Relative error of point estimates

SAMPLING IN PYTHON

James Chapman
Content Developer, DataCamp

Sample size is number of rows

```
len(coffee_ratings.sample(n=300))
```

```
300
```

```
len(coffee_ratings.sample(frac=0.25))
```

```
334
```

Various sample sizes

```
coffee_ratings['total_cup_points'].mean()
```

```
82.15120328849028
```

```
coffee_ratings.sample(n=10)['total_cup_points'].mean()
```

```
83.027
```

```
coffee_ratings.sample(n=100)['total_cup_points'].mean()
```

```
82.4897
```

```
coffee_ratings.sample(n=1000)['total_cup_points'].mean()
```

```
82.1186
```

Relative errors

Population parameter:

```
population_mean = coffee_ratings['total_cup_points'].mean()
```

Point estimate:

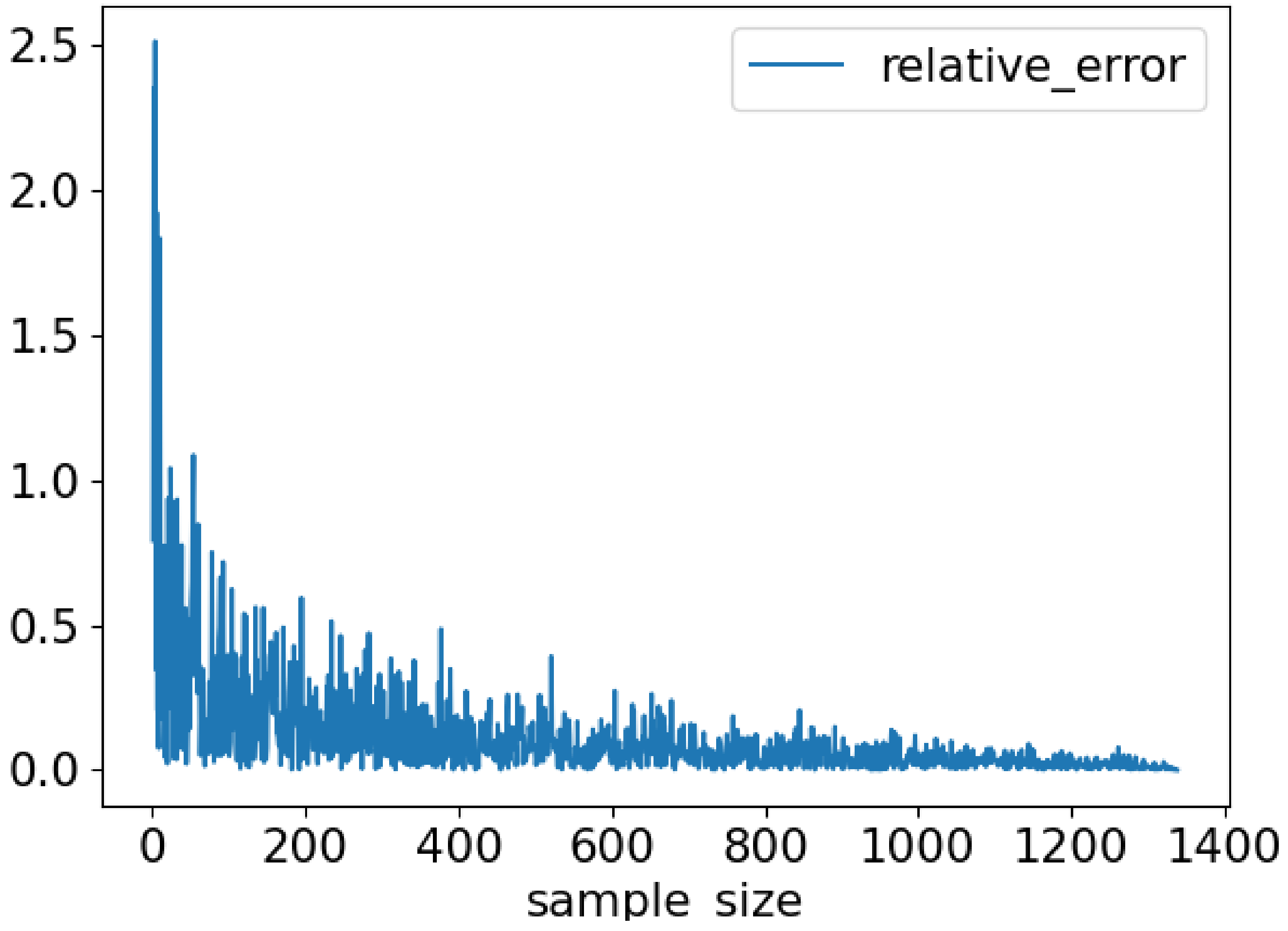
```
sample_mean = coffee_ratings.sample(n=sample_size)['total_cup_points'].mean()
```

Relative error as a percentage:

```
rel_error_pct = 100 * abs(population_mean-sample_mean) / population_mean
```

Relative error vs. sample size

```
import matplotlib.pyplot as plt
errors.plot(x="sample_size",
            y="relative_error",
            kind="line")
plt.show()
```



Properties:

- Really noise, particularly for small samples
- Amplitude is initially steep, then flattens
- Relative error decreases to zero (when the sample size = population)

Let's practice!

