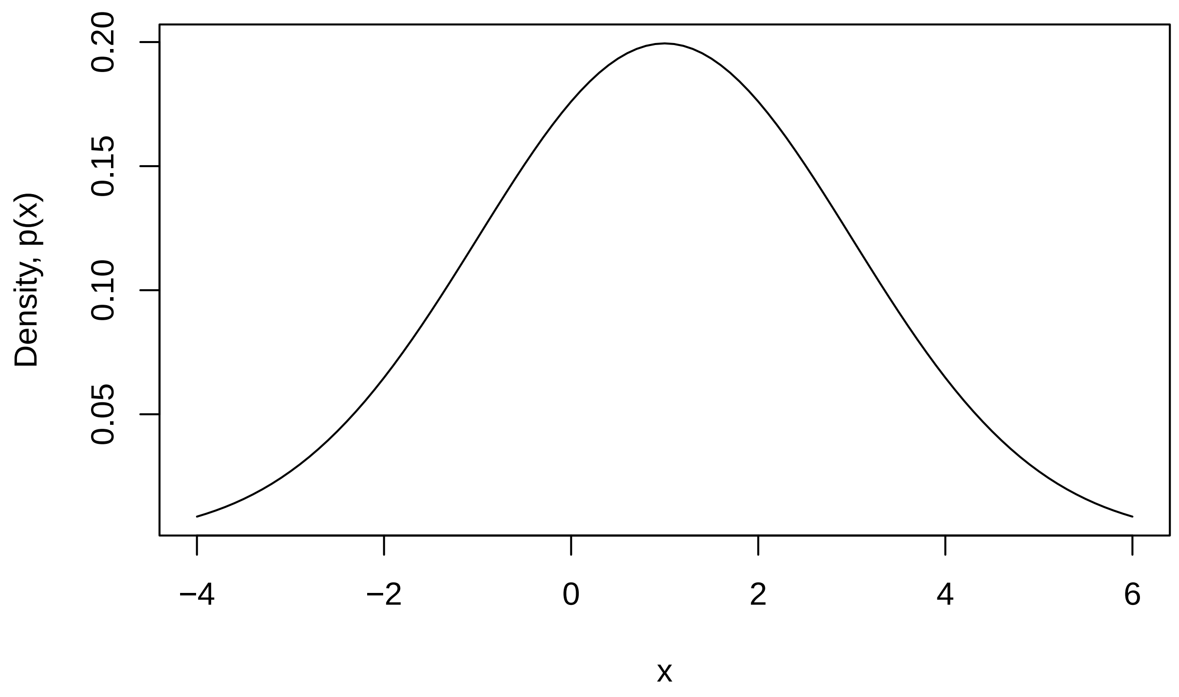


```
x_df = data.frame(x = x, x.dens = x.dens)
ggplot(x_df, aes(x, x.dens)) +
  geom_point(shape = 1) +
  labs(y = 'Density, p(x)') +
  theme_bw()
```

