Under the Bayesian hood

BAYESIAN DATA ANALYSIS IN PYTHON

Michal Oleszak
Machine Learning Engineer

Bayes' Theorem revisited

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Bayes' Theorem revisited

$$P(\text{parameters}|\text{data}) = \frac{P(\text{data}|\text{parameters}) * P(\text{parameters})}{P(\text{data})}$$

- **P(parameters | data)** → **posterior distribution:** what we know about the parameters after having seen the data
- **P(parameters)** → **prior distribution:** what we know about the parameters before seeing any data
- **P(data | parameters)** → **likelihood** of the data according to our statistical model
- **P(data)** → scaling factor

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
num_heads = np.arange(0, 101, 1)
head_prob = np.arange(0, 1.01, 0.01)

coin = pd.DataFrame([(x, y) for x in num_heads for y in head_prob])
coin.columns = ["num_heads", "head_prob"]
```

```
       num_heads  head_prob
0              0       0.00
1              0       0.01
2              0       0.02
             ...        ...
10199        100       0.99
10200        100       1.00
[10201 rows x 2 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
from scipy.stats import uniform
coin["prior"] = uniform.pdf(coin["head_prob"])
```

```
       num_heads  head_prob
0              0       0.00
1              0       0.01
2              0       0.02
             ...        ...
10199        100       0.99
10200        100       1.00
[10201 rows x 2 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
from scipy.stats import uniform
coin["prior"] = uniform.pdf(coin["head_prob"])
```

```
       num_heads  head_prob  prior
0              0       0.00    1.0
1              0       0.01    1.0
2              0       0.02    1.0
             ...        ...    ...
10199        100       0.99    1.0
10200        100       1.00    1.0
[10201 rows x 3 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
from scipy.stats import uniform
coin["prior"] = uniform.pdf(coin["head_prob"])

from scipy.stats import binom
coin["likelihood"] = binom.pmf(coin["num_heads"], 100, coin["head_prob"])
```

```
       num_heads  head_prob  prior
0              0       0.00    1.0
1              0       0.01    1.0
2              0       0.02    1.0
             ...        ...    ...
10199        100       0.99    1.0
10200        100       1.00    1.0
[10201 rows x 3 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
from scipy.stats import uniform
coin["prior"] = uniform.pdf(coin["head_prob"])

from scipy.stats import binom
coin["likelihood"] = binom.pmf(coin["num_heads"], 100, coin["head_prob"])
```

```
       num_heads  head_prob  prior  likelihood
0              0       0.00    1.0    1.000000
1              0       0.01    1.0    0.366032
2              0       0.02    1.0    0.132620
             ...        ...    ...         ...
10199        100       0.99    1.0    0.366032
10200        100       1.00    1.0    1.000000
[10201 rows x 4 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
coin["posterior_prob"] = coin["prior"] * coin["likelihood"]
coin["posterior_prob"] /= coin["posterior_prob"].sum()
```

```
       num_heads  head_prob  prior  likelihood
0              0       0.00    1.0    1.000000
1              0       0.01    1.0    0.366032
2              0       0.02    1.0    0.132620
             ...        ...    ...         ...
10199        100       0.99    1.0    0.366032
10200        100       1.00    1.0    1.000000
[10201 rows x 4 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
coin["posterior_prob"] = coin["prior"] * coin["likelihood"]
coin["posterior_prob"] /= coin["posterior_prob"].sum()
```

```
       num_heads  head_prob  prior  likelihood  posterior_prob
0              0       0.00    1.0    1.000000        0.009901
1              0       0.01    1.0    0.366032        0.003624
2              0       0.02    1.0    0.132620        0.001313
             ...        ...    ...         ...             ...
10199        100       0.99    1.0    0.366032        0.003624
10200        100       1.00    1.0    1.000000        0.009901
[10201 rows x 5 columns]
```

Tossing the coin again: grid approximation

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
from scipy.stats import binom
from scipy.stats import uniform

num_heads = np.arange(0, 101, 1)
head_prob = np.arange(0, 1.01, 0.01)
coin = pd.DataFrame([(x, y) for x in num_heads for y in head_prob])
coin.columns = ["num_heads", "head_prob"]

coin["prior"] = uniform.pdf(coin["head_prob"])
coin["likelihood"] = binom.pmf(coin["num_heads"], 100, coin["head_prob"])

coin["posterior_prob"] = coin["prior"] * coin["likelihood"]
coin["posterior_prob"] /= coin["posterior_prob"].sum()
```

Plotting posterior distribution

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

```
heads75 = coin.loc[coin["num_heads"] == 75]
heads75["posterior_prob"] /= heads75["posterior_prob"].sum()
```

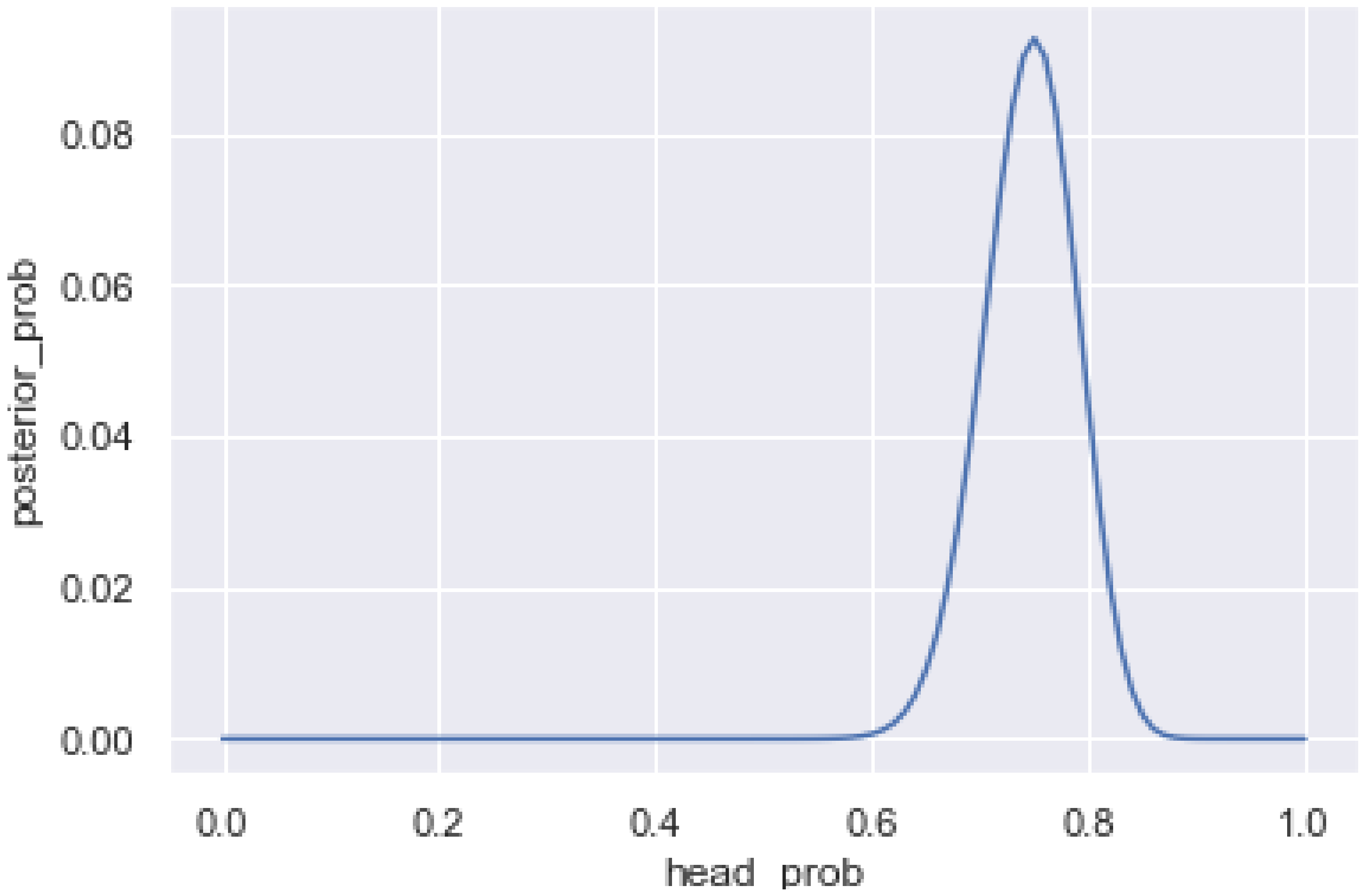
```
      num_heads  head_prob  prior     likelihood  posterior_prob
7575         75       0.00    1.0   0.000000e%2000    0.000000e%2000
7576         75       0.01    1.0  1.886367e-127   1.867690e-129
        ...        ...    ...             ...              ...
7674         75       0.99    1.0   1.141263e-27    1.129964e-29
7675         75       1.00    1.0   0.000000e%2000    0.000000e%2000
[101 rows x 5 columns]
```

```
sns.lineplot(heads75["head_prob"], heads75["posterior_prob"])
plt.show()
```