SAMPLING IN PYTHON

Creating a sampling distribution

SAMPLING IN PYTHON

James Chapman
Content Developer, DataCamp

Same code, different answer

```
coffee_ratings.sample(n=30)['total_cup_points'].mean()
```

```
82.53066666666668
```

```
coffee_ratings.sample(n=30)['total_cup_points'].mean()
```

```
81.97566666666667
```

```
coffee_ratings.sample(n=30)['total_cup_points'].mean()
```

```
82.68
```

```
coffee_ratings.sample(n=30)['total_cup_points'].mean()
```

```
81.675
```

Same code, 1000 times

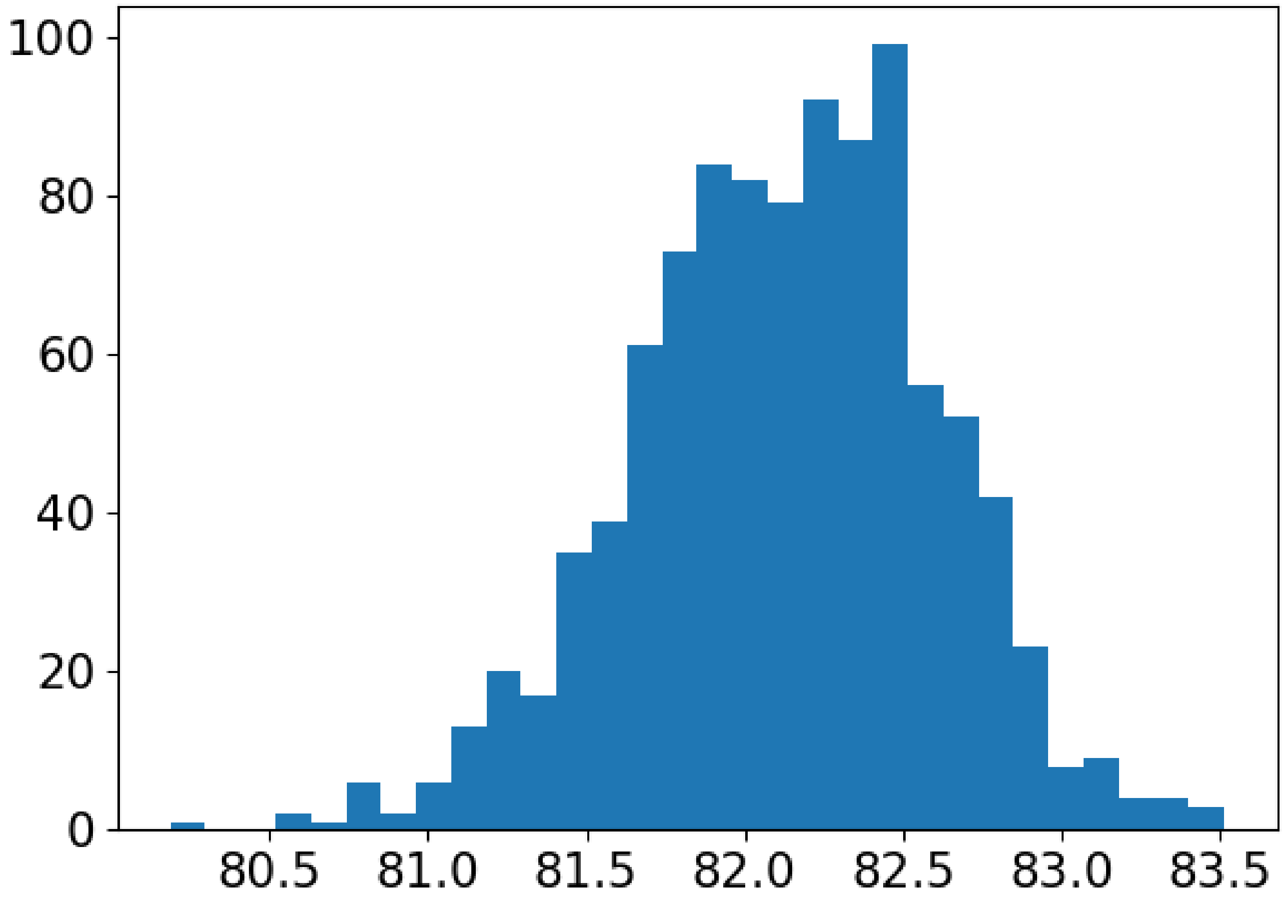
```python
mean_cup_points_1000 = []
for i in range(1000):
    mean_cup_points_1000.append(
        coffee_ratings.sample(n=30)['total_cup_points'].mean()
    )
print(mean_cup_points_1000)
```

```
[82.11933333333333, 82.55300000000001, 82.07266666666668, 81.76966666666667,
...
 82.74166666666666, 82.45033333333335, 81.77199999999999, 82.81633333333333]
```

