week09–statistics

March 22, 2024

1 George McNinch Math 87 - Spring 2024

2 Week 09

2.1 # Statistics & the Central Limit Theorem

3 Why statistics?

From the point-of-view of modeling, one often needs information about average values and expected values.

To reason precisely in this setting, we need the language of statistics.

Here are some examples to keep in mind for our discussion:

- Consider a *lottery* in which 4 million tickets are sold each week, for \$1 each. Of these, one ticket wins $\$1.5 million, 500 tickets win$
  800, \$10,000 tickets win \$10. If you buy a ticket, how much should you *expect* to win??
- At a certain grocery store between the hours of 10 AM and noon, one expects a customer needs to check out every two minutes, and the average check-out time is 3 minutes. How many registers should be staffed?

4 Mean, variance, and standard deviation

Let $X = \{x_i\}_{i=1,\dots,n}$ be a data set of n values (real numbers!).

The **mean** (or average) of X is

$$\mu = \mu(X) = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

The **median** of X is the number $\overline{x}$ with the following properties:

$x_i \leq \overline{x}$ for $\frac{n}{2}$ values of $i \in \{1, 2, \dots, n\}$ and $x_j \geq \overline{x}$ for $\frac{n}{2}$ values of $j \in \{1, 2, \dots, n\}$.

Suppose we re-arrange the numbers x_i so that

$$x_1 \leq x_2 \leq \cdots \leq x_{n-1} \leq x_n;$$

if n is odd, then $\overline{x} = x_{(n+1)/2}$.

if n is even, a common convention is take $\overline{x}$ to be the average of $x_{n/2}$ and $x_{1+n/2}$.

5 Examples:

[4]:
```
import pandas as pd
import numpy as np
```

[5]:
```
d = [[1,2,3,4,5],
     [1,2,3,6]]
X = pd.DataFrame(d)
display(X)
pd.DataFrame({"mean":X.mean(axis=1),
              "median":X.median(axis=1)})
```

```
   0  1  2  3    4
0  1  2  3  4  5.0
1  1  2  3  6  NaN
```

[5]:

	mean	median
0	3.0	3.0
1	3.0	2.5

Let's generate some *random* data use the `python` module `pandas`, and compute means and medians.

As an aside, this blog post (I don't know anything about the author...!) has some - I think - useful discussion of *axes* in Python.

[6]:
```
from numpy.random import default_rng

rng = default_rng()

X = pd.DataFrame(rng.integers(0,20,size=(100,10)))
X
```

[6]:

	0	1	2	3	4	5	6	7	8	9
0	14	4	14	5	13	7	17	4	8	11
1	18	16	13	18	15	16	7	18	15	16
2	4	2	0	5	9	11	13	1	7	18
3	4	10	18	7	7	10	18	18	14	0
4	7	7	17	13	1	6	1	6	8	5
..	..	..	..	..	..	..	..	..	..	..
95	0	18	1	0	4	10	14	1	11	17
96	2	4	17	5	7	9	2	8	4	9
97	6	7	1	16	1	10	17	19	12	14
98	18	8	13	12	17	13	11	10	4	11
99	15	11	0	19	13	5	16	13	8	4

[100 rows x 10 columns]

[7]:
```
pd.DataFrame({"median":X.median(),
              "mean":X.mean()})
```

[7]:

	median	mean
0	9.0	9.20
1	9.0	9.43
2	8.0	8.23
3	10.0	10.14
4	8.0	8.95
5	9.0	8.62
6	11.0	10.40
7	10.0	9.66
8	10.5	9.70
9	9.0	9.03

6 What do you lose by only knowing mean/median?

Suppose that $X = \{x_i\}$ represents wait times in minutes at a certain bus-stop at 7:00 AM each day. Over n days, you collect the wait time values $x_1, x_2, \ldots, x_n$. The mean wait time

$$\mu(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

is surely a relevant statistic!

Let's consider a number of different possible outcomes

```
X = pd.DataFrame({"week1":[1,2,5,7,3,6],
                  "week2":[1,1,1,7,7,7],
                  "week3":[3,3,3,5,5,5],
                  "week4":[1,2,3,5,6,70],
                  "week5":[1,2,3,3,5,10]})
```

Let's compute the mean/median for the alternative outcomes.

[8]:
```
X = pd.DataFrame({"week1":[1,2,5,7,3,6],
                  "week2":[1,1,1,7,7,7],
                  "week3":[3,3,3,5,5,5],
                  "week4":[1,2,3,5,6,70],
                  "week5":[1,2,3,3,5,10]})
X
```

[8]:

	week1	week2	week3	week4	week5
0	1	1	3	1	1
1	2	1	3	2	2
2	5	1	3	3	3
3	7	7	5	5	3
4	3	7	5	6	5

```
5      6      7      5     70     10
```

[9]:
```
pd.DataFrame({"mean":X.mean(),
              "median":X.median()})
```

[9]:
```
       mean  median
week1   4.0     4.0
week2   4.0     4.0
week3   4.0     4.0
week4  14.5     4.0
week5   4.0     3.0
```

6.0.1 Observe:

The crucial observation is that a number of different outcomes produce the same or similar medians and means!

7 Variance

The preceding data was supposed to persuade you that by themselves, the mean and median hide information – e.g. the "range" of the values taken.