Plotting posterior distribution

Q: What's the probability of tossing heads with a coin, if we observed 75 heads in 100 tosses?

A:



Let's practice calculating posteriors using grid approximation!

BAYESIAN DATA ANALYSIS IN PYTHON

Prior belief

BAYESIAN DATA ANALYSIS IN PYTHON

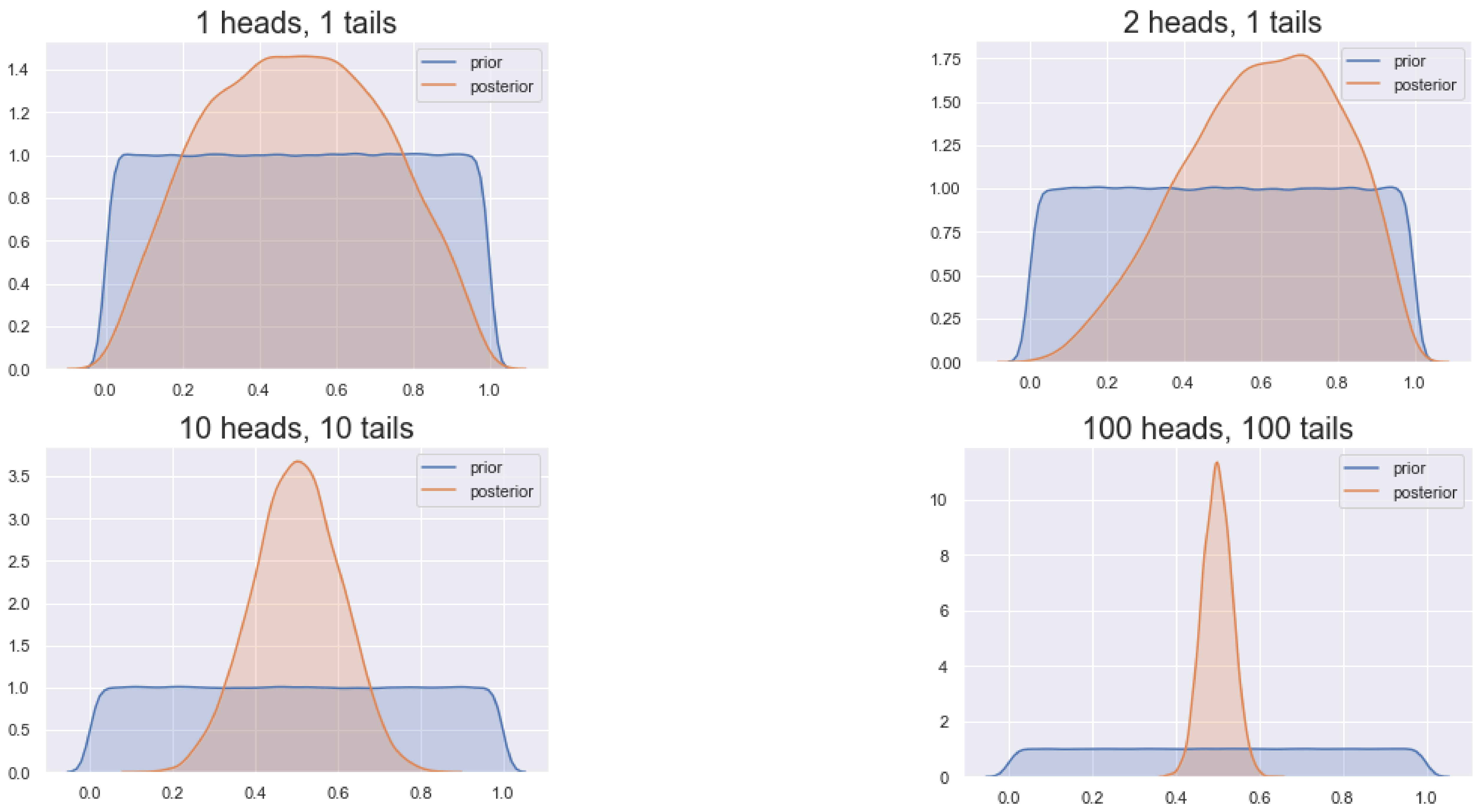
Michal Oleszak
Machine Learning Engineer

Prior distribution

- Prior distribution reflects what we know about the parameter before observing any data:
  - nothing → uniform distribution (all values equally likely)
  - old posterior → can be updated with new data

- One can choose any probability distribution as a prior to include external info in the model:
  - expert opinion
  - common knowledge
  - previous research
  - subjective belief