Distribution of sample means for size 30

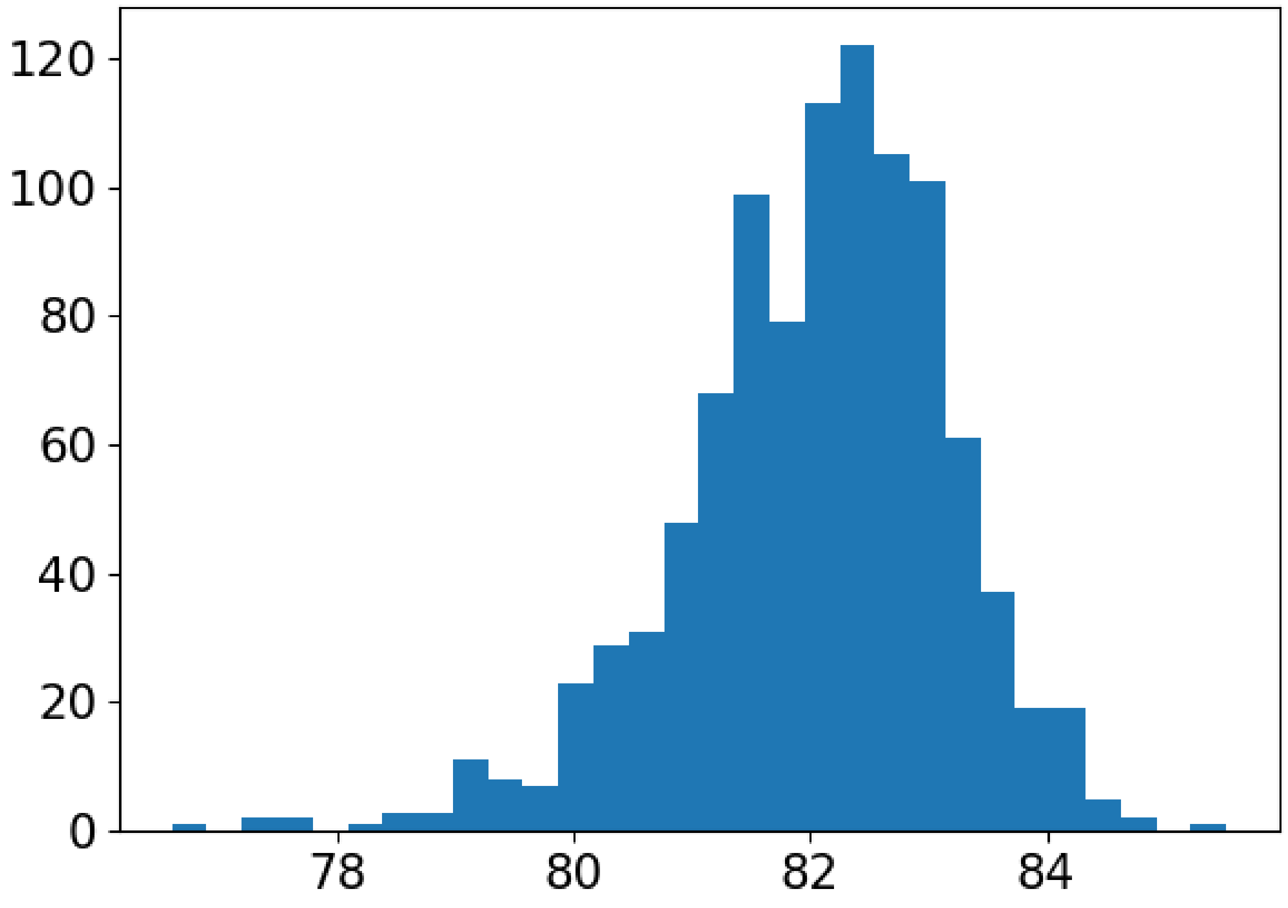
```
import matplotlib.pyplot as plt
plt.hist(mean_cup_points_1000, bins=30)
plt.show()
```

> A *sampling distribution* is a distribution of replicates of point estimates.