The **variance** of X, written $\mathrm{var}(X)$, is the mean of the following values:

$$\{(x_i - \mu)^2 \mid i = 1, 2, \ldots, n\};$$

thus

$$\mathrm{var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2.$$

The variance amounts to the average distance (really: squared distance) of the data values from the average value.

Unfortunately the *units* of the variance are the square of those of X. E.g. if the values in X measure minutes (waiting for the bus, say) then the units of $\mathrm{var}(X)$ are min^2.

So rather than the variance, one often consider the **standard deviation** $\sigma = \sigma(X)$ which by definition is the *square root* of the variance:

$$\sigma(X) = (\mathrm{var}(X))^{1/2} = \left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2\right)^{1/2}.$$

The units of $\sigma(X)$ match those of X.

[10]:
```
X
```

[10]:
```
   week1  week2  week3  week4  week5
0      1      1      3      1      1
```

```
1      2      1      3      2      2
2      5      1      3      3      3
3      7      7      5      5      3
4      3      7      5      6      5
5      6      7      5     70     10
```

[11]:
```
pd.DataFrame({"variance": X.var(),
              "standard deviation":X.std()})
```

[11]:

	variance	standard deviation
week1	5.6	2.366432
week2	10.8	3.286335
week3	1.2	1.095445
week4	742.7	27.252523
week5	10.4	3.224903

8 Random Variables & probability density functions

Now view the data set X as arising from some *process*.

For example, X may be generated by rolling a 6-sided dice. Or X may be generated by finding the wait-time at a certain bus stop at 7:00 AM.

We want to be able to study or speak about the process that controls the generation of this data.

For this, we want to define the notion of a *random variable*. If you look e.g. at the discussion on the wikipedia page you'll see the following:

> In probability and statistics, a random variable [...] is described informally as a variable whose values depend on outcomes of a random phenomenon
>
> A random variable's possible values might represent the possible outcomes of a yet-to-be-performed experiment, or the possible outcomes of a past experiment whose already-existing value is uncertain (for example, because of imprecise measurements or quantum uncertainty). They may also conceptually represent either the results of an "objectively" random process (such as rolling a die) or the "subjective" randomness that results from incomplete knowledge of a quantity. The meaning of the probabilities assigned to the potential values of a random variable is not part of probability theory itself

So we'll just say that a random variable X is a variable that assumes a value each time a particular random process is realized.

8.1 Examples of random variables:

- The value Y representing the outcome of rolling a 6-sided dice.

  Y is a *discrete* random variable, because Y can only take the values $\{1, 2, 3, 4, 5, 6\}$.

- The value T representing wait-time at the bus, in minutes.

  T is a *continuous* random variable, since the wait time is described by a non-negative *real number*.

9 Probability distribution

A probability distribution describes the probabilities with which a random variable assume its possible values.

For example, if Y is the random variable as before which describes a roll of a *fair* 6-sided dice, there are 6 outcomes: $r = 1, 2, 3, 4, 5, 6$. The probability that Y assumes any of these values is

$$P(Y = r) = \frac{1}{6}.$$

Let $Y2$ be the random variable that describes the result of adding the values for a simultaneous roll of 2 fair 6-sided dice. Here the outcomes are the values $r = 2, 3, 4, \ldots, 12$.

Let's compute the probabilities. The main thing we need is to count the number of times that a number k can be written as a sum $k = i + j$ where $1 \leq i, j \leq 6$.

We use the `value_counts` method for this:

[12]:
```
from itertools import product
I = [1,2,3,4,5,6]

results_2 = pd.DataFrame([i+j for i,j in product(I,I)])
Y2_prob = results_2.value_counts()/len(I)**2

Y2_prob
```

[12]:
```
7     0.166667
6     0.138889
8     0.138889
5     0.111111
9     0.111111
4     0.083333
10    0.083333
3     0.055556
11    0.055556
2     0.027778
12    0.027778
Name: count, dtype: float64
```

The indices `r` in the dataframe `Y2_prob` represent the possible dice-roll values, and the values represent the corresponding probabilities.

Continuing this point of view, consider the random variable Y3 determined by the sum of 3 simultaneous rolls of a 6-sided dice.

[13]:
```
results_3 = pd.DataFrame([i+j+k for i,j,k in product(I,I,I)])
Y3_prob = results_3.value_counts()/len(I)**3

Y3_prob
```

[13]:
```
10    0.125000
11    0.125000
9     0.115741
12    0.115741
8     0.097222
13    0.097222
7     0.069444
14    0.069444
6     0.046296
15    0.046296
5     0.027778
16    0.027778
4     0.013889
17    0.013889
3     0.004630
18    0.004630
Name: count, dtype: float64
```

[14]:
```
##
results_1 = pd.DataFrame([i for i in I])
Y1_prob = results_1.value_counts()/len(I)
Y1_prob
```

[14]:
```
1    0.166667
2    0.166667
3    0.166667
4    0.166667
5    0.166667
6    0.166667
Name: count, dtype: float64
```

10 Probability distributions – case of a continuous random variable

For a continuous random variable X and a real number r, usually the probability $P(X = r)$ is **zero**; this reflects the fact that there are *a lot* of real numbers!

