DSCI351-351m-451: Class 14a p3 Illustrating CLT in R and Python

Code ▾

2108-351-351m-451-w14a-p3-Illustrate-Central-Limit-Theorem-in-RandPython

Roger H. French, Sameera Nalin Venkat, Raymond Weiser

06 December, 2022

Two high level, interpretted languages for Data Science

- R and [Python](https://en.wikipedia.org/wiki/Python_(programming_language) (https://en.wikipedia.org/wiki/Python_(programming_language))
  - are high level scripting languages
  - They are "interface" languages, that serve
    - To provide a comfortable environment for people
  - While connecting to efficient code libraries in other languages

We can use both R and Python3 code blocks in Rstudio

- Just change the beginning from
  - ```{r}
  - ```{python}

Illustrating the Central Limit Theorem in R and Python3

- This is the first article in our series "Lost in Translation between R and Python".
  - The aim to provide high-quality R and Python 3 code
    - to achieve some non-trivial tasks.
  - If you want to learn Python, and you know R,
    - then the Python code will be new to you.
    - but it does the same as the R code we'll do first

Let's start with a little bit of statistics

- Illustrating the Central Limit Theorem (CLT).

Take a sample of a random variable X with finite variance.

The CLT says: No matter how "unnormally" distributed X is,

- its sample mean $\bar{x}$
  - will be approximately normally distributed,
  - at least if the sample size is not too small.

This classic result is the basis

- to construct simple confidence intervals
- and hypothesis tests for the (true) mean of X,
  - i.e. μ
- check out Wikipedia for information on the Central Limit Theorem (https://en.wikipedia.org/wiki/Central_limit_theorem).

R code: So lets do a simulation of the CLT and LLN

The code below illustrates this famous statistical result by simulation,

- using a very asymmetrically distributed X,

- namely $X = 1$
    - with probability 0.2
- and $X = 0$ otherwise.

X could represent the result of

- asking a randomly picked person whether he smokes.
- Conducting such a poll,
    - the mean of the collected sample of such results
- would be a statistical estimate of
    - the proportion of people smoking.

Curiously, by a tiny modification,

- the same code will also illustrate another key result in statistics
    - the Law of Large Numbers (https://en.wikipedia.org/wiki/Law_of_large_numbers):
- For growing sample size,
    - the distribution of the sample mean of X
    - contracts to the expectation $E(X)$.

Hide

```
# Fix seed, set constants
set.seed(2006)
sample_sizes <- c(1, 10, 30, 1000)
nsims <- 10000
```

Lets make a helper function in R

Hide

```
# Helper function: Mean of one sample of X
one_mean <- function(n, p = c(0.8, 0.2)) {
  mean(sample(0:1, n, replace = TRUE, prob = p))
}
one_mean(10)
```