Prior's impact

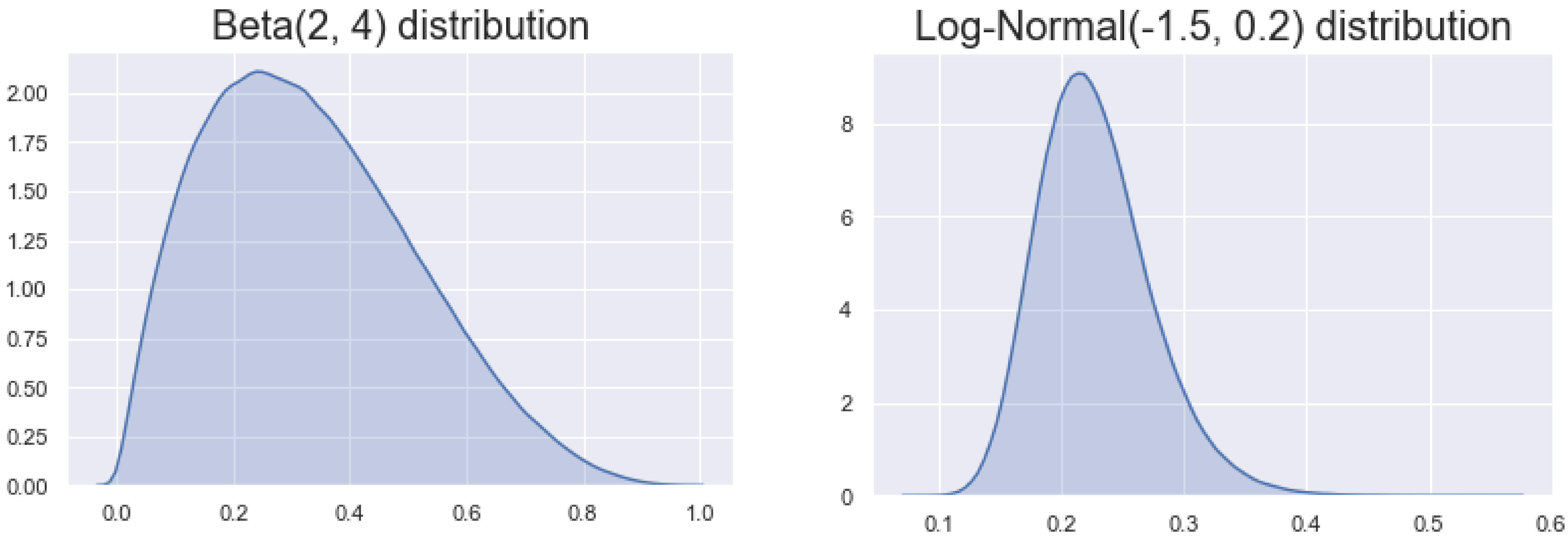


Prior distribution

- Prior distribution chosen before we see the data.
- Prior choice can impact posterior results (especially with little data).
- To avoid cherry-picking, prior choices should be:
  - clearly stated,
  - explainable: based on previous research, sensible assumptions, expert opinion, etc.

Choosing the right prior

Our prior belief: heads less likely



Some choices are better than others!

Conjugate priors

- Some priors, multiplied with specific likelihoods, yield known posteriors.
- They are known as **conjugate priors**.
- In the case of coin tossing:
  - if we choose a prior Beta(a, b),
  - then the posterior is Beta(*#heads* + a, *#tosses* - *#heads* + b)
- We can sample from the posterior using `numpy`.
- `get_heads_prob()` from Chapter 1:

```
def get_heads_prob(tosses):
  num_heads = np.sum(tosses)
  # prior: Beta(1,1)
  return np.random.beta(num_heads + 1, len(tosses) - num_heads + 1, 1000)
```

Two ways to get the posterior

Simulation

- If posterior is known, we can sample from it using `numpy`:

```
draws = np.random.beta(2, 4, 1000)
```

- Outcome: an array of 1000 posterior draws:

```
array([0.05941031, ..., 0.70015975])
```

- Can be plotted with

```
sns.kdeplot(draws)
```

Calculation

- If posterior is not known, we can calculate it using grid approximation.
- Outcome: posterior probability for each grid element:

```
       head_prob  posterior_prob
0           0.00        0.009901
1           0.01        0.003624
             ...             ...
10199       0.99        0.003624
10200       1.00        0.009901
```

- Can be plotted with

```
sns.lineplot(df["head_prob"], df["posterior_prob"])
```

Let's practice working with priors!

BAYESIAN DATA ANALYSIS IN PYTHON

Reporting Bayesian results

BAYESIAN DATA ANALYSIS IN PYTHON

Michal Oleszak
Machine Learning Engineer

The honest way

- Report the prior and the posterior of each parameter

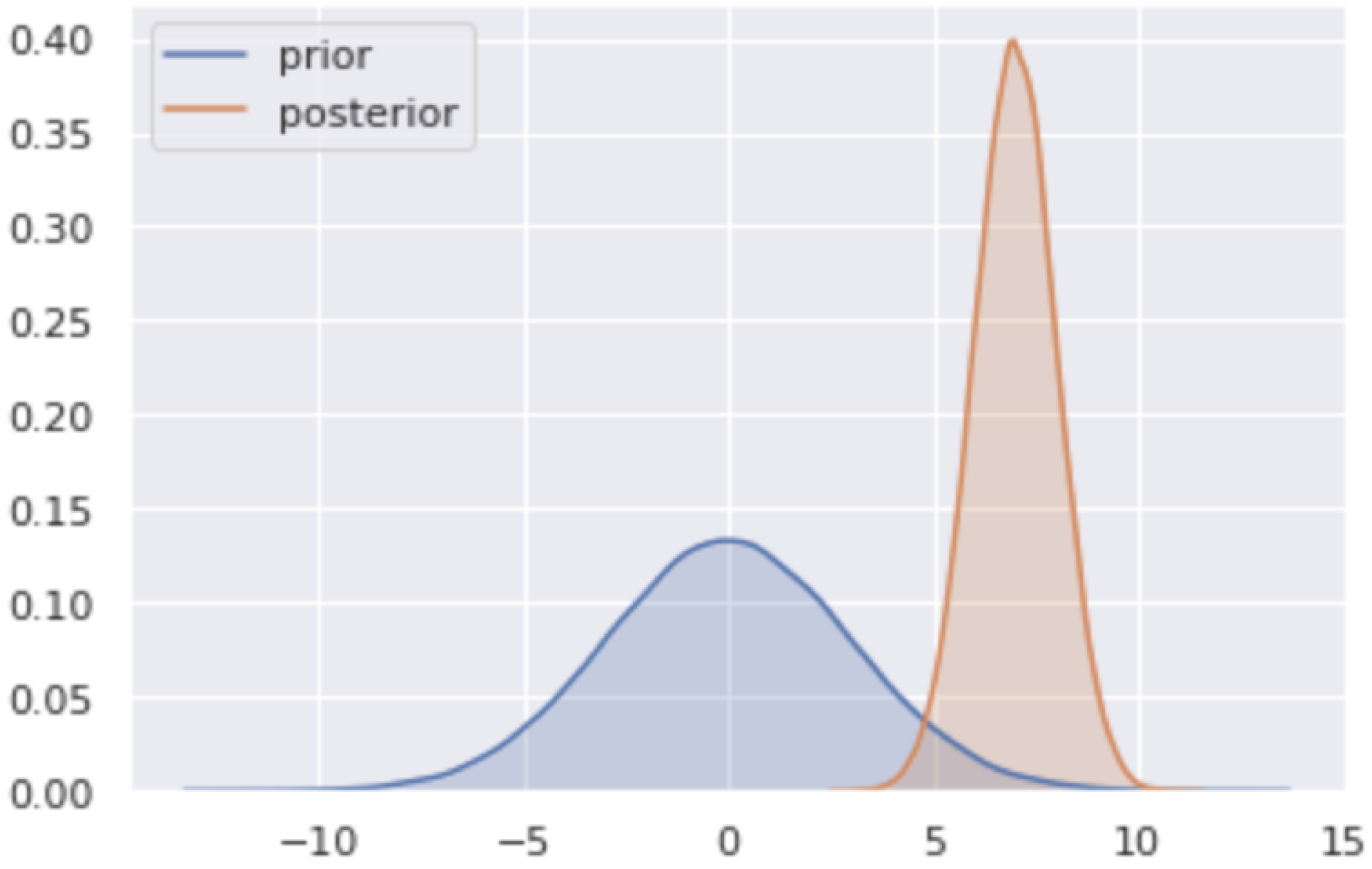
```
posterior_draws
```

```
array([8.02800413, 8.97359548, 7.57437476, ..., 5.85264609, 7.92875104,
       7.41463758])
```