Instead, in this setting one considers the *probability density function* $f(x)$ with the following property:

the probability that X is in the interval $[a, b]$ is

$$P(a \leq X \leq b) = \int_a^b f(x)dx$$

Note that the value of the random variable X has to be *somewhere*, so in particular we require that

$$P(-\infty < X < \infty) = \int_{-\infty}^{\infty} f(x)dx = 1.$$

On the other hand, as is well-known from integral calculus, we have:

$$P(X = a) = \int_{a}^{a} f(x)dx = 0$$

11 Statistics for a discrete random variable

For a discrete random variable Y, the expected value $E(Y)$ is the weighted average of the possible values that Y can take. More precisely, if the values that Y can take are $I = \{y_1, y_2, \dots, y_N\}$, then

$$E(Y) = \sum_{i=1}^{N} y_i P(y_i) = \sum_{i=1}^{N} y_i P(Y = y_i).$$

and the *variance* of Y is

$$\text{var}(Y) = E\left((Y - \mu)^2\right) \quad \text{if } \mu = E(Y)$$

Example:

- one roll of 6-sided dice

  If Y represents rolling a fair 6-sided dice, $P(r) = P(Y = r) = \frac{1}{6}$, and so

  $$E(Y) = \sum_{i=1}^{6} \frac{1}{6} \cdot i = \frac{1+2+3+4+5+6}{6} = \frac{21}{6} = 3.5$$

  and the variance is

  $$\text{var}(Y) = E\left((Y - 3.5)^2\right) = \sum_{i=1}^{6} \frac{(3.5 - i)^2}{6}$$

Let's confirm this using `python`:

[15]:
```
Y_expected = sum([value*prob for (value,),prob in Y1_prob.items()],0)

Y_var = sum([(Y_expected - value)**2 * prob for (value,),prob in Y1_prob.
 ↪items()],0)

[Y_expected, Y_var]
```

[15]:
```
[3.5, 2.9166666666666665]
```

- two or three 6-sided dice

  If $Y2$, $Y3$ represent the simultaneously rolling of two, resp. three, 6-sided dice and adding the results, we computed above the values $P(Y2 = r)$ for $2 \leq r \leq 12$ and $P(Y3 = r)$ for $3 \leq r \leq 18$.

  Let's compute the expected values. Recall that the probabilities are stored in the variables `Y2_prob` and `Y3_prob`.

[16]:
```
Y2_expected = sum([value*prob for (value,),prob in Y2_prob.items()],0)
Y2_var = sum([(Y2_expected - value)**2 * prob for (value,),prob in Y2_prob.
 ↪items()],0)

[Y2_expected, Y2_var]
```

[16]:
```
[6.999999999999999, 5.833333333333334]
```

[17]:
```
Y3_expected = sum([value*prob for (value,),prob in Y3_prob.items()],0)
Y3_var = sum([(Y3_expected - value)**2 * prob for (value,),prob in Y3_prob.
 ↪items()],0)


[Y3_expected, Y3_var]
```

[17]:
```
[10.5, 8.749999999999998]
```

12 Statistics for a continuous random variable (definitions)

Suppose the continuous random variable Y is determined by the probability distribution function $f(x)$. The expected value $E(Y)$ is given by

$$\mu = E(Y) = \int_{-\infty}^{\infty} x \cdot f(x)dx$$

The *variance* of Y is defined to be

$$\sigma^2 = \text{var}(Y) = \int_{-\infty}^{\infty} (x-\mu)^2 \cdot f(x)dx$$

and the standard deviation is

$$\sigma = \sqrt{\text{var}(Y)}$$

.

13 Normal distribution

Perhaps the most important probability distribution function is known as the *Gaussian* or *normal* distribution. For mean μ and standard deviation σ, the probability density function is determined by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

This probability distribution function determines a continuous random variable Y_{normal} with expected value μ and standard deviation σ; these latter conditions mean:

$$\mu = E(Y_{\text{normal}}) = \int_{-\infty}^{\infty} x \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) dx$$

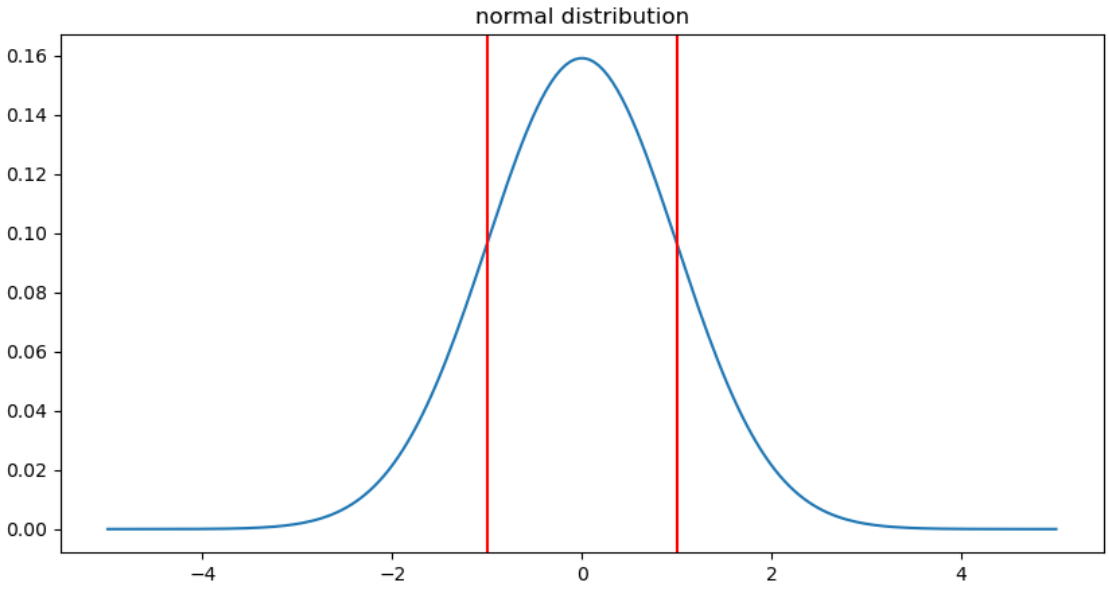
and

$$\sigma^2 = \text{var}(Y_{\text{normal}}) = \int_{-\infty}^{\infty} (x-\mu)^2 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) dx$$