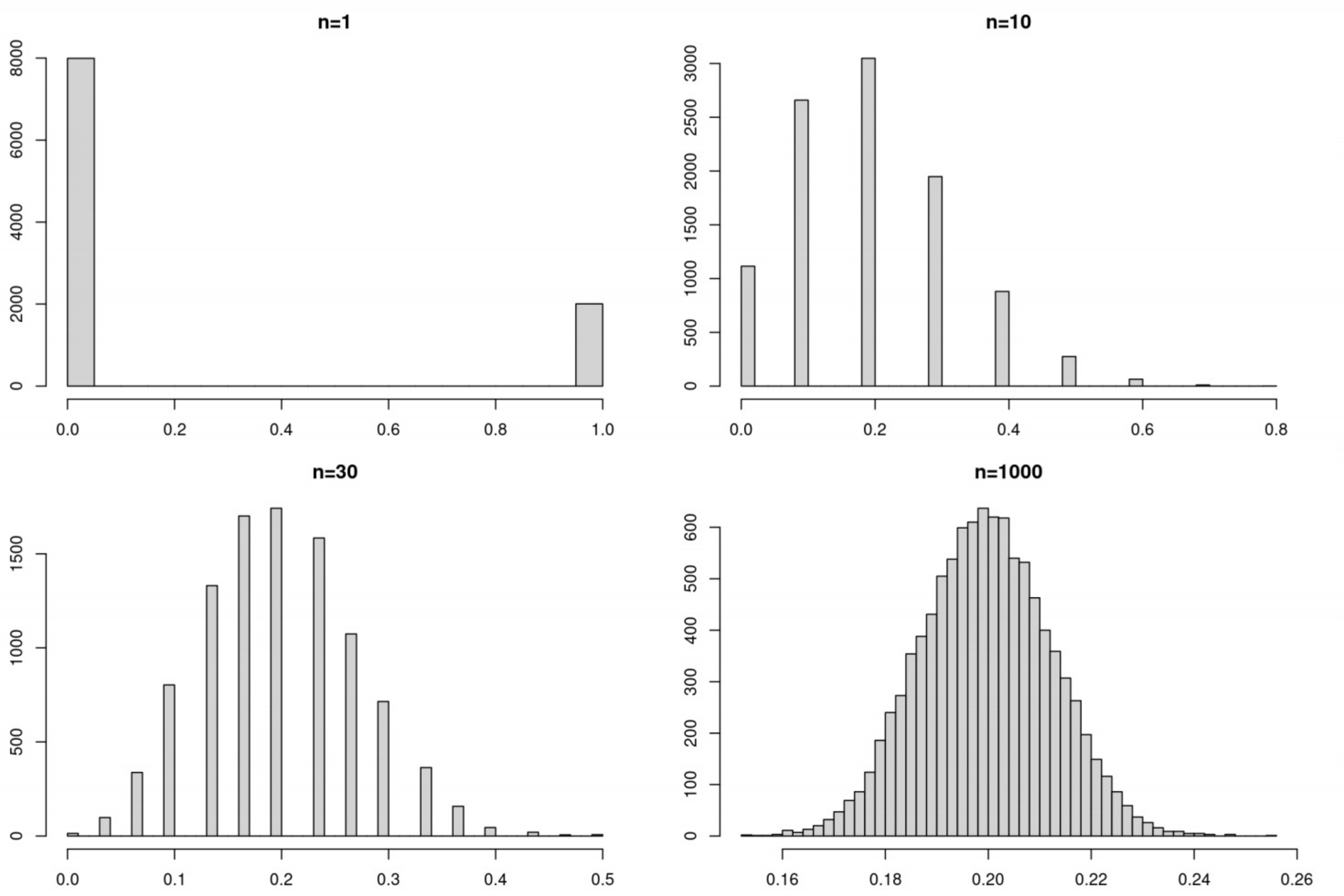
```
[1] 0.2
```

Now lets run our simulation 1000 times for the CLT in R

Hide

```
# Simulate and plot
par(mfrow = c(2, 2), mai = rep(0.4, 4))

for (n in sample_sizes) {
  means <- replicate(nsims, one_mean(n))
  hist(means, breaks = "FD",
       # xlim = 0:1, # uncomment for LLN
       main = sprintf("n=%i", n))
}
```