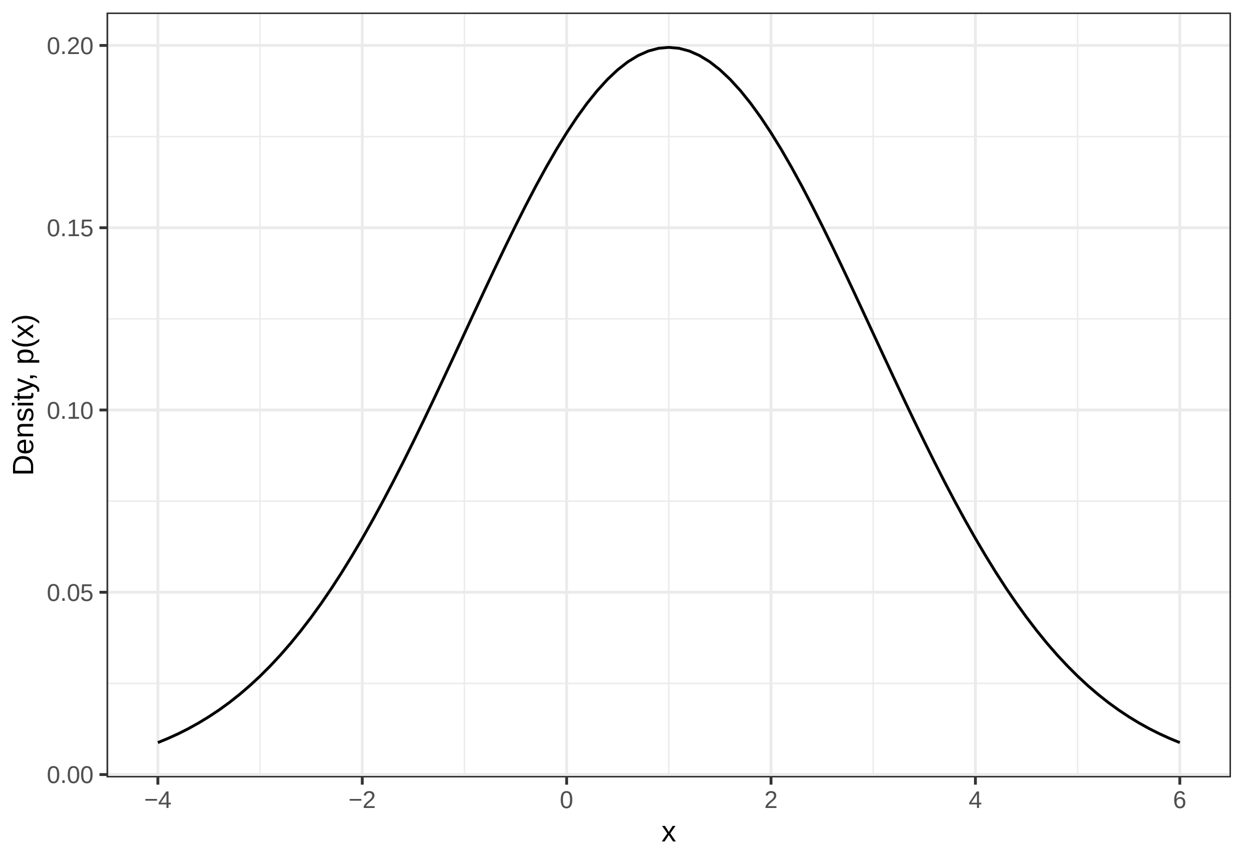


We'd really rather plot a curve, than these points. We can do this by setting `type="l"`; this makes `plot` act like the function `lines`, with the difference that it doesn't have to add to an existing plot.

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

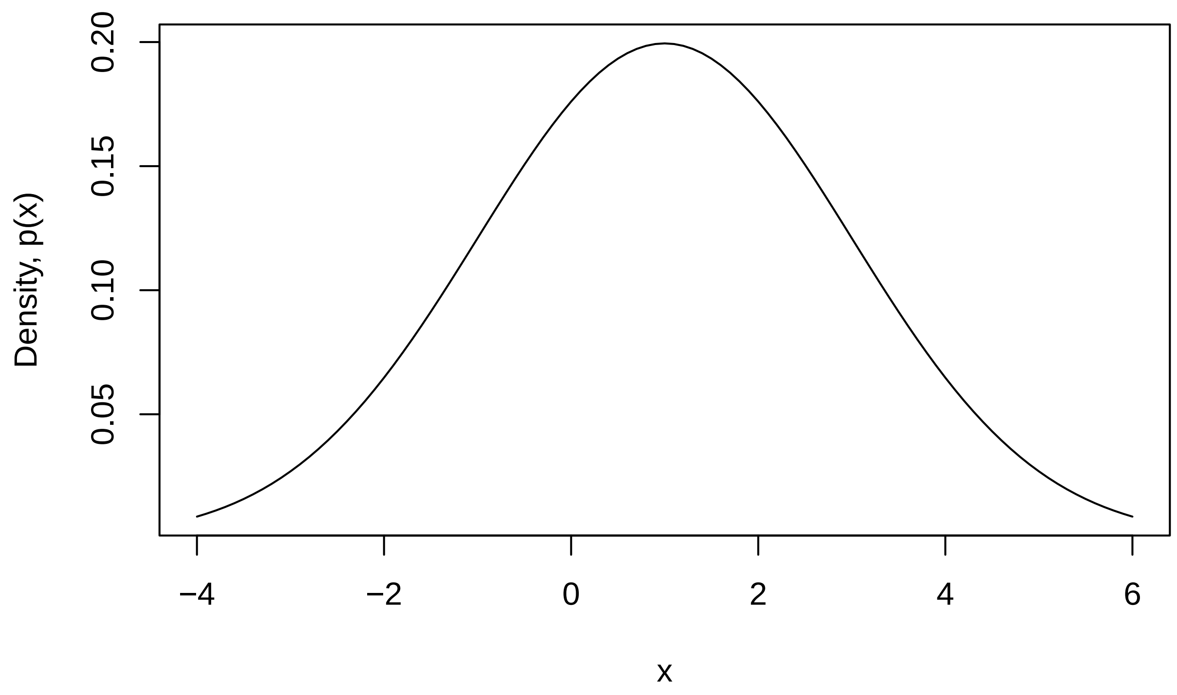


```
ggplot(x_df, aes(x, x.dens)) +
  geom_line() +
  labs(y = 'Density, p(x)') +
  theme_bw()
```