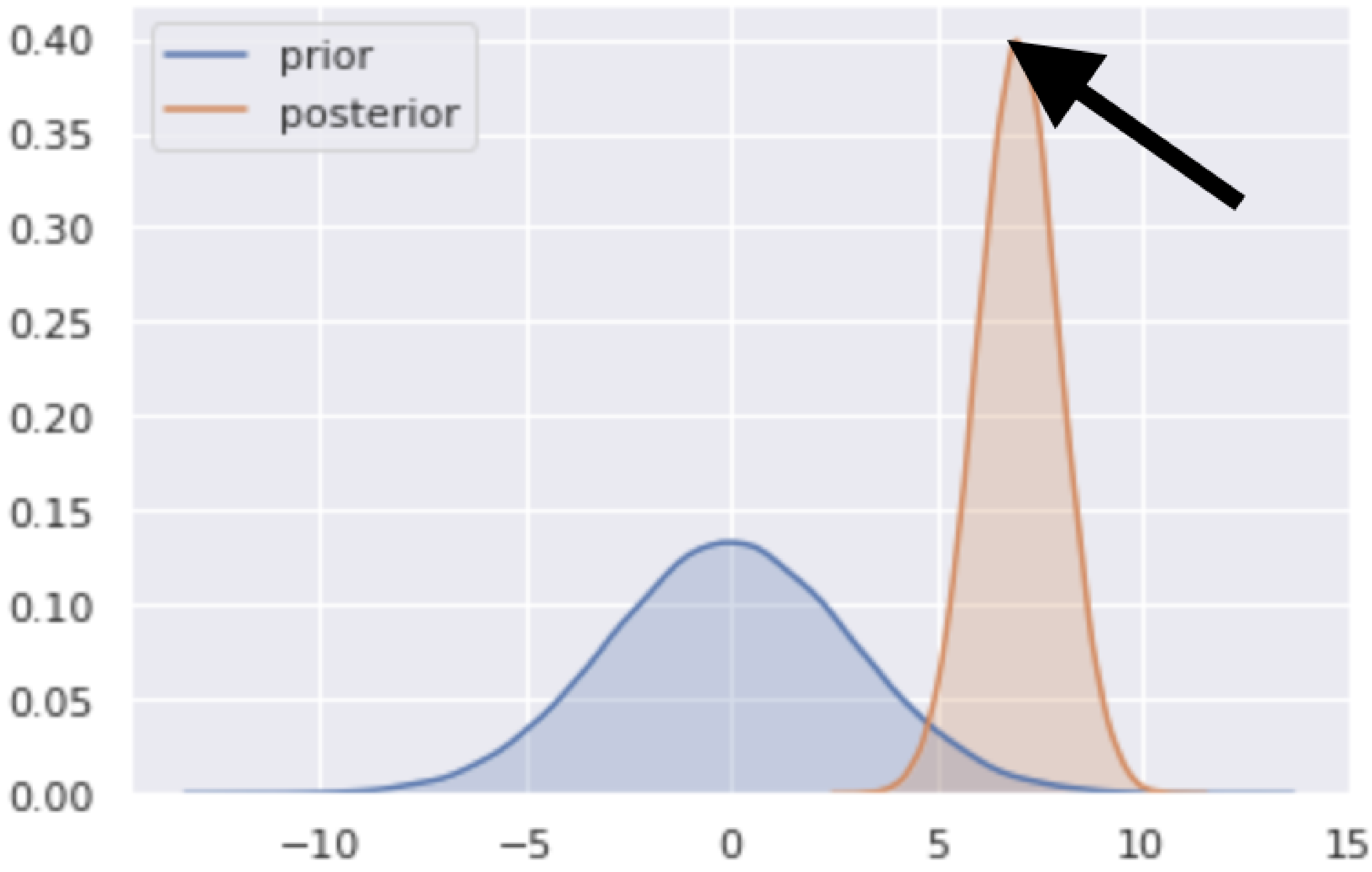
- Plot prior and posterior distributions

```
sns.kdeplot(prior_draws, shade=True, label="prior")
sns.kdeplot(posterior_draws, shade=True, label="posterior")
```