The CLT result is

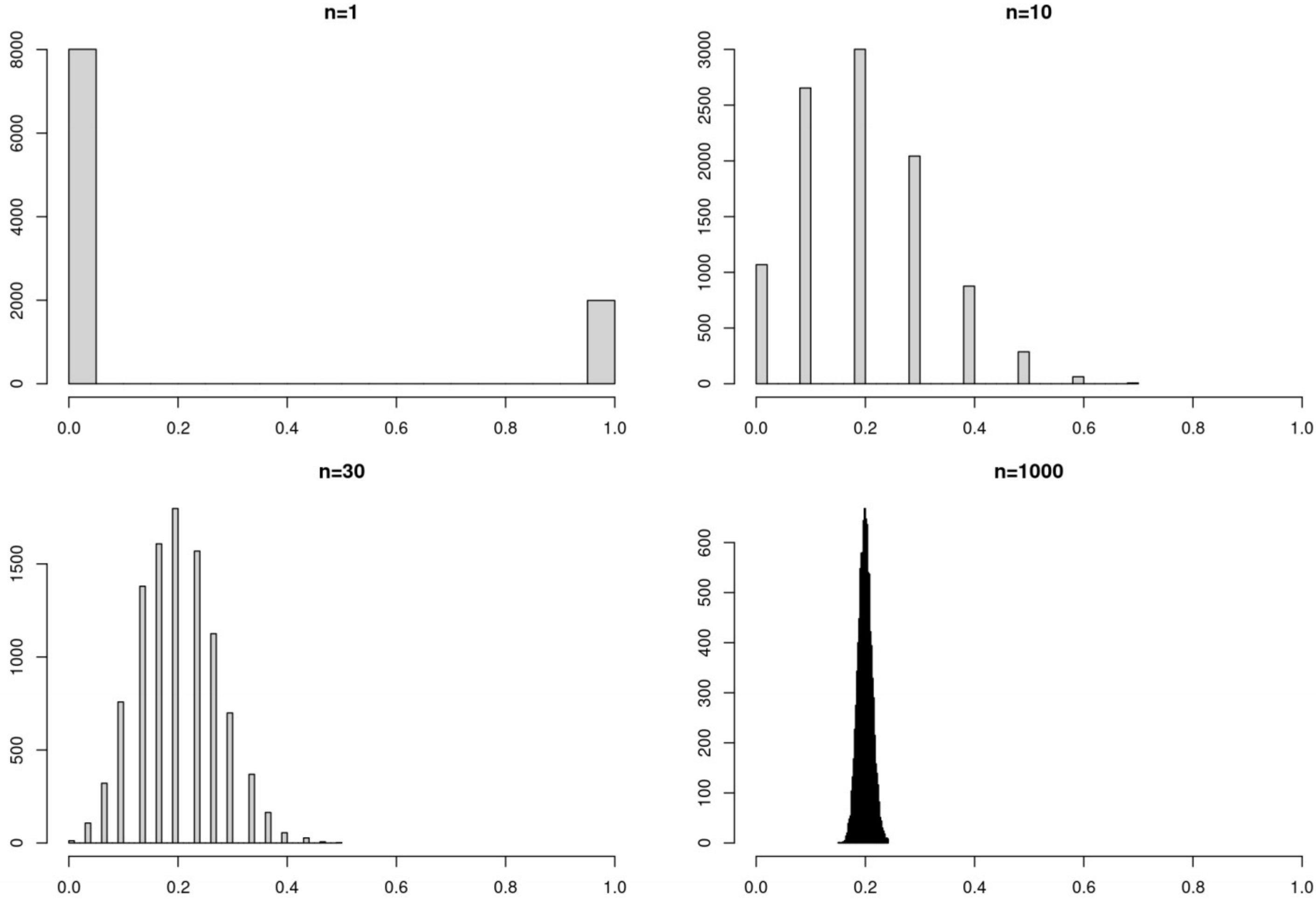
- The larger the sample size,
  - the closer the histogram of the simulated means
  - resembles a symmetric bell shaped curve.

Now lets look at the Law of Large Numbers in R

Hide

```
# Simulate and plot
par(mfrow = c(2, 2), mai = rep(0.4, 4))

for (n in sample_sizes) {
  means <- replicate(nsims, one_mean(n))
  hist(means, breaks = "FD",
       xlim = 0:1, # uncomment for LLN
       main = sprintf("n=%i", n))
}
```



And the Law of Large Numbers

Fixing the x-scale illustrates - for free(!)

- the Law of Large Numbers:
- The distribution of the mean
    - contracts more and more
    - to the expectation value of 0.2.