---

[18]:
```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(-5,5,200)
## density function with mu = 0 and sigma = 1
def f(x):
    return np.exp(-x**2/2)/(2*np.pi)

fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(x, f(x))
ax.set_title("normal distribution")
ax.axvline(x=1,color="red")
ax.axvline(x=-1,color="red")
```

[18]:
```
<matplotlib.lines.Line2D at 0x7f55fcd01710>
```



14 Law of Large Numbers

Suppose that Y is a random variable (discrete or continuous), and consider a sample $X_1, \ldots, X_n$ of n events drawn from the random variable, for $n \geq 1$. We write $\overline{X}_n$ for the mean of this sample:

$$\overline{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

We can view the sample mean $\overline{X}_n$ as a random variable (depending on the choices made)!

Law of Large Numbers: The sample means $\overline{X}_n$ converges to the expected value $E(Y)$ as $n \to \infty$.

This formulation hides a bit of complexity; namely, what does precisely is meant by convergence in this context? We'd like to write something like

$$\overline{X}_n \xrightarrow{\text{prob}} E(Y) \quad \text{as } n \to \infty$$

to indicate "converges in probability" (which we haven't defined…). Here is more a precise form, which is known as the *weak* law of large numbers. This weak law says that

$$\lim_{n \to \infty} P(|\overline{X}_n - E(Y)| > \epsilon) = 0 \quad \text{for any } \epsilon > 0.$$

Roughly speaking, this formulation means that the probability that the sample mean is even "slightly" different from the expected value goes to 0 as the number of trials goes to infinity.

We aren't going to try to be completely precise here – this is a course about modeling, not the full details of probability! – but it is perhaps worth mention that we need to be more precise about

what is meant by the probability $P(|\overline{X}_n - E(Y)| > \epsilon)$; this depends on viewing the sample mean $\overline{X}_n$ as a random variable.

Very roughly speaking, the idea is that as the sample sizes grow, you can expect the sample mean to behave like the expected value of the random variable.

If you'd like to know a bit more, the wikipedia article on the Law of Large Numbers is worth scanning.

15 The Central Limit Theorem

Let's keep the notation and terminology from the preceding discussion. Thus the $X_1, \ldots, X_n$ represent random samples of n events drawn from the random variable Y, for $n \geq 1$.

We view the choice of each sample as X_i as a random variable; we want to suppose that these random variables are independent from one-another and identically distributed – abbreviated i.i.d. In particular, $E(X_i) = E(Y) = \mu$ and $\text{var}(X_i) = \text{var}(Y) = \sigma^2$.

Recall that we view the sample mean $\overline{X}_n$ as a random variable. Thus, we can view $\sqrt{n} \cdot (\overline{X}_n - \mu)$ as a random variable, which is thus given by its distribution – i.e. by its probability density function.

Theorem: (Central Limit Theorem) If $\{X_1, \ldots, X_n\}$ are i.i.d., then as $n \to \infty$, the distribution of $\sqrt{n} \cdot (\overline{X}_n - \mu)$ converges to the normal distribution with expected value 0 and variance σ^2.

Note that if f_n is the probability distribution function for the random variable $\overline{X}_n$, then for real numbers $a < b$,

$$P(a < \overline{X}_n < b) = \int_a^b f_n(x) ds$$

so that

$$(\clubsuit) = P\left(\sqrt{n}(a - \mu) < \sqrt{n}(\overline{X}_n - \mu) < \sqrt{n}(b - \mu)\right) = \int_a^b f_n(x) dx$$

On the other hand, for large n the Theorem says that, at least roughly,

$$(\clubsuit) \approx \frac{1}{\sigma\sqrt{2\pi}} \int_{\sqrt{n}(a-\mu)}^{\sqrt{n}(b-\mu)} \exp\left(-\left(\frac{x}{\sigma}\right)^2\right) dx$$

(The RHS reflects the indicated *normal distribution*). Now use the substitution $v = \dfrac{x}{\sqrt{n}} + \mu$; the RHS becomes

$$\frac{1}{(\sigma/\sqrt{n})\sqrt{2\pi}} \int_a^b \exp\left(-\left(\frac{v - \mu}{\sigma/\sqrt{n}}\right)^2\right) dv$$

Conclusion: the theorem says that for large enough n, the distribution of $\overline{X}_n$ is close to the normal distribution with mean μ and variance σ^2/n.