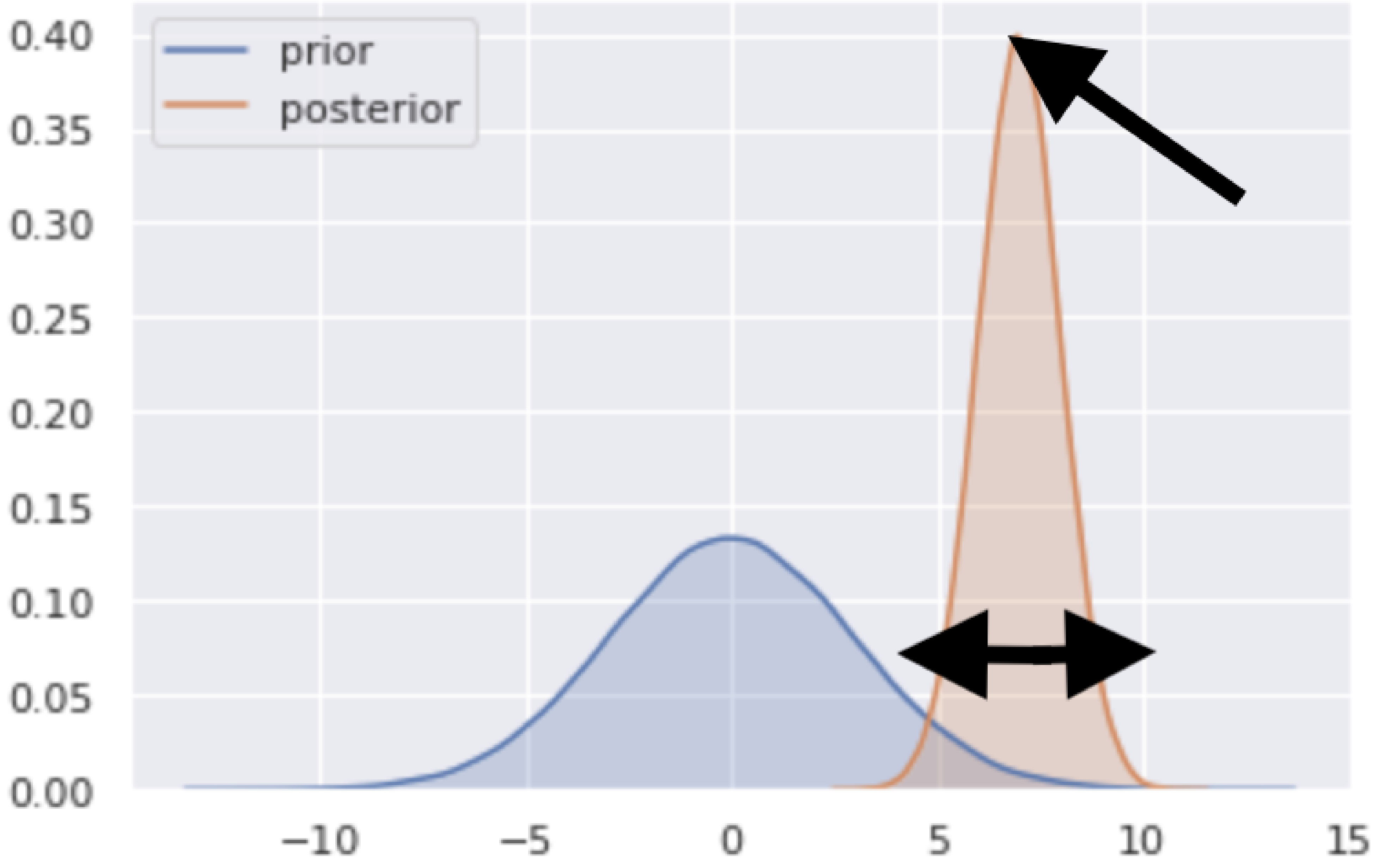
The honest way



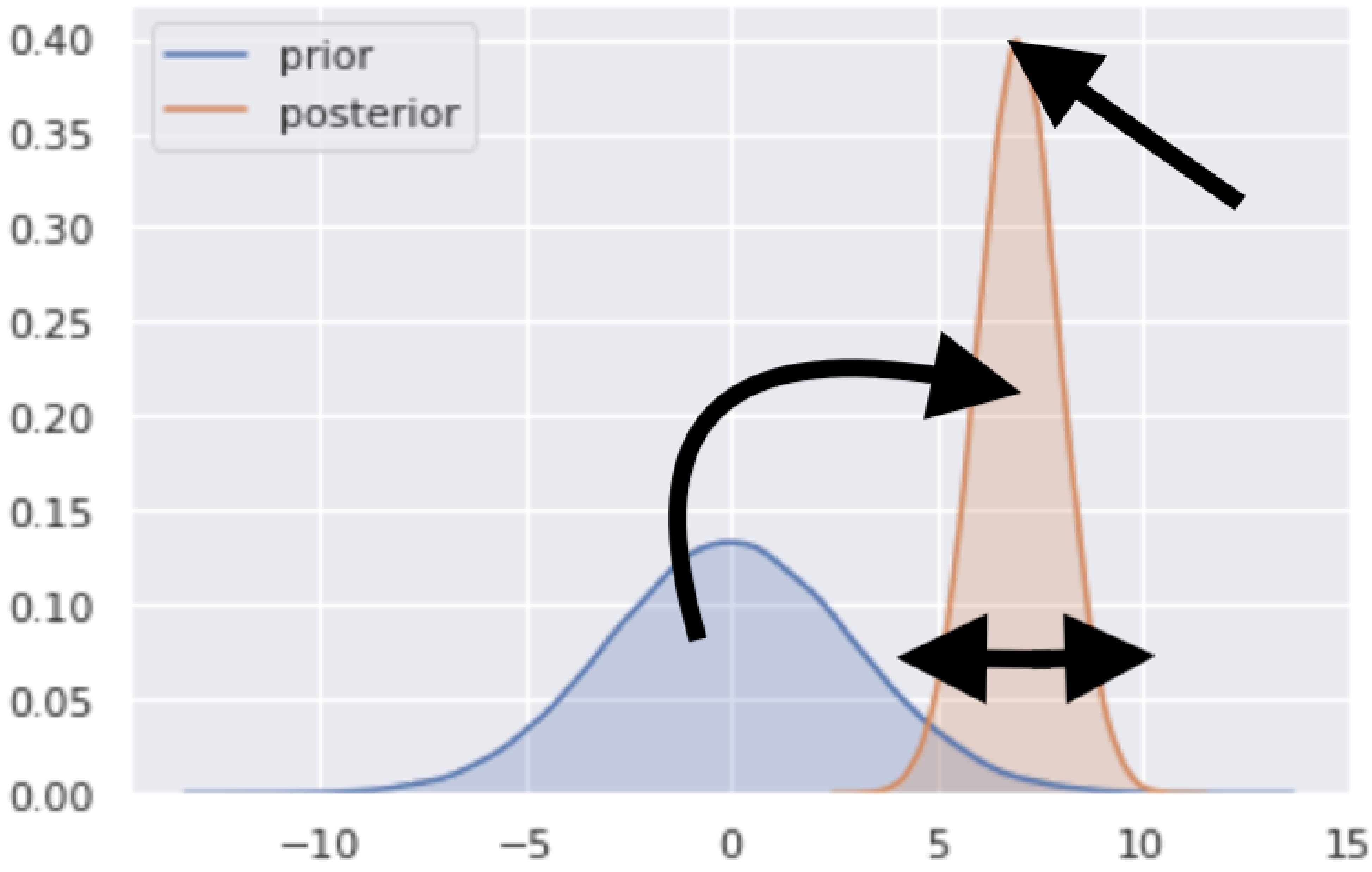
The honest way



The honest way

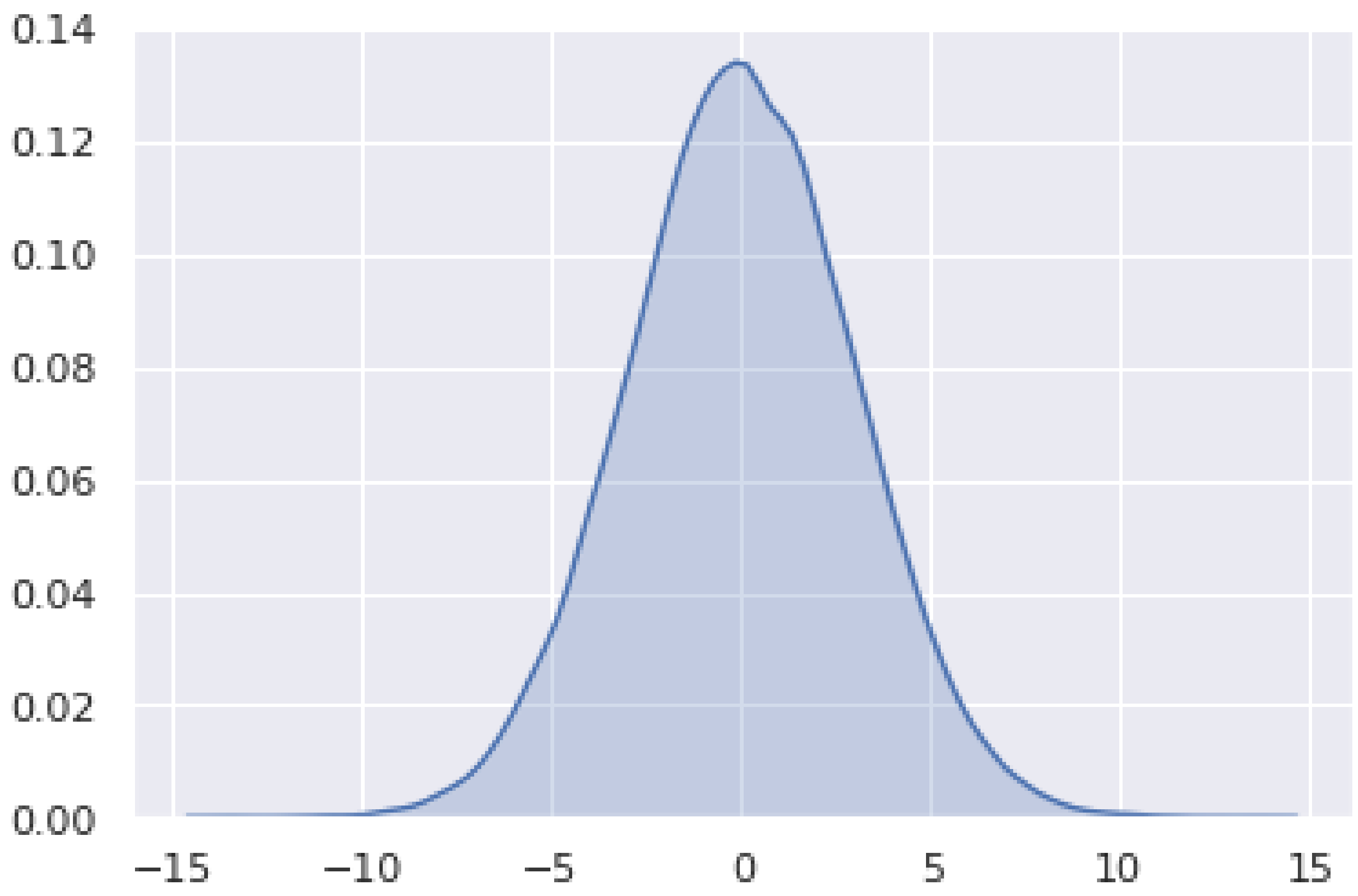


The honest way



Bayesian point estimates

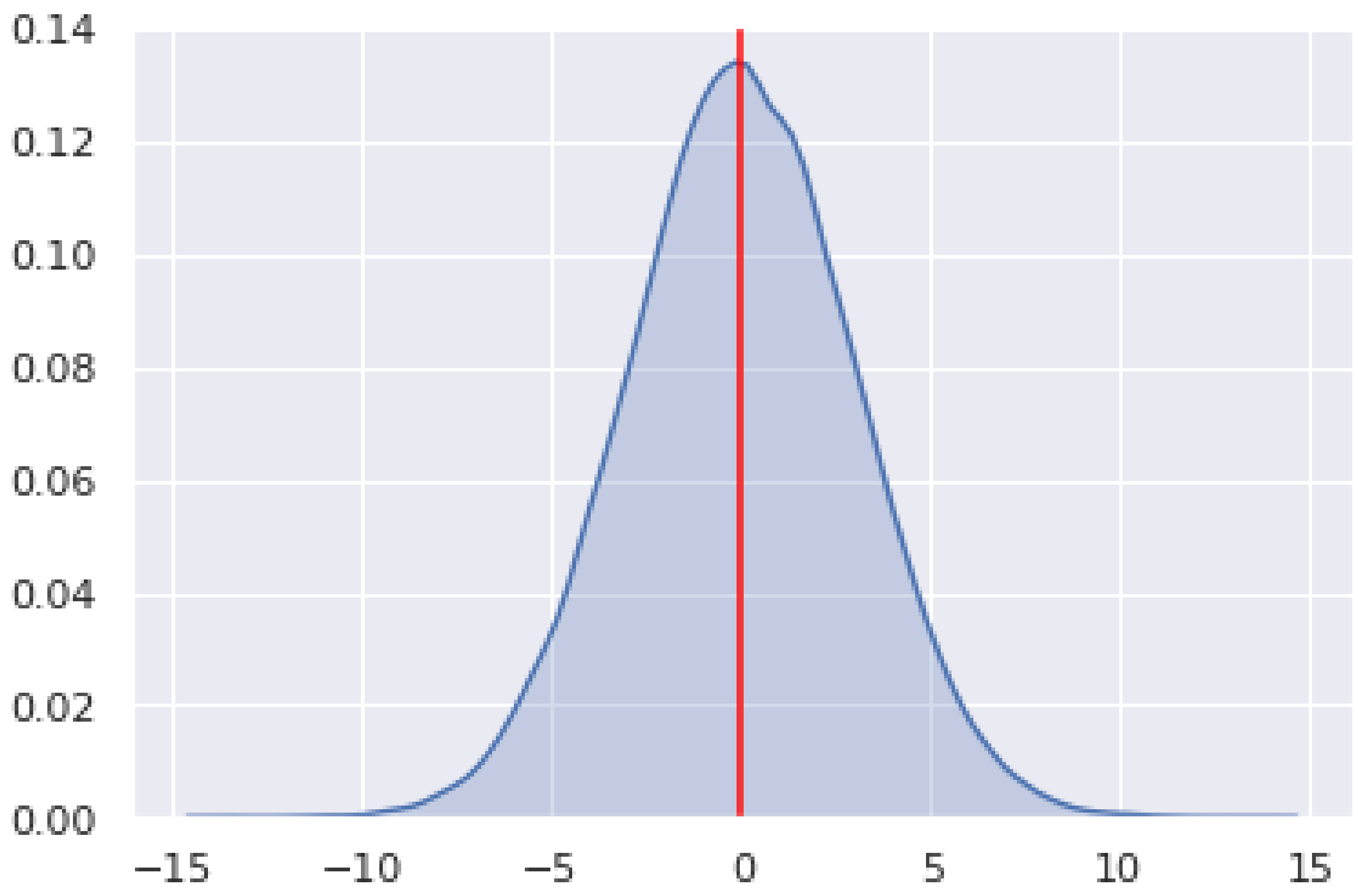
- No single number can fully convey the complete information contained in a distribution
