- The function `system.time()` will do this for us.

  - R can directly save compressed `.gz` files with the `file=gzfile("FileName.gz")` option.
    - Using a `csv.gz` file could be faster or slower (depending on your storage subsystem speeds) but could save a lot of storage space.

- Let's save the data frame to `temp2.csv.gz`
    - The new file should be much smaller.
    - Let's read the `.gz` file to see if this worked.
      - R will automatically recognize the `.gz` extensions and decompress when loading.

Using gzipped flat files is a great and simple way to transfer, load, and work with moderately large data sets.

***

- If speed or memory are an issue for your project when using R built-in I/O functions, you might need an external package.
- For example, there are many packages that can speed up the reading and writing speeds in R.
  - The `vroom` package comes with functions called vroom and `vroom_write` which are very efficient and speedy.

  - This function works great for reading and writing data from flat files (delimited text files).
      - `vroom` automatically uses the full abilities of your CPU, namely executes in multi-threaded (using the several processer cores that modern CPUs have).

- Let's install the vroom package and time writing the same data frame, let's call the file temp3.

  - Note that vroom is using tab delimiter, so you need to specify `delim=","`.

  - If some R package is already installed, you could actually access a package function with `::` without loading the full package with `library()` or `require()`.

```
system.time(vroom::vroom_write(a,"temp3.csv",d
elim=","))
```

***

- Importing *other data types* directly (MS Excel, SAS, SPSS, STATA, Minitab, etc.) or importing data directly from *relational databases* is also available.
- Besides *vroom*, there are also dedicated packages to handle importing of flat file data.
  - For instance: *readr* and *data.table*.

- If you want to study any of the above in more details, please refer to:

https://www.datacamp.com/courses/importing-data-in-r-part-1

https://www.datacamp.com/courses/importing-data-in-r-part-2

- A seemingly great free DataCamp tutorial gives details for reading different types of data is also available.
  - *"... comprehensive, yet easy tutorial to quickly import data into R, going from simple text files to the more advanced SPSS and SAS files."*

https://www.datacamp.com/community/tutorials/r-data-import-tutorial

- Yet again, there is **rarely a real need to go fancy**.
  - Just use a flat file format like `.csv` and move ahead with your project.
  - In my experience in almost all kinds of settings it is almost always better to use the KISS principle (credit: U.S. Navy in the mid-last century).

***

- What about importing a real data set from the web?
- The **Stock Exchange of Hong Kong** is one of the fastest growing stock exchanges in Asia. It has 2,500+ listed companies with a combined market capitalization of more than 5 trillion USD.
  - Let us consider an actual data set that is produced daily by this stock exchange.
  - Most files are flat tables.
    - However, you often need to study the file before you are able to actually parse it in R (or any other language) ...

- The website is https://www.hkex.com.hk/
  - Or just google HK exchange.
- To save time we will download

https://www.hkex.com.hk/eng/dwrc/search/dwFullList.csv
  - Let's try directly downloading this `csv` file, supposedly comma separated values file.
  - What now?
    - You need to look at the file or read the documentation…
    - Small files like this, you can just download and investigate.
    - Now let's resolve the issue…

19. Simulations and Central Limit Theorem

- One of the true powers of R is to run simulations, **numeric experiments with random outcomes**, by sampling from:
  - the built-in R distributions
  - other distributions in external libraries/packages
  - or from the data directly.

- Recall, random sample from some distribution is done with `r...` where ... is replaced with the distribution name
  - For instance, `rnorm(30)` will generate 30 numbers from the standard normal distribution.

- If you want to sample from data, one of the main tools is the **`sample()`** function.

```
# sample(x,k) generates a random permutation
of k unique elements from the vector x, no
repeated elements are allowed by default

# run each several times
sample (1:10) # permute all 10 elements
sample (1:10,4) # 4 elements out of 10
sample(1:5)  # permutation 5P5
sample(1:5,6) # will give an error since
              # no repetitions are allowed
```

- You might want to allow repeats.

- Allowing repetitions is like randomly selecting an element and “returning it back” in the set before the next selection (*sampling with replacement*).

```
sample(x,6,replace=T) # this works just fine
```

- Let us simulate rolling a fair die 6,000,000 times.
  - Let’s first “simulate” manually with the participants in our class session.
    - Randomly pick one integer from 1 to 6 and write it down.
  - Bar-graph (with the histogram or the barplot R function) is the default choice for a graphical representation here for non-continuous data.

- Now use R to simulate the same number of die rolls; store the results in `rolls`.

- We expect ~1/6 of the outcomes to be 4s. Let’s compute how many we’ve got.

- How about 6,000,000 rolls in `rolls`?