Python3 code: So lets do a simulation of the CLT and LLN

- First import some python packages
    - namely `numpy` (https://en.wikipedia.org/wiki/NumPy) for
        - a Python library that adds support for
        - large, multi-dimensional arrays and matrices,
        - along with a large collection of high-level mathematical functions
        - to operate on these arrays.
    - and `matplotlib` (https://en.wikipedia.org/wiki/Matplotlib) for plotting
        - is a Python plotting library
        - that works with `numpy` , the numerical mathematics package.
        - It provides an object-oriented API for embedding plots into applications
        - using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Hide

```
# First we'll 'import' some Python3 packages
  # This is analogous to 'library'ing in some R packages
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
# matplotlib$use("Agg", force = TRUE) # this makes the matplotlib plot appear in Rmarkdown
```

Now lets set seed and some constants for 1000 simulation runs

Hide

```
# Fix seed, set constants
np.random.seed(100)
sample_sizes = [1, 10, 30, 1000]
nsims = 10_000
```

Lets make our helper function to calculate the mean of one sample

Hide

```
# Helper function: Mean of one sample
def one_mean(n, p=0.2):
    return np.random.binomial(1, p, n).mean()

one_mean(10)
```

```
0.2
```

Now lets run our simulation 1000 times for the CLT in Python3

Hide

```
# Simulate and plot
fig, axes = plt.subplots(2, 2, figsize=(8, 8))

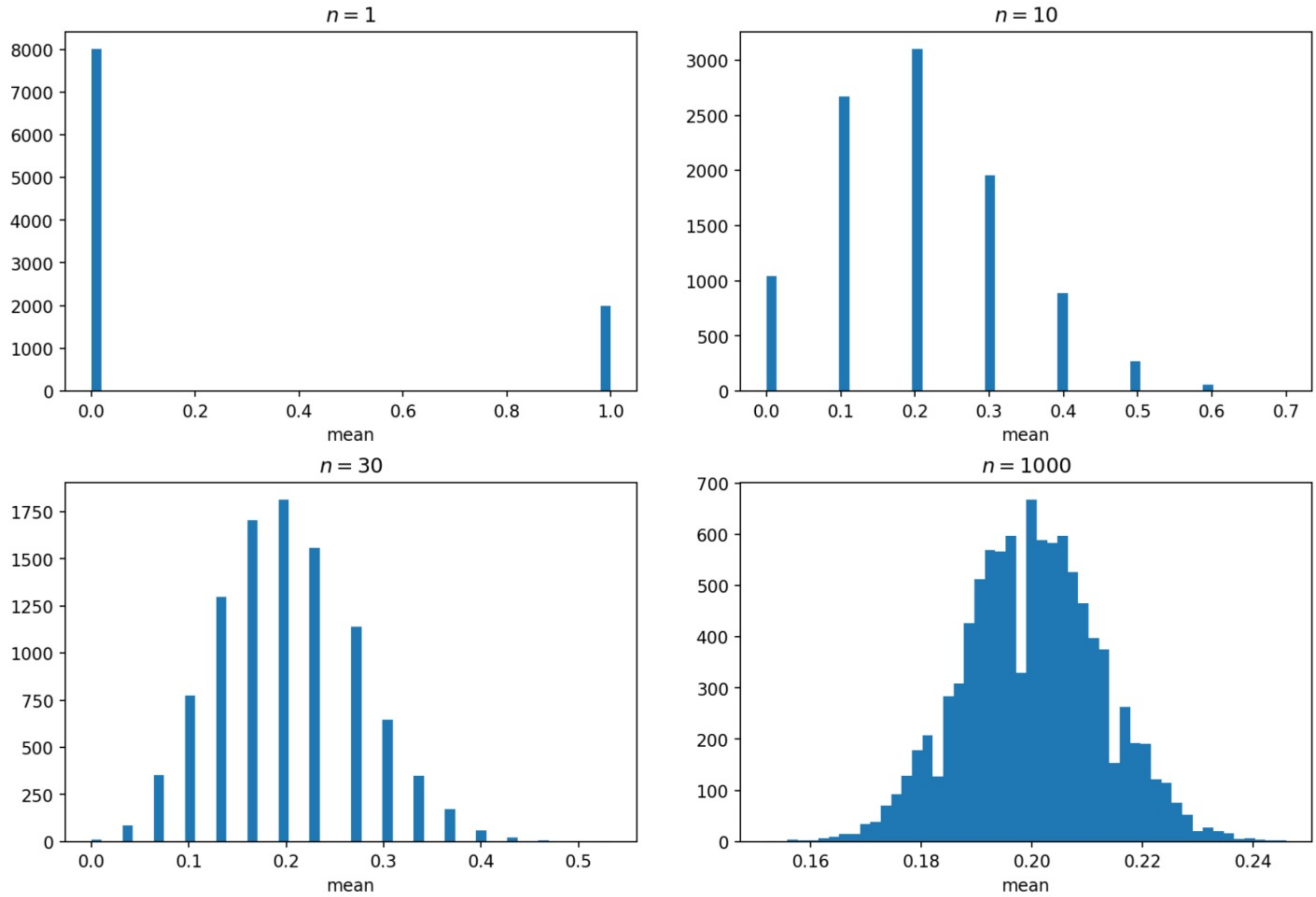
for i, n in enumerate(sample_sizes):
    means = [one_mean(n) for ell in range(nsims)]
    ax = axes[i // 2, i % 2]
    ax.hist(means, 50)
    ax.title.set_text(f'$n = {n}$')
    ax.set_xlabel('mean')
    # ax.set_xlim(0, 1)  # uncomment for LLN
fig.tight_layout()
```

```