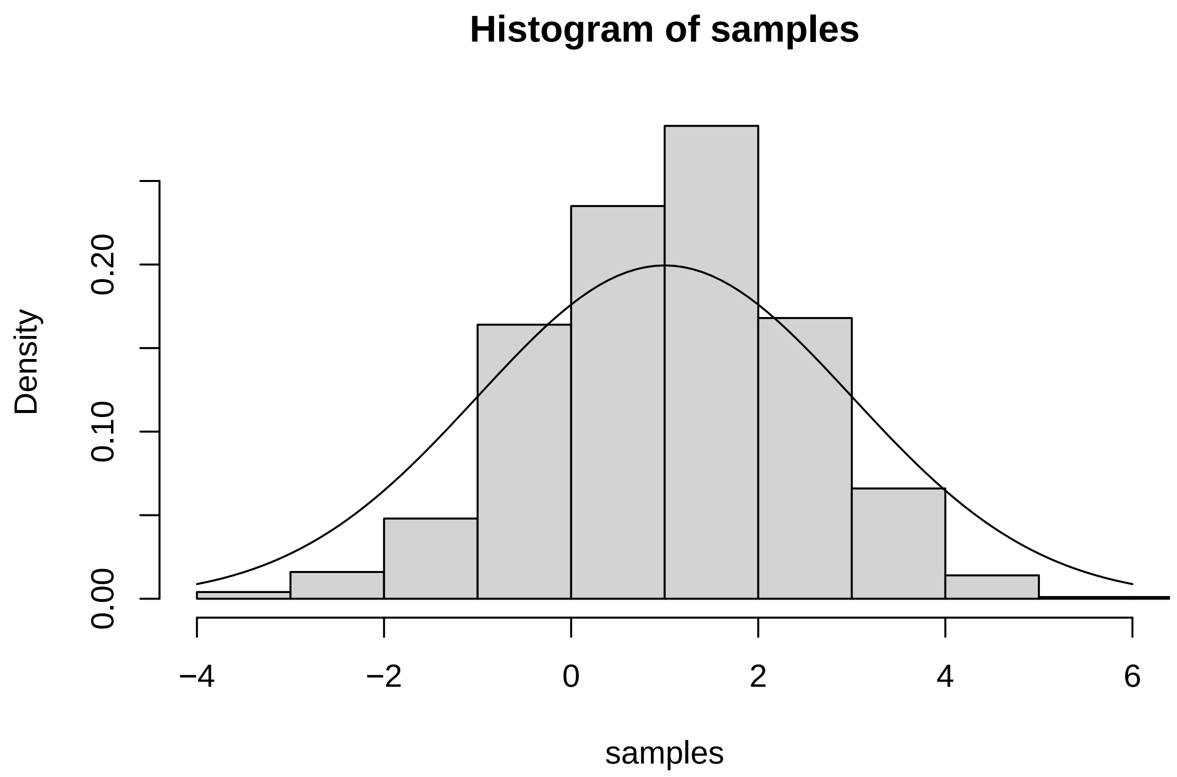
It's useful to know how to manually plot function as described above, but for simple functions `curve` can be clearer