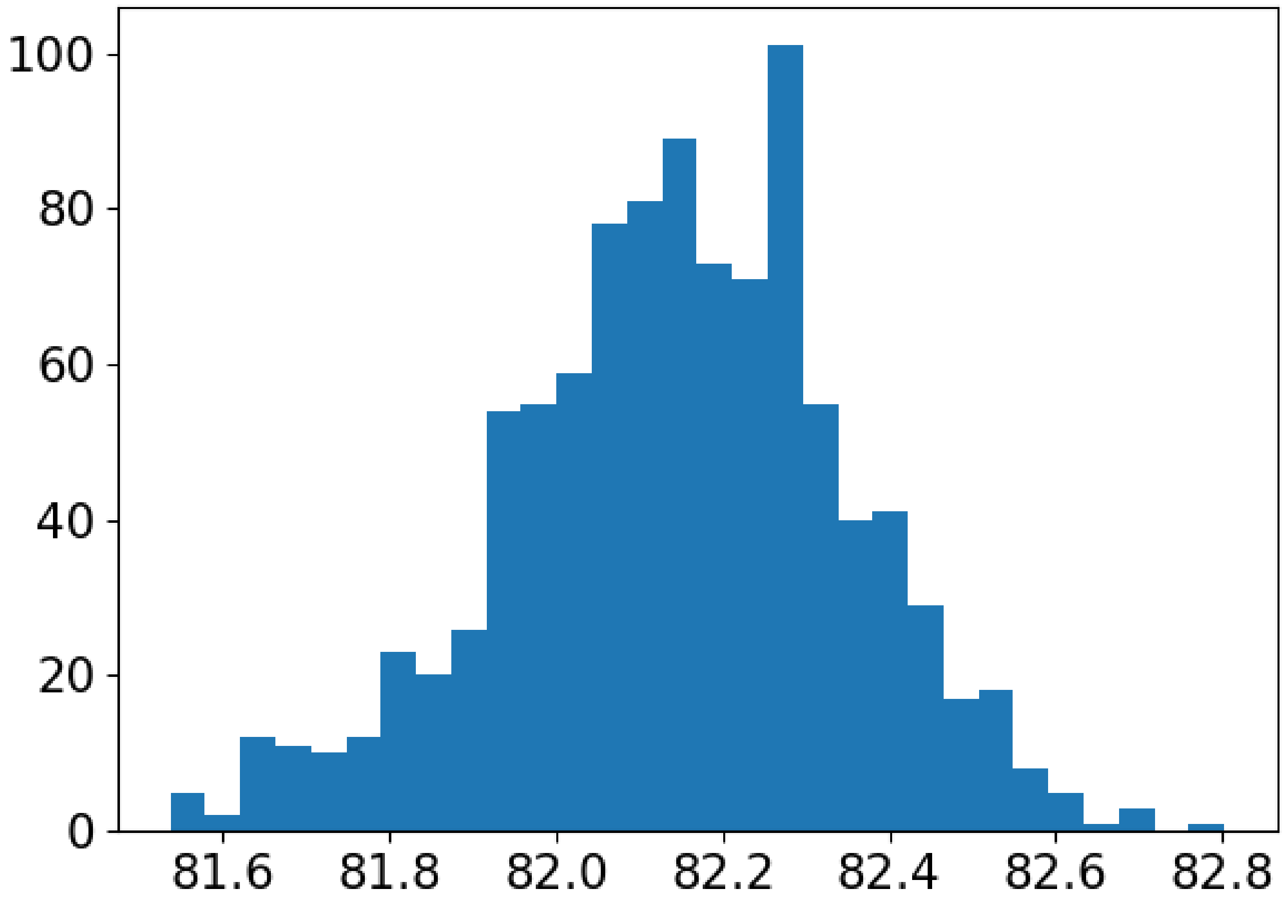


Different sample sizes

Sample size: 6



Sample size: 150



Let's practice!

SAMPLING IN PYTHON

Approximate sampling distributions

SAMPLING IN PYTHON

James Chapman
Content Developer, DataCamp

4 dice

```
dice = expand_grid(
  {'die1': [1, 2, 3, 4, 5, 6],
   'die2': [1, 2, 3, 4, 5, 6],
   'die3': [1, 2, 3, 4, 5, 6],
   'die4': [1, 2, 3, 4, 5, 6]
  }
)
```

```
      die1  die2  die3  die4
0        1     1     1     1
1        1     1     1     2
2        1     1     1     3
3        1     1     1     4
4        1     1     1     5
...    ...   ...   ...   ...
1291     6     6     6     2
1292     6     6     6     3
1293     6     6     6     4
1294     6     6     6     5
1295     6     6     6     6

[1296 rows x 4 columns]
```