16 Examples

16.0.1 Coin tossing

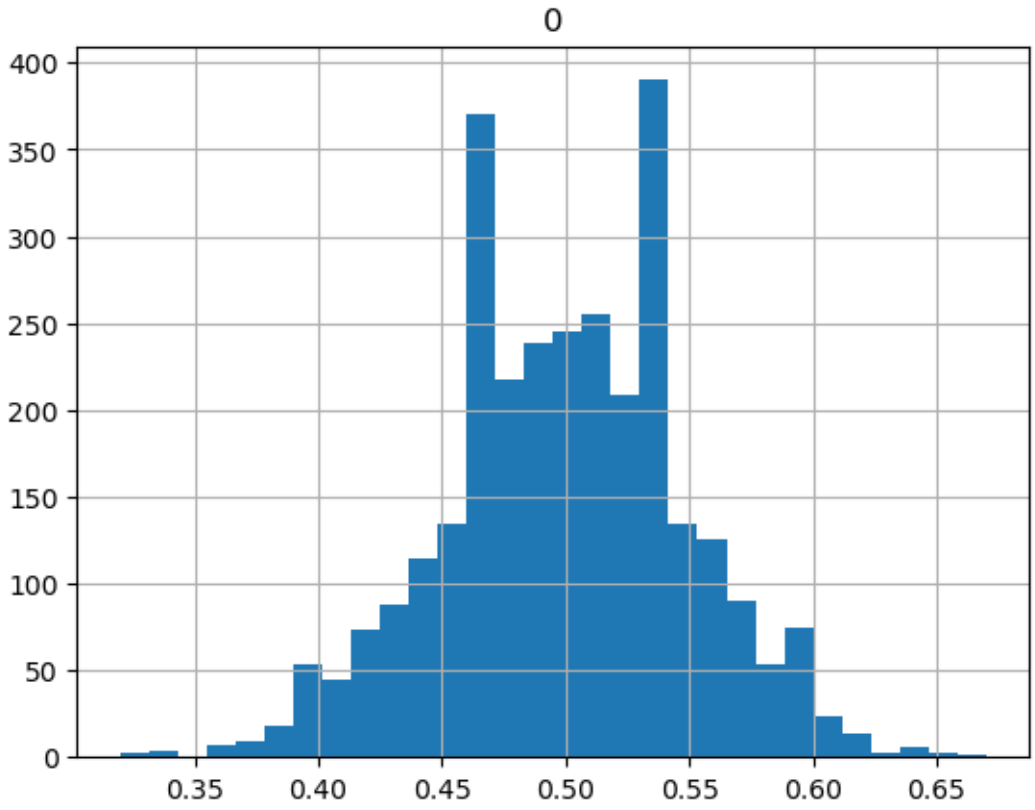
First, view the a coin toss as a random variable making a choice between `0` and `1`, and compute the mean of `num_tosses` choices – i.e. $X_1, \ldots, X_n$ all represent the result of a coin toss, and $\overline{X}_n$ is the mean of this trial.

We repeat this for several trials to describe the *distribution* of $\overline{X}_n$:

[29]:
```
def report(df):
    ## return a string with the standard deviation and mean of the
    ## DataFrame df
    return "\n".join([f"std:  {df.std().values}",
                      f"variance: {df.var().values}",
                      f"mean: {df.mean().values}"])

#------------------------------


## use  rng.choice([...],n) to pick n random elements from the list [...]

def trial(num_events,results):
    # return the data frame for a *trial* invoulving a number of *events*
    return pd.DataFrame(rng.choice(results,num_events)).mean()

def distribution(num_trials,num_events,results=[0,1]):
    return pd.DataFrame( [ trial(num_events,results) for _ in range(num_trials)
↪] )

cd_200=distribution(3000,200)
```

[38]:
```
# 3000 trials, each consisting of 100 coin tosses
cd_100=distribution(3000,100,results=[0,1])

cd_100.hist(bins=30)
print(report(cd_100))
```

```
std:  [0.04967482]
variance: [0.00246759]
mean: [0.49939333]
```



Of course, this confirms that the mean is tending to `0.5`.

Recall that each of the "coin toss" random variables X_i has mean `1/2` and variance

$$\text{var}(X_i) = \left(\left(0-\frac{1}{2}\right)^2 + \left(1-\frac{1}{2}\right)^2 \cdot\right)\frac{1}{2} = \frac{1}{4}.$$