- However, sometimes a point estimate of a parameter is needed



Bayesian point estimates

- No single number can fully convey the complete information contained in a distribution
- However, sometimes a point estimate of a parameter is needed

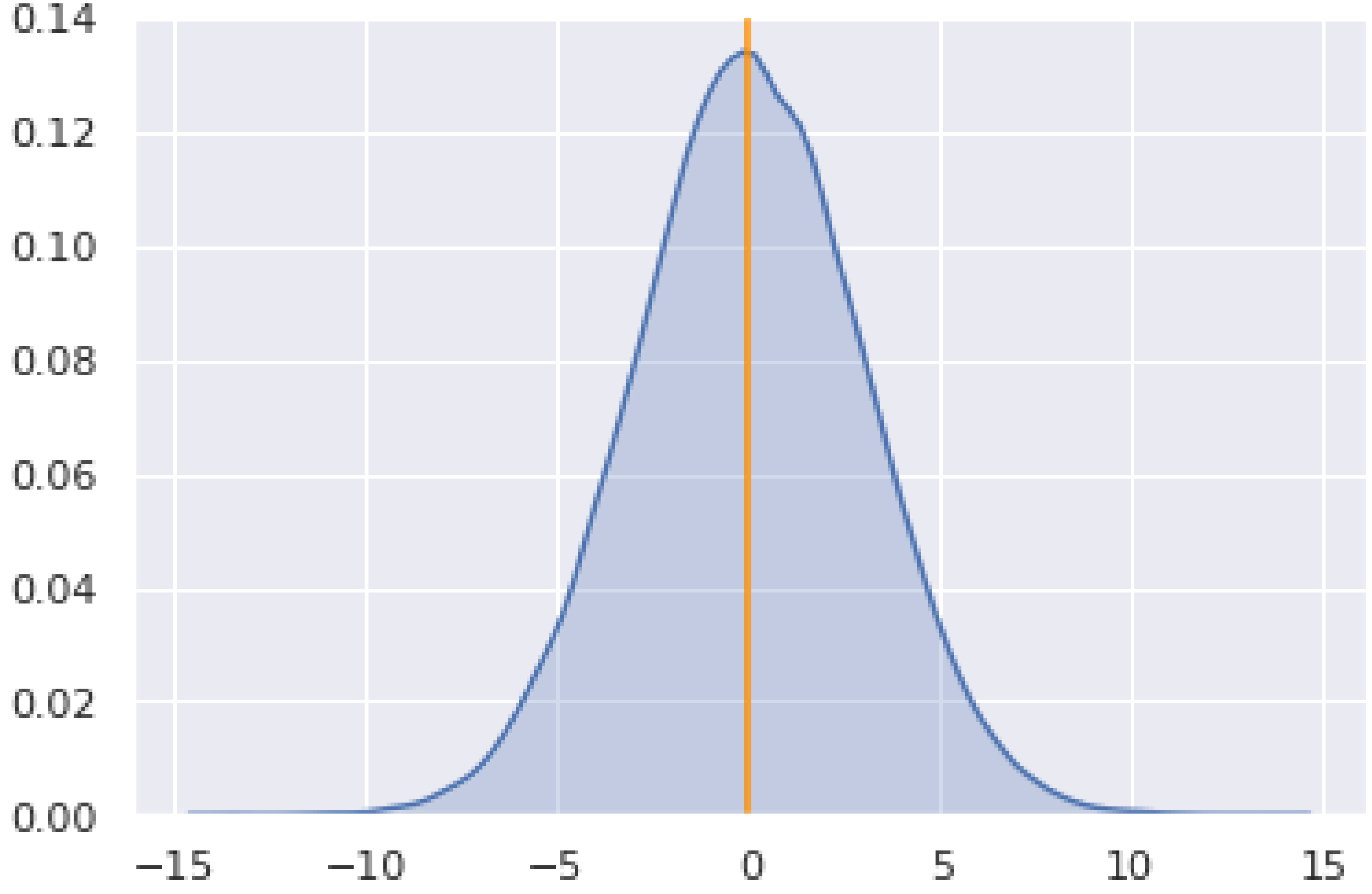
```
posterior_mean = np.mean(posterior_draws)
```