Mean roll

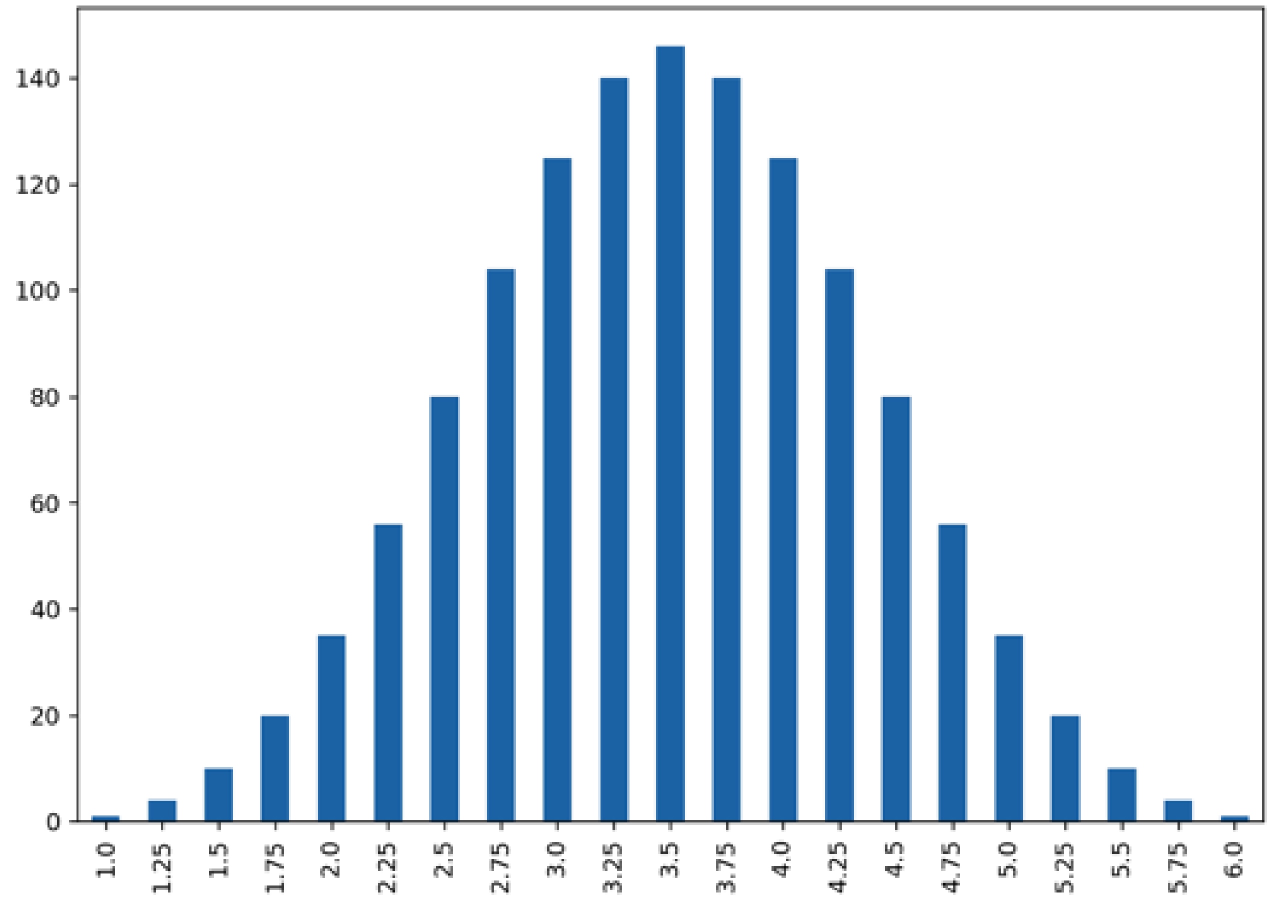
```
dice['mean_roll'] = (dice['die1'] +
                     dice['die2'] +
                     dice['die3'] +
                     dice['die4']) / 4
print(dice)
```

```
      die1  die2  die3  die4  mean_roll
0        1     1     1     1       1.00
1        1     1     1     2       1.25
2        1     1     1     3       1.50
3        1     1     1     4       1.75
4        1     1     1     5       2.00
...    ...   ...   ...   ...        ...
1291     6     6     6     2       5.00
1292     6     6     6     3       5.25
1293     6     6     6     4       5.50
1294     6     6     6     5       5.75
1295     6     6     6     6       6.00

[1296 rows x 5 columns]
```

Exact sampling distribution

```
dice['mean_roll'] = dice['mean_roll'].astype('category')
dice['mean_roll'].value_counts(sort=False).plot(kind="bar")
```