(array([8007.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0., 1993.]), array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.1
2, 0.14, 0.16, 0.18, 0.2 ,
       0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
       0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
       0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
       0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(array([1.045e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
       0.000e+00, 2.673e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
       0.000e+00, 0.000e+00, 3.101e+03, 0.000e+00, 0.000e+00, 0.000e+00,
       0.000e+00, 0.000e+00, 0.000e+00, 1.958e+03, 0.000e+00, 0.000e+00,
       0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 8.890e+02, 0.000e+00,
       0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 2.710e+02,
       0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
       6.000e+01, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
       0.000e+00, 3.000e+00]), array([0.   , 0.014, 0.028, 0.042, 0.056, 0.07 , 0.084, 0.09
8, 0.112,
       0.126, 0.14 , 0.154, 0.168, 0.182, 0.196, 0.21 , 0.224, 0.238,
       0.252, 0.266, 0.28 , 0.294, 0.308, 0.322, 0.336, 0.35 , 0.364,
       0.378, 0.392, 0.406, 0.42 , 0.434, 0.448, 0.462, 0.476, 0.49 ,
       0.504, 0.518, 0.532, 0.546, 0.56 , 0.574, 0.588, 0.602, 0.616,
       0.63 , 0.644, 0.658, 0.672, 0.686, 0.7  ]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(array([9.000e+00, 0.000e+00, 0.000e+00, 8.700e+01, 0.000e+00, 0.000e+00,
       3.520e+02, 0.000e+00, 0.000e+00, 7.760e+02, 0.000e+00, 0.000e+00,
       1.297e+03, 0.000e+00, 0.000e+00, 1.704e+03, 0.000e+00, 0.000e+00,
       1.814e+03, 0.000e+00, 0.000e+00, 1.558e+03, 0.000e+00, 0.000e+00,
       0.000e+00, 1.140e+03, 0.000e+00, 0.000e+00, 6.470e+02, 0.000e+00,
       0.000e+00, 3.490e+02, 0.000e+00, 0.000e+00, 1.720e+02, 0.000e+00,