- Let’s seed the random generator with set.seed (2024) and compute how many of the 6,000,000 rolls are 4s.

- `rolls` vector is only about ~0.02Gb in the RAM.
  - Piece of cake for machines nowadays.

***

- Let's estimate the **probability of getting <u>at least one 6</u> in five consecutive rolls of a fair die**.
  - The exact (theoretical) probability is easy to derive…

  - We will use a quick simulation to "check" our theoretical result (computing experimental probability).
    - We should be very close to the theoretical probability if we have enough simulated die rolls.

Probability of getting <u>at least one 6</u> in five consecutive die rolls…

- We will use the matrix function to store the results (although a loop can be used too).

- Each column in the matrix will be a "trial", i.e., 5 die rolls.

```
# Always start modest when building code
trls <- 10  # only 10 die rolls
M <- matrix(sample(1:6,5*trls, replace=T),
ncol=trls)
# let's build our code one step a time ...
# ...
# Next, let's try 1,000,000 rolls
trls <- 10^6  # number of trials


# The theoretical probability was...
```


- Sometimes theoretical results are not feasible…

- Obtaining a theoretical **confidence interval for the ratio of two proportions** is such an example.
  - The distribution of a ratio of random variables does not generally follow a simple known distribution.
  - In practice, we often use simulations to estimate the confidence interval for the ratio of two proportions, as exact theoretical results are difficult to obtain.
  - These methods are based on resampling techniques and can provide good approximations when the simulation size is large enough.

- Let's consider an example like this:

Suppose that in a survey of 250 males and 250 females, you observe that 88% of the females have health insurance, but only 80% of the males have it.

Computing a confidence interval for the ratio of these two proportions could be challenging but is needed as the data is just a sample.

Let us simulate to estimate the confidence interval …

- Let us create a code to practice and to also illustrate the Central Limit Theorem
  - Start from real data and create a simulated population following the instructions closely.
    - We will do more than simply resampling…

```
# Analyzing Rental Data

# This assignment will have you analyze rental price data
# and take samples to compare sample means.

# The data gives 45 one-bedroom apartment rental prices
# listed on 1/18/2017 at rent.com and craigslist.
```



```
# Data source:
data1 <- c(775, 900, 775, 820, 810, 1150, 790, 610,
875, 775, 890, 600, 760, 625, 710, 960, 500, 690,
745, 550, 685, 625, 775, 580, 590, 495, 595, 550,
495, 650, 580, 700, 1000, 900, 605, 475, 875, 850,
650, 785, 600, 825, 490, 500, 795)

# Instructions:

# 1. Generate a histogram of the 45 actual rental
prices

# 2. Create a total "population" of size 1000
Generate additional rental prices by:
```



```
#   - Setting a seed for reproducibility:
set.seed(12347)

#   - Sampling from data1 with replacement
#   - Adding random noise by sampling from normal
#     distribution centered around each sampled
#     value with SD=20. Each random number will
#     have a different mean, randomly picked from
#     the data.
#   - Rounding the simulated values to nearest 10
#   - Combing simulated and original rents in a
#     single vector aptRentPop


# 3. Calculate summary statistics of the population
```



```
# 4. Take 2 random samples of size 30 from the
"population" in aptRentPop.

# 5. Calculate and compare the means of the two
samples.

# 6. Take 1000 samples of size 30 from the
"population" and store the 1000 sample means.

# 7. Calculate mean and standard deviation of the
1000 sample means.

# 8. Generate a histogram of the 1000 sample means.
```

```
# 9. Repeat the 1000 sample means to be for samples
#    of size 9 this time.

# Note that the original distribution was not close
# to the normal distribution(not bell shaped).
# The distribution of the sample means however is
# approximately normal

# This is the Central Limit Theorem in action ...
```


Central Limit Theorem (CLT)

The distribution of the sample mean is a probability distribution of all possible sample means for a given sample size.

- **CLT**: for a "large enough" sample size the **distribution of the sample mean** is approximately normal regardless of the distribution of the population we sample from.
- In our example, regardless of the shape of our "population" (`aptRentPop`), the distribution of the sample means tends toward normal distribution as the sample size increases.

- **CLT**: For **any population**, the distribution of sample means is **approximately normal** for large enough sample size.
  - For large enough sample size the sample means have a distribution that can be approximated by a normal distribution with the population mean μ and population standard deviation $\frac{\sigma}{\sqrt{n}}$.
  - This assumes simple random sampling.
  - The concept of 'large enough' (often cited as a sample size greater than 30) varies depending on the distribution of the population.
  - Different sample sizes are needed to achieve approximate normality distribution of the sample mean depending on the original population distribution.