Bayesian point estimates

- No single number can fully convey the complete information contained in a distribution
- However, sometimes a point estimate of a parameter is needed

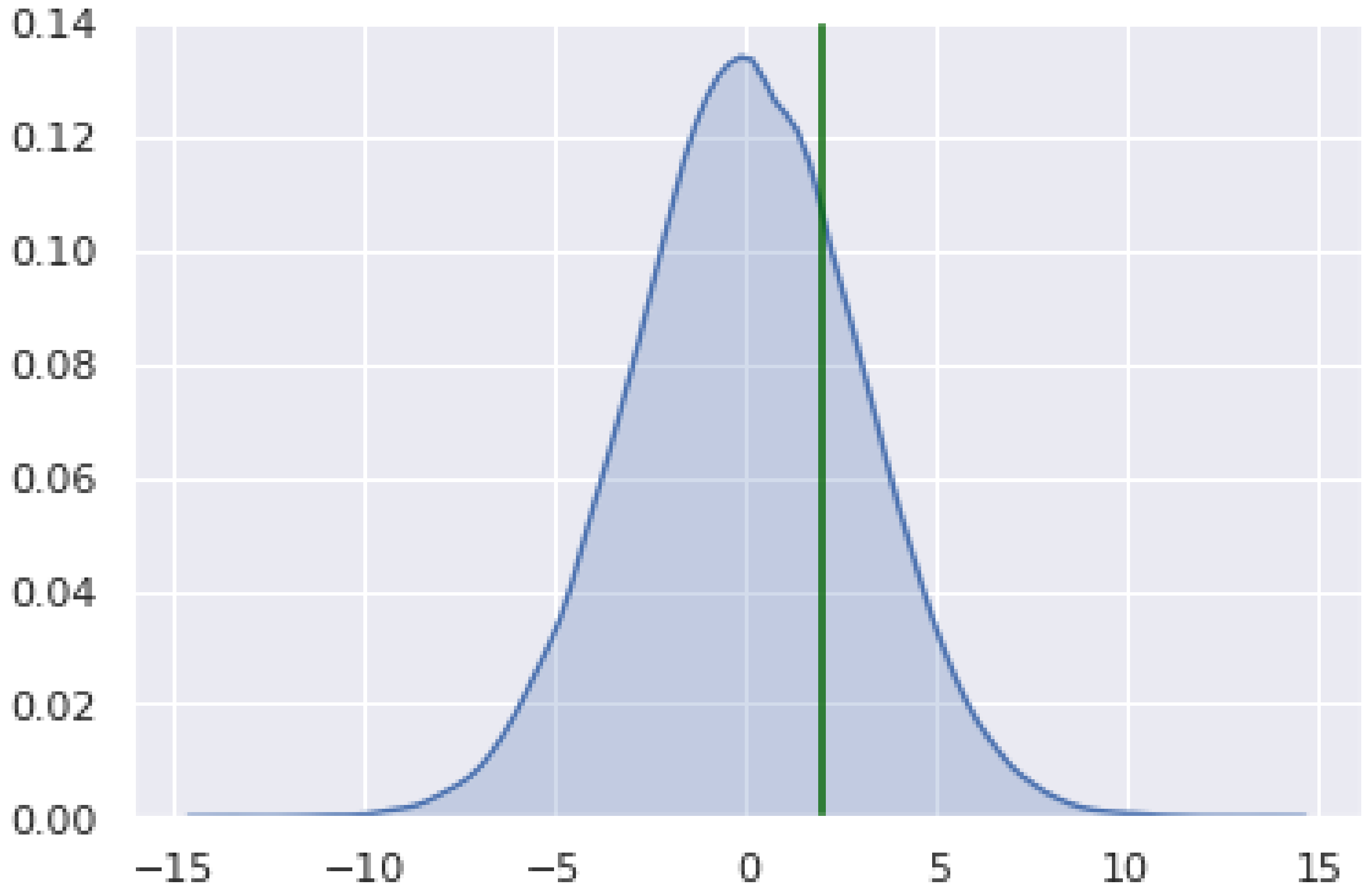
```
posterior_mean = np.mean(posterior_draws)
posterior_median = np.median(posterior_draws)
```