       0.000e+00, 6.100e+01, 0.000e+00, 0.000e+00, 2.100e+01, 0.000e+00,
       0.000e+00, 8.000e+00, 0.000e+00, 0.000e+00, 4.000e+00, 0.000e+00,
       0.000e+00, 1.000e+00]), array([0.        , 0.01066667, 0.02133333, 0.032     , 0.042
66667,
       0.05333333, 0.064     , 0.07466667, 0.08533333, 0.096     ,
       0.10666667, 0.11733333, 0.128     , 0.13866667, 0.14933333,
       0.16      , 0.17066667, 0.18133333, 0.192     , 0.20266667,
       0.21333333, 0.224     , 0.23466667, 0.24533333, 0.256     ,
       0.26666667, 0.27733333, 0.288     , 0.29866667, 0.30933333,
       0.32      , 0.33066667, 0.34133333, 0.352     , 0.36266667,
       0.37333333, 0.384     , 0.39466667, 0.40533333, 0.416     ,
       0.42666667, 0.43733333, 0.448     , 0.45866667, 0.46933333,
       0.48      , 0.49066667, 0.50133333, 0.512     , 0.52266667,
       0.53333333]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(array([  1.,   0.,   4.,   2.,   3.,   6.,  10.,  15.,  15.,  35.,  39.,
        70.,  92., 129., 178., 208., 127., 284., 309., 427., 513., 570.,
       567., 597., 330., 668., 589., 583., 597., 526., 465., 398., 376.,
       153., 263., 192., 191., 121., 115.,  76.,  52.,  20.,  27.,  20.,
        16.,   5.,   7.,   4.,   2.,   3.]), array([0.152  , 0.15388, 0.15576, 0.15764, 0.1
5952, 0.1614 , 0.16328,
       0.16516, 0.16704, 0.16892, 0.1708 , 0.17268, 0.17456, 0.17644,
       0.17832, 0.1802 , 0.18208, 0.18396, 0.18584, 0.18772, 0.1896 ,
       0.19148, 0.19336, 0.19524, 0.19712, 0.199  , 0.20088, 0.20276,
       0.20464, 0.20652, 0.2084 , 0.21028, 0.21216, 0.21404, 0.21592,
       0.2178 , 0.21968, 0.22156, 0.22344, 0.22532, 0.2272 , 0.22908,
```

```
       0.23096, 0.23284, 0.23472, 0.2366 , 0.23848, 0.24036, 0.24224,
       0.24412, 0.246  ]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
```