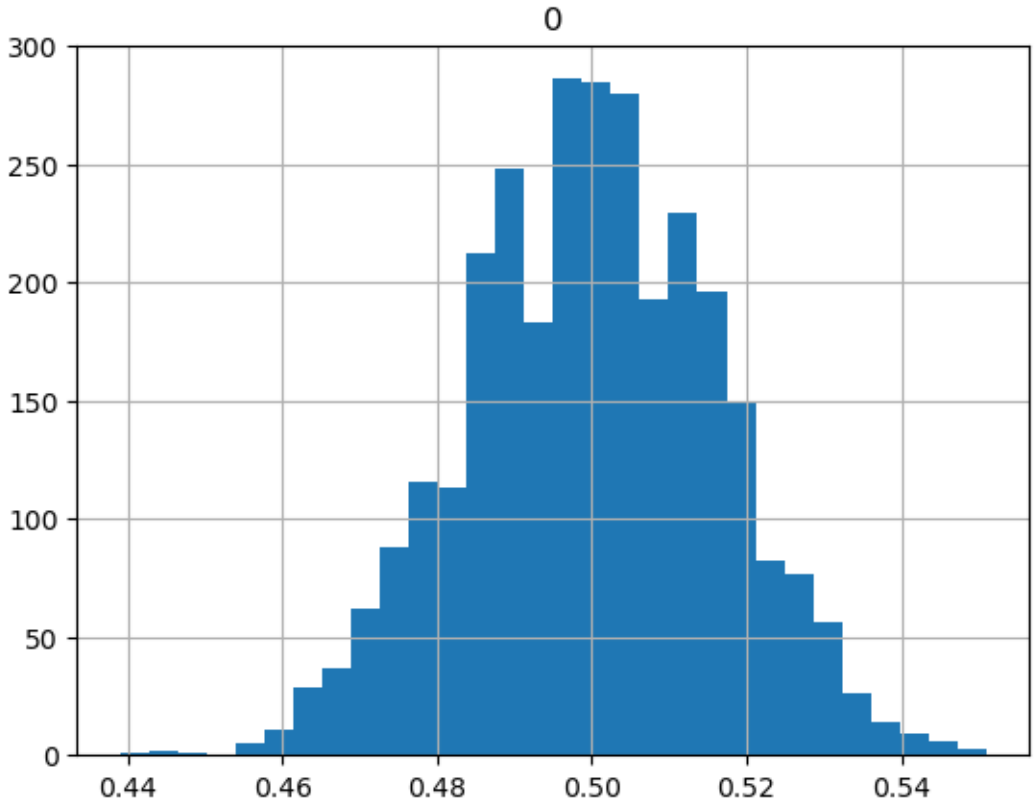
Thus the central limit theorem predicts that the variance $\text{var}(\overline{X}_n)$ should be $\frac{1}{4n}$ and $\sigma = \sqrt{\frac{1}{4n}}$.

For $n = 100$, this amounts to $\text{var}(\overline{X}_{100}) = \frac{1}{400} = 0.0025$ and $\sigma = \sqrt{\frac{1}{400}} = .05$

For $n = 200$, this amounts to $\text{var}(\overline{X}_{100}) = \frac{1}{800} = 0.00125$ and $\sigma = \sqrt{\frac{1}{800}} = .035355$

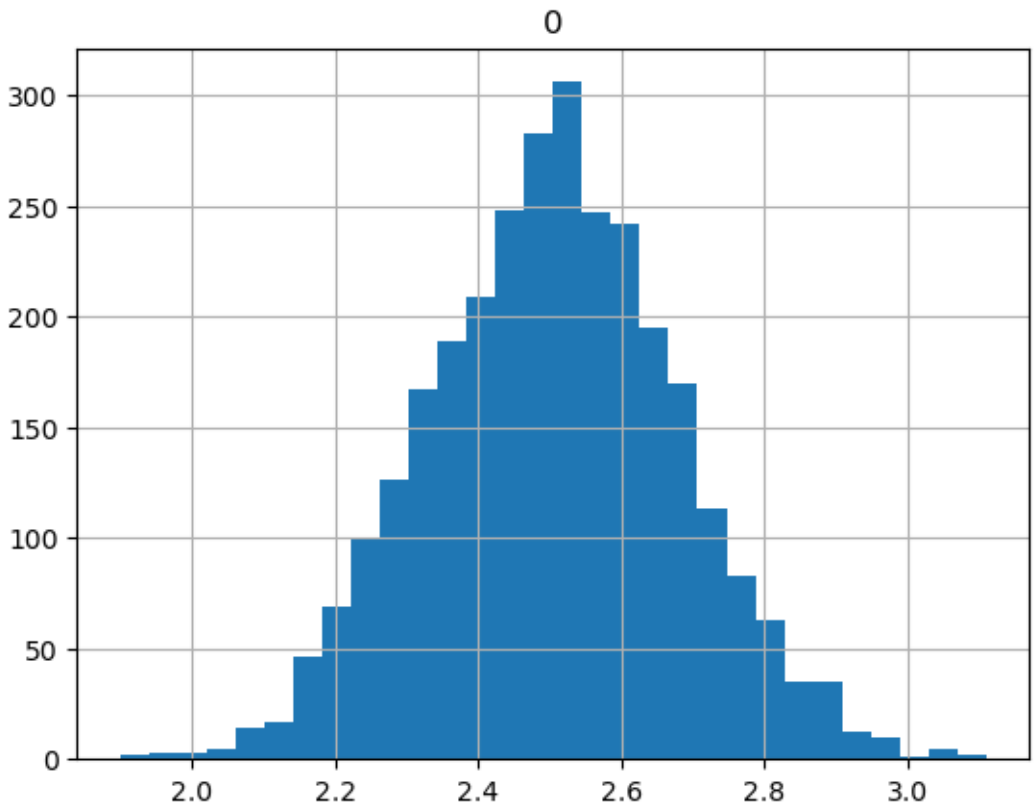
[41]:
```
# 3000 trials, each consisting of 200 coin tosses
cd_200=distribution(3000,1000,results=[0,1])

cd_200.hist(bins=30)
print(report(cd_200))
```

```
std:  [0.01624675]
variance: [0.00026396]
mean: [0.49989233]
```



[35]:
```
# 3000 trials, each consisting of 100 dice rolls
cd_100=distribution(3000,100,results=range(6))

cd_100.hist(bins=30)
print(report(cd_100))
```

```
std:  [0.17314]
variance: [0.02997746]
mean: [2.50499333]
```



[ ]:
```
cd_200.hist(bins=30)
print(report(cd_200))
```

16.1 Dice Rolling

We consider an ordinary 6-sided dice. We'll look at two scenarios: a *fair* die where all rolls occurs with equal probability, and a *broken* die.