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



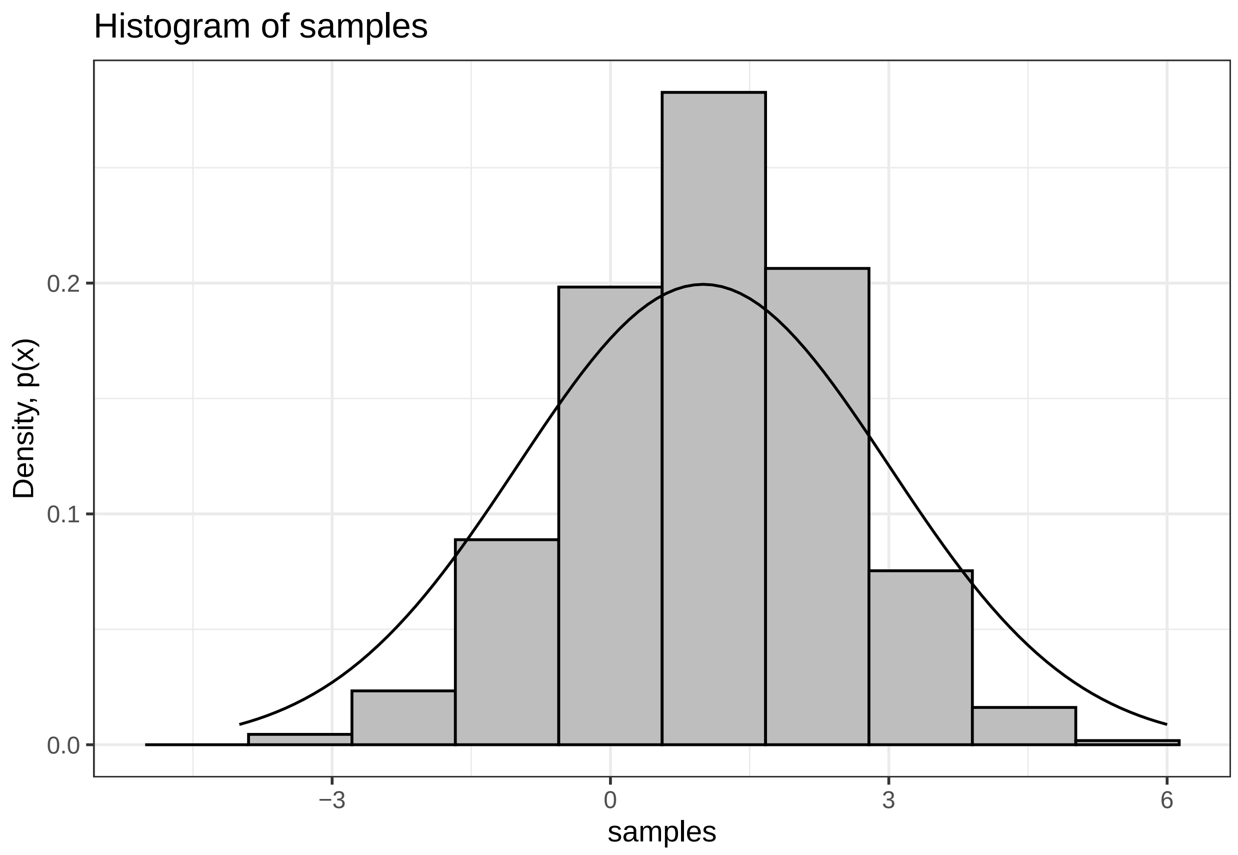
We can also overlay the density on top of the histogram, but we need to make sure the histogram is on the density scale:

```
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()
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```


Histogram of samples
Density, p(x)
0.2
0.1
0.0
−3
0
3
6
samples

Exercise 1.

(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
```

Replicate

`replicate` function in R can be used to replicate a function a certain number of times. The replicate function takes two arguments – the number of replications (B), and the function to replicate.