Hide

```
plt.show()
```



The CLT result is

- The larger the sample size,
    - the closer the histogram of the simulated means
    - resembles a symmetric bell shaped curve.

Now lets look at the Law of Large Numbers in Python3

Hide

```
# Simulate and plot
fig, axes = plt.subplots(2, 2, figsize=(8, 8))

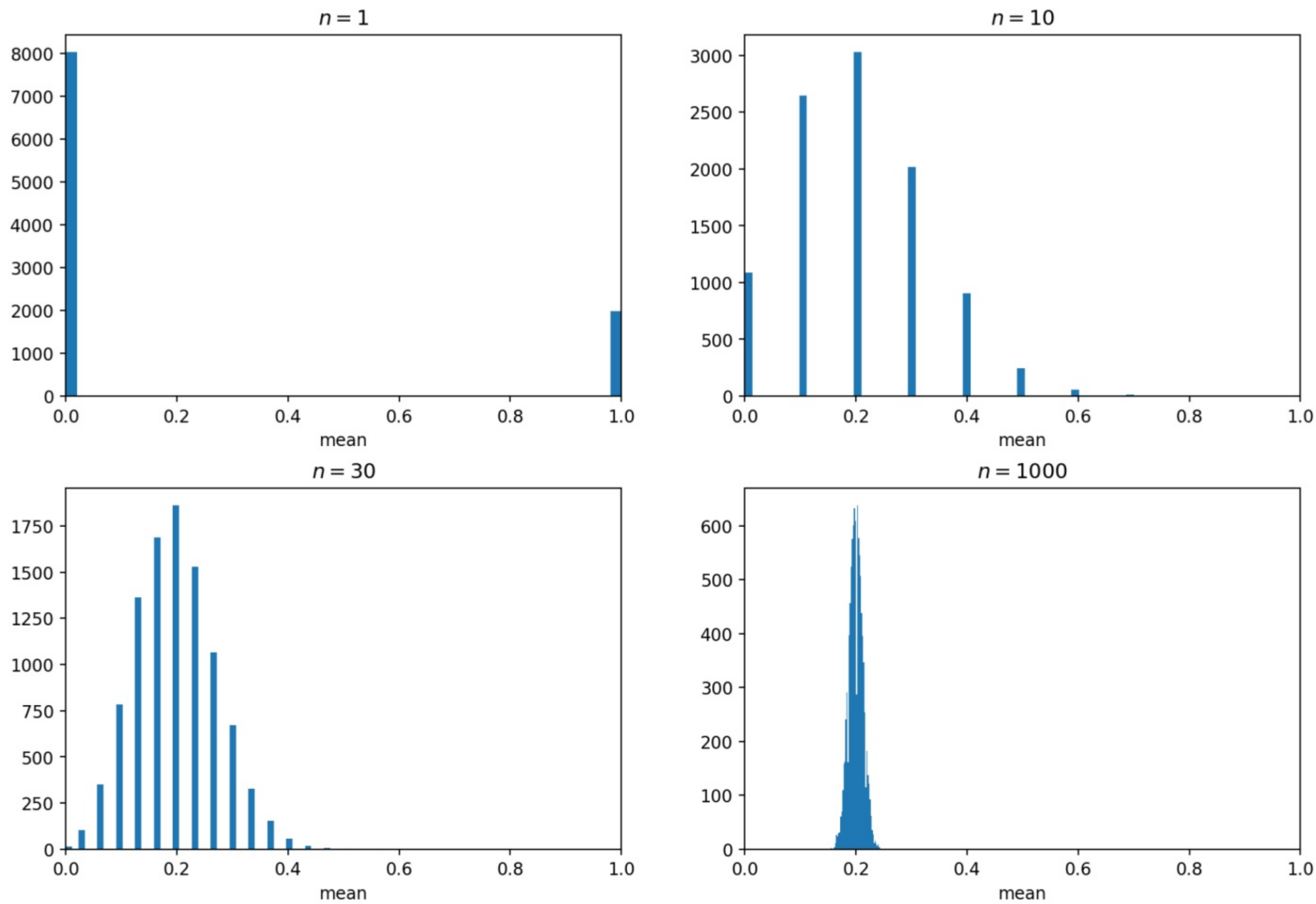
for i, n in enumerate(sample_sizes):
    means = [one_mean(n) for ell in range(nsims)]
    ax = axes[i // 2, i % 2]
    ax.hist(means, 50)
    ax.title.set_text(f'$n = {n}$')
    ax.set_xlabel('mean')
    ax.set_xlim(0, 1)  # uncomment for LLN
fig.tight_layout()
```

```
(array([8026.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0., 1974.]), array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.1
2, 0.14, 0.16, 0.18, 0.2 ,
       0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
       0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
       0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
       0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(0, 1)
(array([1090.,    0.,    0.,    0.,    0.,    0.,    0., 2643.,    0.,
          0.,    0.,    0.,    0.,    0., 3028.,    0.,    0.,    0.,
          0.,    0.,    0., 2017.,    0.,    0.,    0.,    0.,    0.,
          0.,  906.,    0.,    0.,    0.,    0.,    0.,    0.,  245.,
          0.,    0.,    0.,    0.,    0.,    0.,   56.,    0.,    0.,
          0.,    0.,    0.,    0.,   15.]), array([0.   , 0.014, 0.028, 0.042, 0.056, 0.07
, 0.084, 0.098, 0.112,
       0.126, 0.14 , 0.154, 0.168, 0.182, 0.196, 0.21 , 0.224, 0.238,
       0.252, 0.266, 0.28 , 0.294, 0.308, 0.322, 0.336, 0.35 , 0.364,
       0.378, 0.392, 0.406, 0.42 , 0.434, 0.448, 0.462, 0.476, 0.49 ,
       0.504, 0.518, 0.532, 0.546, 0.56 , 0.574, 0.588, 0.602, 0.616,
       0.63 , 0.644, 0.658, 0.672, 0.686, 0.7  ]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(0, 1)
(array([  14.,    0.,  102.,    0.,    0.,  351.,    0.,    0.,  782.,
          0.,    0., 1363.,    0.,    0., 1689.,    0.,    0., 1862.,
          0.,    0., 1529.,    0.,    0., 1065.,    0.,    0.,  673.,
          0.,    0.,  329.,    0.,    0.,  155.,    0.,    0.,   57.,