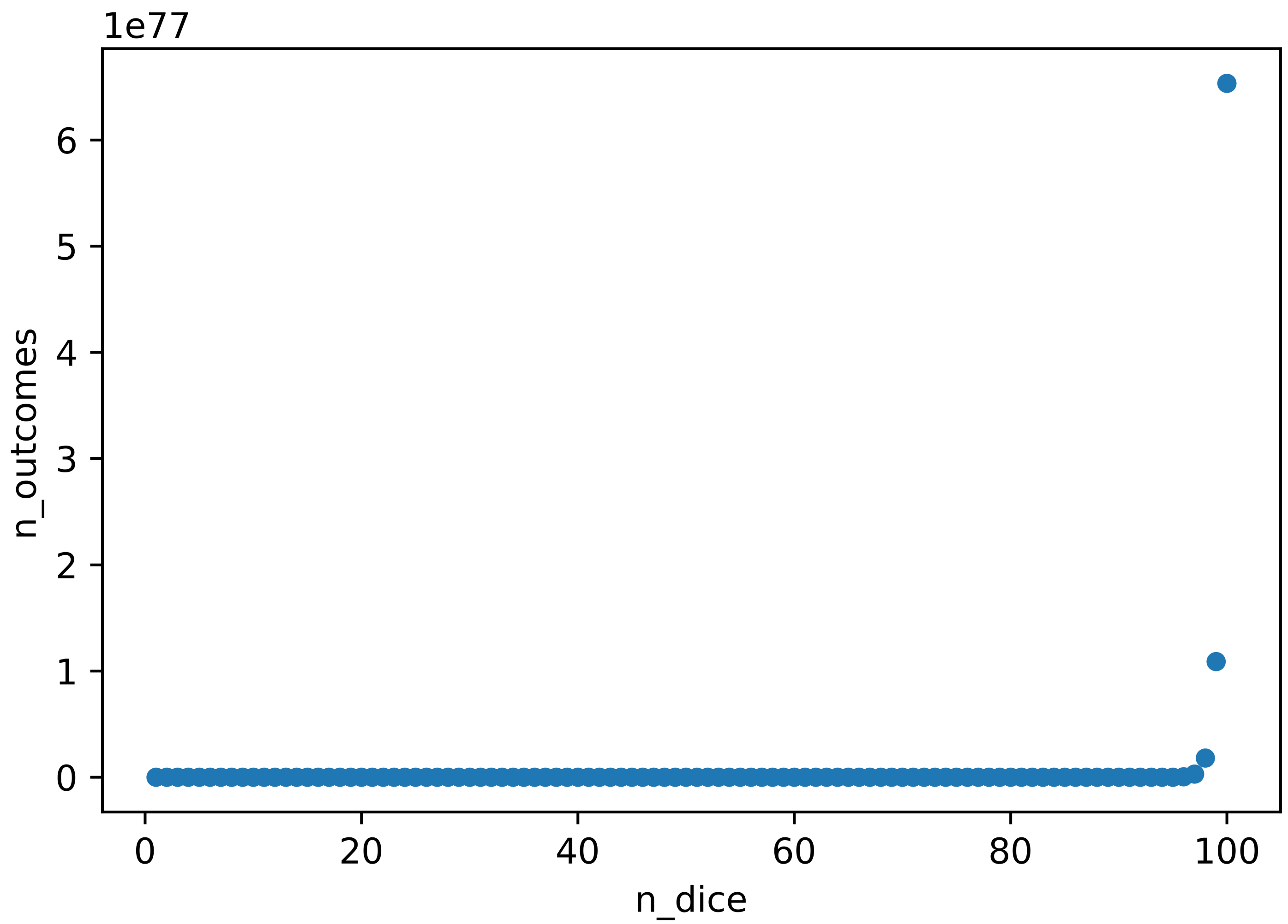


The number of outcomes increases fast

```
n_dice = list(range(1, 101))
n_outcomes = []
for n in n_dice:
    n_outcomes.append(6**n)

outcomes = pd.DataFrame(
  {"n_dice": n_dice,
  "n_outcomes": n_outcomes})
```

```
outcomes.plot(x="n_dice",
              y="n_outcomes",
              kind="scatter")
plt.show()
```



Simulating the mean of four dice rolls

```
import numpy as np

np.random.choice(list(range(1, 7)), size=4, replace=True).mean()
```