We can use the replicate function along with the sample function to simulate rolls of a dice.

```
dice <- c(1,2,3,4,5,6)  ## Setting up the die

sample(dice,1) ### A single roll of the draw
```

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

```
B = 10 ##Number of rolls

replicate(B, sample(dice,1))  ## Getting the outcomes for 10 rolls of a die
```

```
##  [1] 1 2 1 2 5 5 6 5 1 4
```

Exercise

Find the mean and sd of house prices in the craigslist dataset

```
#insert code here
```

Draw a sample of size 50 from craigslist price column and compute the mean of the sample

```
#insert code here
```

Repeat the process of drawing a sample of size 50 and calculating it's mean 100 times using `replicate` function

```
#insert code here

#Mean100Draws =
```

Draw a density histogram of 100 replicated draws

```
#insert code here
```

Add a normal curve to the density histogram (using the mean and sd from part a)

Summarize dataset by groups

In this dataset, we are more interested in the summaries of rent price by cities. The `tapply` function is useful when we need to break a vector into groups (subvectors), and apply a function (for example, the `mean` function) within each group. The usage is:

`tapply(Variable_of_interest, Factor_vector_representing_groups, Function_you_want_to_apply_on)`

For example, to obtain the median rent price by cities.

```
tapply(craigslist$price, craigslist$location, median)
```

```
##            alameda albany / el cerrito            berkeley         emeryville
##             2497.5              2300.0              2885.0             3095.0
##         menlo park       mountain view             oakland          palo alto
##             3600.0              2862.5              2495.0             3395.0
##       redwood city            richmond           sunnyvale
##             3152.0              2850.0              2625.0
```

You can write and apply your own functions. For example, to get the percentage of rent price less than $2000/month by city.

```
tapply(craigslist$price, craigslist$location, function(x){mean(x < 2000)})
```

```
##            alameda albany / el cerrito            berkeley         emeryville
##         0.16500000          0.21621622          0.10101010         0.03809524
##         menlo park       mountain view             oakland          palo alto
##         0.01285347          0.08555133          0.27809308         0.03802817
##       redwood city            richmond           sunnyvale
##         0.06333973          0.18625277          0.08795812
```

The rent price in Berkeley is much better than Palo Alto! The median monthly rent is much lower. And the percentage of rent price less than $2000 per month is much higher. But do not rush to conclusions, let us break down the dataset further.

Exercise 3

Use `tapply` to get following statistics for each city.

(a) The percentage of listings that are one bedroom;

```
# insert code here save the precentage of one bedrooms by cites as
# 'pct.1b'
```

(b) the median price of one bedroom listings. (Use the subset `one.bedrooms` created above)

```
# insert code here save the median of one bedrooms by cites as
# 'med.ib'
```

There are more one-bedroom rent postings in Berkeley. The median prices of one-bedrooms are less different for Berkeley and Palo Alto. The fact that the overall median price differs may be caused by the large proportion of small apartment postings in Berkeley. How you obtain the sample may greatly influence your results. Lets look at a stratified sampling dataset, where houses/apartments with 1, 2, 3 bedrooms account for 40%, 40%, 20% of the total samples of each city.

```r
prop = c(0.4, 0.4, 0.2)
samples = c()
for (city in unique(craigslist$location)){
  for (b in 1:3){
    samples = c(samples, sample(which(craigslist$brs == b & craigslist$location == city), prop[b]*60))
  }
}
craigslist.srs <- craigslist[samples, ]
```

Now we look at the median rent price by cities for the stratified sampling dataset.

```r
tapply(craigslist.srs$price, craigslist.srs$location, median)
```

```
##             alameda albany / el cerrito            berkeley          emeryville
##              2527.0              2300.0              2937.5              3095.0
##          menlo park       mountain view             oakland           palo alto
##              3435.0              2967.5              2689.0              3200.0
##        redwood city            richmond           sunnyvale
##              3037.5              2750.0              2582.0
```

Below is the percentage of rent price less than $2000/month by cities for the stratified sampling dataset.

```r
tapply(craigslist.srs$price, craigslist.srs$location, function(x){mean(x < 2000)})
```

```
##             alameda albany / el cerrito            berkeley          emeryville
##          0.18333333          0.28333333          0.10000000          0.03333333
##          menlo park       mountain view             oakland           palo alto
##          0.03333333          0.11666667          0.20000000          0.00000000
##        redwood city            richmond           sunnyvale
##          0.08333333          0.16666667          0.11666667
```

Now the difference of price median between Berkeley and Palo Alto is reduced compared to the SRS sample. This is a case where simple random samples may be misleading. The results from stratified samples may well depend on how you assign the proportions to each stratum. Care must be taken before you reach conclusions.