          0.,    0.,   18.,    0.,    0.,    5.,    0.,    0.,    3.,
          0.,    0.,    0.,    0.,    3.]), array([0.        , 0.01133333, 0.02266667, 0.03
4     , 0.04533333,
       0.05666667, 0.068     , 0.07933333, 0.09066667, 0.102     ,
       0.11333333, 0.12466667, 0.136     , 0.14733333, 0.15866667,
       0.17      , 0.18133333, 0.19266667, 0.204     , 0.21533333,
       0.22666667, 0.238     , 0.24933333, 0.26066667, 0.272     ,
       0.28333333, 0.29466667, 0.306     , 0.31733333, 0.32866667,
       0.34      , 0.35133333, 0.36266667, 0.374     , 0.38533333,
       0.39666667, 0.408     , 0.41933333, 0.43066667, 0.442     ,
       0.45333333, 0.46466667, 0.476     , 0.48733333, 0.49866667,
       0.51      , 0.52133333, 0.53266667, 0.544     , 0.55533333,
       0.56666667]), <a list of 50 Patch objects>)
Text(0.5, 0, 'mean')
(0, 1)
(array([  2.,   1.,   1.,   3.,  12.,  26.,  23.,  30.,  32.,  61.,  70.,
       110., 158., 163., 240., 291., 161., 397., 457., 524., 576., 601.,
       633., 609., 287., 638., 577., 545., 507., 438., 395., 346., 254.,
       115., 183., 137., 121.,  93.,  62.,  35.,  28.,  11.,  16.,  10.,
         6.,   9.,   4.,   1.,   0.,   1.]), array([0.155  , 0.15688, 0.15876, 0.16064, 0.1
6252, 0.1644 , 0.16628,
       0.16816, 0.17004, 0.17192, 0.1738 , 0.17568, 0.17756, 0.17944,
       0.18132, 0.1832 , 0.18508, 0.18696, 0.18884, 0.19072, 0.1926 ,
       0.19448, 0.19636, 0.19824, 0.20012, 0.202  , 0.20388, 0.20576,
       0.20764, 0.20952, 0.2114 , 0.21328, 0.21516, 0.21704, 0.21892,
       0.2208 , 0.22268, 0.22456, 0.22644, 0.22832, 0.2302 , 0.23208,
       0.23396, 0.23584, 0.23772, 0.2396 , 0.24148, 0.24336, 0.24524,
       0.24712, 0.249  ]), <a list of 50 Patch objects>)
```

```
Text(0.5, 0, 'mean')
(0, 1)
```

Hide

```
plt.show()
```



And the Law of Large Numbers

Fixing the x-scale illustrates - for free(!)

- the Law of Large Numbers:
- The distribution of the mean
    - contracts more and more
    - to the expectation value of 0.2.

Links

- Michael Mayer, Lost in Translation between R and Python Series, 021-01-07
    - Illustrating the Central Limit Theorem (https://lorentzen.ch/index.php/2021/01/07/illustrating-the-central-limit-theorem/)