We view the random variable X_i as the outcome of a die-roll. We consider a sample $X_1, X_2, \ldots, X_n$ and we compute the sample mean $\overline{X}_n$.

Our goal is to describe the distribution of the sample mean $\overline{X}_n$.

[42]:
```
six_sided_dice = [1,2,3,4,5,6]

fair_prob   = np.array(6*[1/6])
broken_prob = (1./9)*np.array([3,2,1,1,1,1])

## rng.choice(six_sided_dice,n) = rng.choice(six_sided_dice,n,p=fair_prob)␣
 ↪returns a list of
##    n random of a fair six-sided dice
```

```
## rng.choice(six_sided_dice,n,p=broken_prob) again rolls the dice n times, but␣
 ↪with
##    the indicated probabilities for the rolls

def dice_trial(pr=fair_prob):
    ## dice_trial() returns a function, suitable for passing to map(...)
    ## if f = dice_trial(), then f(n) returns a DataFrame with the results
    ## of n dice roll results
    return lambda n: pd.DataFrame(rng.choice(six_sided_dice,n,p=pr)).mean()


def dice_distribution(num_trials,sample_size,pr=fair_prob):
    return pd.DataFrame(map(dice_trial(pr),num_trials*[sample_size]))
```

Recall for a fair die, each random variable has expected value

$$E(X_i) = \frac{1+2+3+4+5+6}{6} = 3.5$$

and variance

$$\text{var}(X_i) = \frac{1}{6}\sum_{j=1}^{6}(j-3.5)^2 \approx 2.9167$$

Our "broken die" probabilities are `[1/3,2/9,1/9,1/9,1/9,1/9]`

So our expected value for the broken dice rolls can be determined using the following code:

[43]:
```
broken_prob = (1./9)*np.array([3,2,1,1,1,1])
E=sum([(j+1)*broken_prob[j] for j in range(6)],0)

V=sum([(E-j-1)**2*broken_prob[j] for j in range(6)],0)

[E,V]
```

[43]:
```
[2.7777777777777772, 3.0617283950617282]
```

Thus in this case $E(X_i) \approx 2.778$ and $\text{var}(X_i) \approx 3.062$.

Lets simulate some trials for both sorts of diceand various sample sizes.

[45]:
```
dd_fair_50 = dice_distribution(num_trials=3000,sample_size = 50)
dd_fair_100 = dice_distribution(num_trials=3000,sample_size = 100)

dd_broken_50 = dice_distribution(num_trials=3000,sample_size=50,pr=broken_prob)
dd_broken_100 =␣
 ↪dice_distribution(num_trials=3000,sample_size=100,pr=broken_prob)
```

For a fair die, and n trials, the central limit theorem predicts

$$\text{var}(\overline{X}_n) = \frac{2.9167}{n}.$$

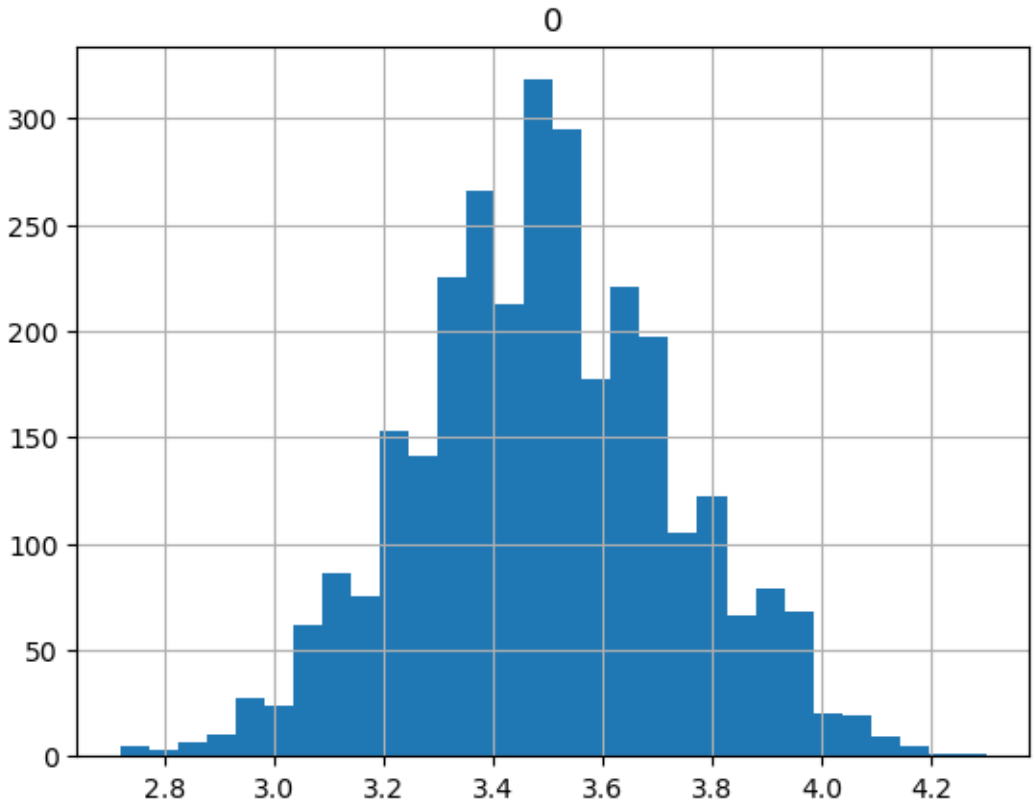
n	var
50	0.05833
100	0.029167

[47]:
```
print(report(dd_fair_50))
print()
print(report(dd_fair_100))
```

```
std:  [0.24042032]
variance: [0.05780193]
mean: [3.49376]

std:  [0.17297698]
variance: [0.02992104]
mean: [3.50052]
```