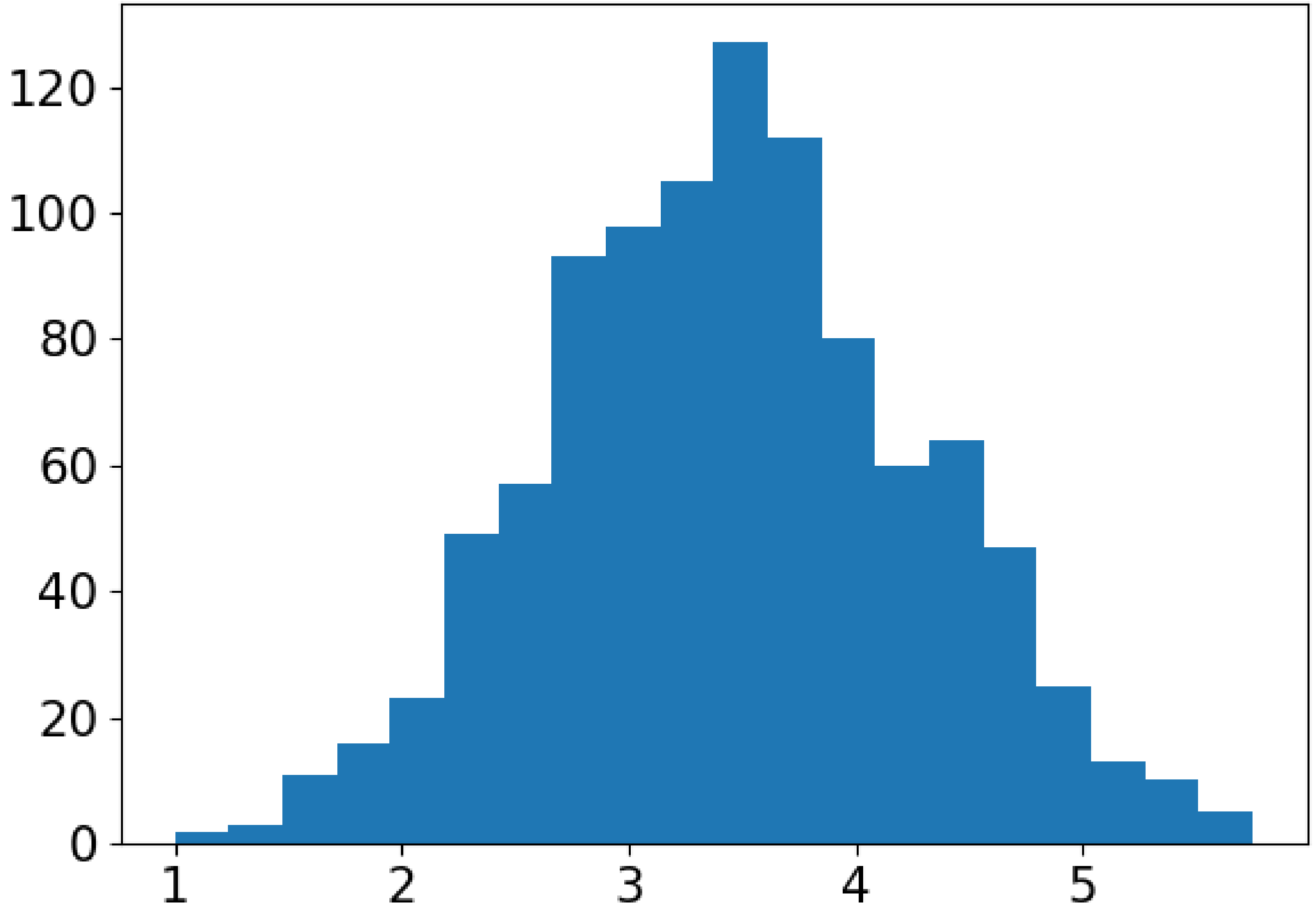
Simulating the mean of four dice rolls

```
import numpy as np
sample_means_1000 = []
for i in range(1000):
    sample_means_1000.append(
        np.random.choice(list(range(1, 7)), size=4, replace=True).mean()
    )
print(sample_means_1000)
```

```
[3.25, 3.25, 1.75, 2.0, 2.0, 1.0, 1.0, 2.75, 2.75, 2.5, 3.0, 2.0, 2.75,
...
1.25, 2.0, 2.5, 2.5, 3.75, 1.5, 1.75, 2.25, 2.0, 1.5, 3.25, 3.0, 3.5]
```

Approximate sampling distribution

```
plt.hist(sample_means_1000, bins=20)
```



Let's practice!

SAMPLING IN PYTHON

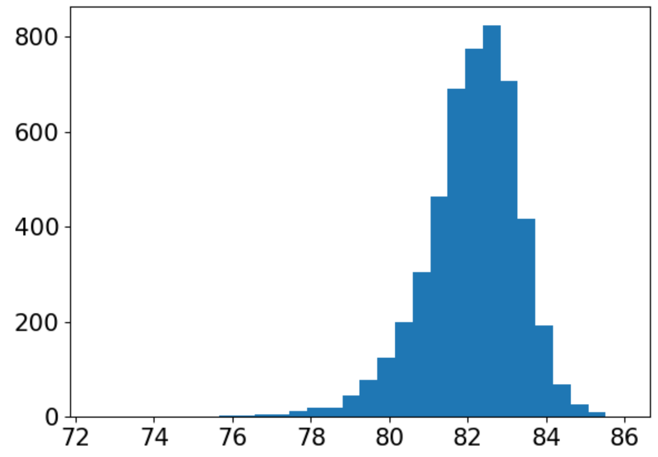
Standard errors and the Central Limit Theorem

SAMPLING IN PYTHON

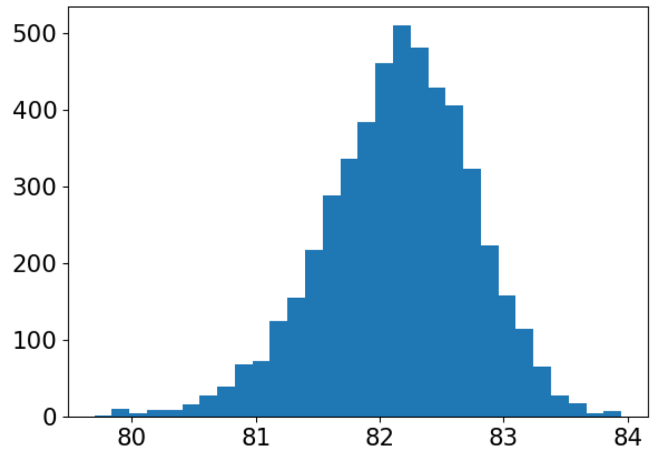
James Chapman
Content Developer, DataCamp

Sampling distribution of mean cup points

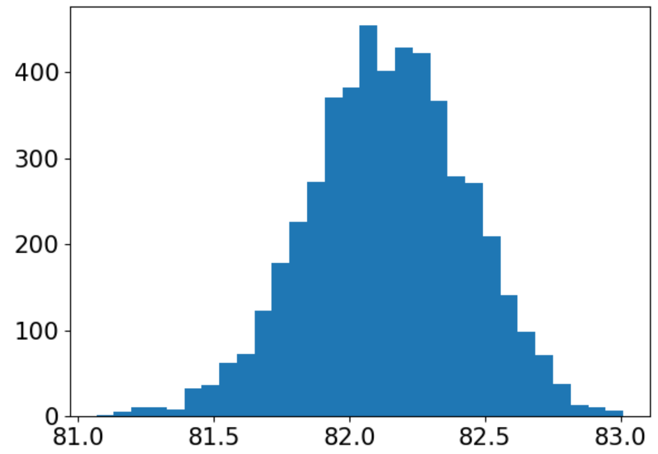
Sample size: 5



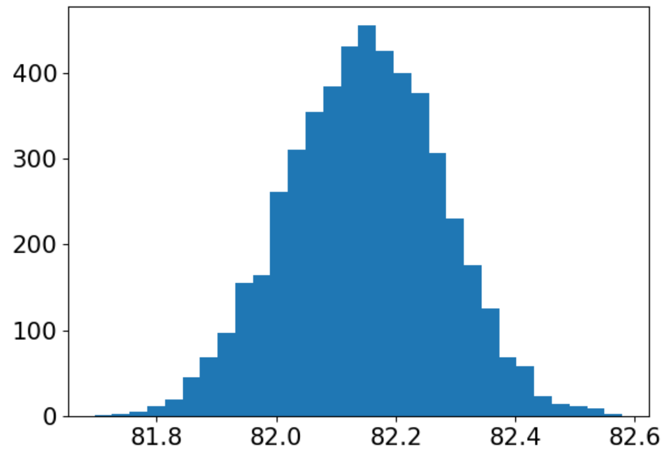
Sample size: 20



Sample size: 80



Sample size: 320



Consequences of the central limit theorem

> Averages of independent samples have approximately **normal distributions.**

As the sample size increases,

- The distribution of the averages gets *closer to being normally distributed*
- The width of the sampling distribution gets *narrower*

Population & sampling distribution means

```
coffee_ratings['total_cup_points'].mean()
```

```
82.15120328849028
```

Use `np.mean()` on each approximate sampling distribution:

Sample size	Mean sample mean
5	`82.18420719999999`
20	`82.1558634`
80	`82.14510154999999`
320	`82.154017925`

Population & sampling distribution standard deviations

```
coffee_ratings['total_cup_points'].std(ddof=0)
```

```
2.685858187306438
```

- Specify `ddof=0` when calling `.std()` on populations
- Specify `ddof=1` when calling `np.std()` on samples or sampling distributions

Sample size	Std dev sample mean
5	1.1886358227738543
20	0.5940321141669805
80	0.2934024263916487
320	0.13095083089190876

Population mean over square root sample size

Sample size	Std dev sample mean	Calculation	Result
5	`1.1886358227738543`	`2.685858187306438 / sqrt(5)`	`1.201`
20	`0.5940321141669805`	`2.685858187306438 / sqrt(20)`	`0.601`
80	`0.2934024263916487`	`2.685858187306438 / sqrt(80)`	`0.300`
320	`0.13095083089190876`	`2.685858187306438 / sqrt(320)`	`0.150`

Standard error

- Standard deviation of the sampling distribution
- Important tool in understanding sampling variability

Let's practice!

SAMPLING IN PYTHON