Bayesian point estimates

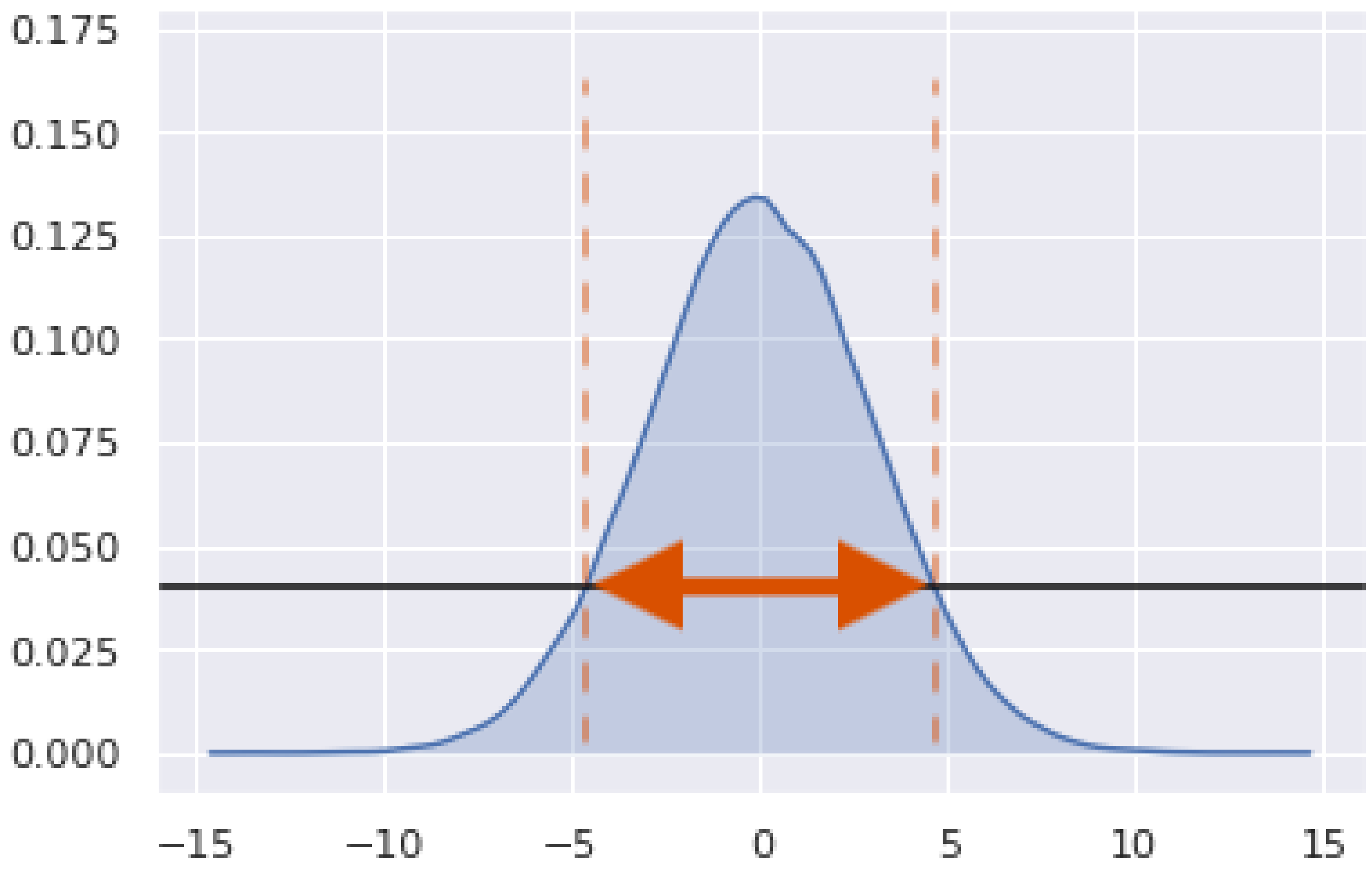
- No single number can fully convey the complete information contained in a distribution
- However, sometimes a point estimate of a parameter is needed

```
posterior_mean = np.mean(posterior_draws)
posterior_median = np.median(posterior_draws)
posterior_p75 = np.percentile(posterior_draws, 75)
```

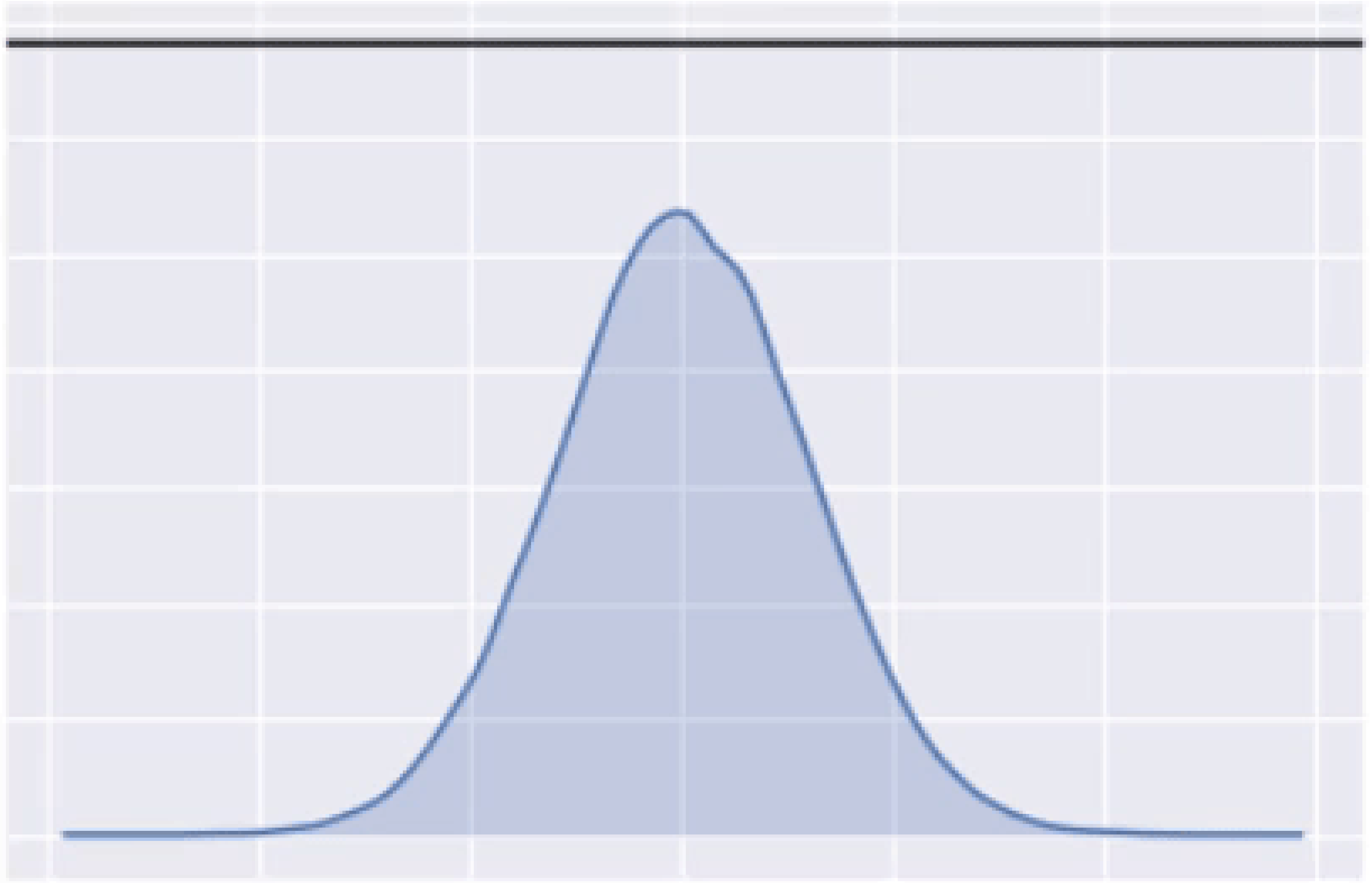


Credible intervals

- Such an interval that the probability that the parameter falls inside it is x%
- The wider the credible interval, the more uncertainty in parameter estimate
- Parameter is random, so it can fall into an interval with some probability
- In the frequentist world, the (confidence) interval is random while the parameter is fixed



Highest Posterior Density (HPD)



```python
import pymc3 as pm


hpd = pm.hpd(posterior_draws,
             hdi_prob=0.9)
print(hpd)
```

```
[-4.86840193  4.96075498]
```

Let's practice reporting Bayesian results!

BAYESIAN DATA ANALYSIS IN PYTHON