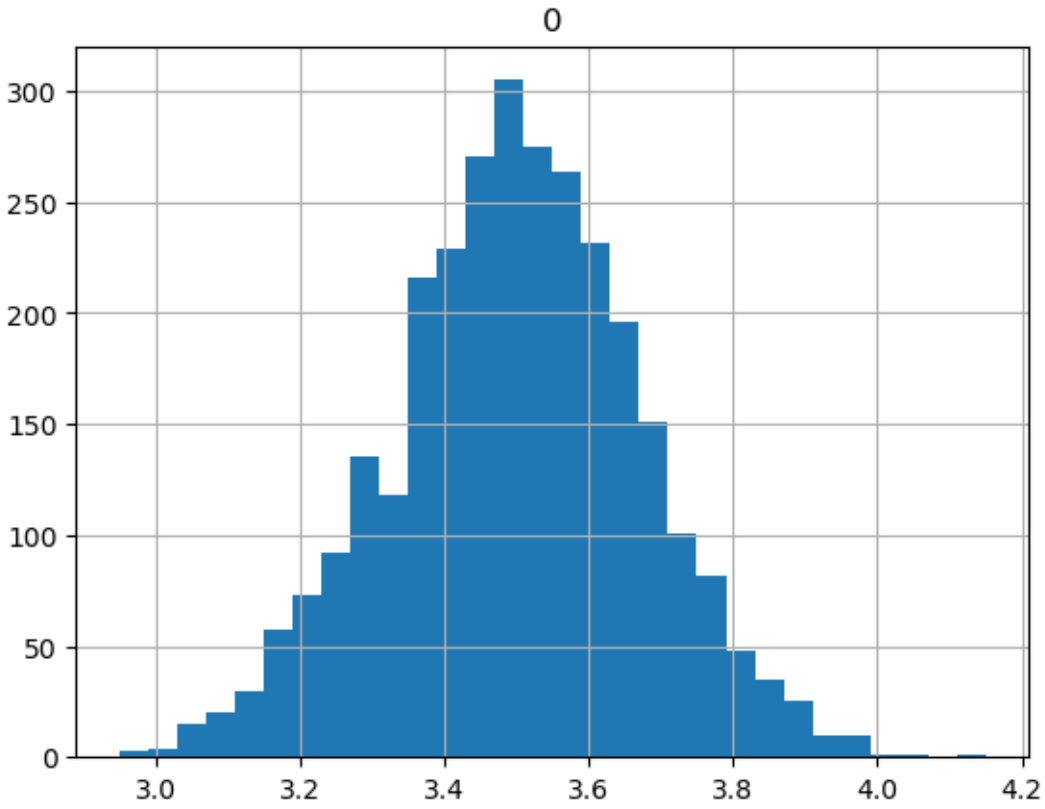
[48]:
```
dd_fair_50.hist(bins=30)
```

[48]:
```
array([[<Axes: title={'center': '0'}>]], dtype=object)
```



[49]:
```
dd_fair_100.hist(bins=30)
```

[49]:
```
array([[<Axes: title={'center': '0'}>]], dtype=object)
```



For our broken die, and n trials, the central limit theorem predicts

$\text{var}(\overline{X}_n) = \dfrac{3.062}{n}$.

n	var
50	0.06124
100	0.03062

[ ]:
```
print(report(dd_broken_50))
print(report(dd_broken_100))
```

[ ]:
```
## "broken" dice results
##
dd_broken_50.hist(bins=30)
```

[ ]:
```
dd_broken_100.hist(bins=30)
```

17 Random Number Generation

We have been using the random number generator provided by `numpy` without much comment.

e.g. we've seen that

```
from numpy.random import default_rng
rng = default_rng()
```

produces an object `rng` with methods include `random` and `choice`:

```
>>> rng.random(4)
array([0.1403315 , 0.32259798, 0.60620108, 0.7533085 ])

>>> rng.choice(["red","blue","green"],5)
array(['blue', 'green', 'blue', 'green', 'red'], dtype='<U5')
```

Really, these methods only produce so-called *pseudo-random* results. A common way to produce a list of *pseudo-random* numbers is to begin with a `seed` value X_0, and then describe an algorithm for computing X_n from X_{n-1} for any $n \geq 1$.

A standard example for generating random integers depends on a choice of large integers a, b, m; one then sets

$$X_n = aX_{n-1} + b \pmod{n}$$

If the algorithm is good enough, the resulting sequence of numbers "seems random" e.g. in the sense of passing statistical tests for randomness.

Note however, that if you know the seed and the algorithm, you can construct the sequence. One says that the sequence is deterministic – which doesn't seem very random!! Such determinism can be problematic e.g. in crypotgraphy. But for our point-of-view, pseudo-random numbers are very useful in modeling.

For example, statistics and random number generators will be very important to modeling applications of so-called *Monte-Carlo simulations* that we will study next week.

[ ]: