Data 102 Lecture 7 Demo

In [1]:
```
import numpy as np
import pandas as pd
from scipy import stats
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

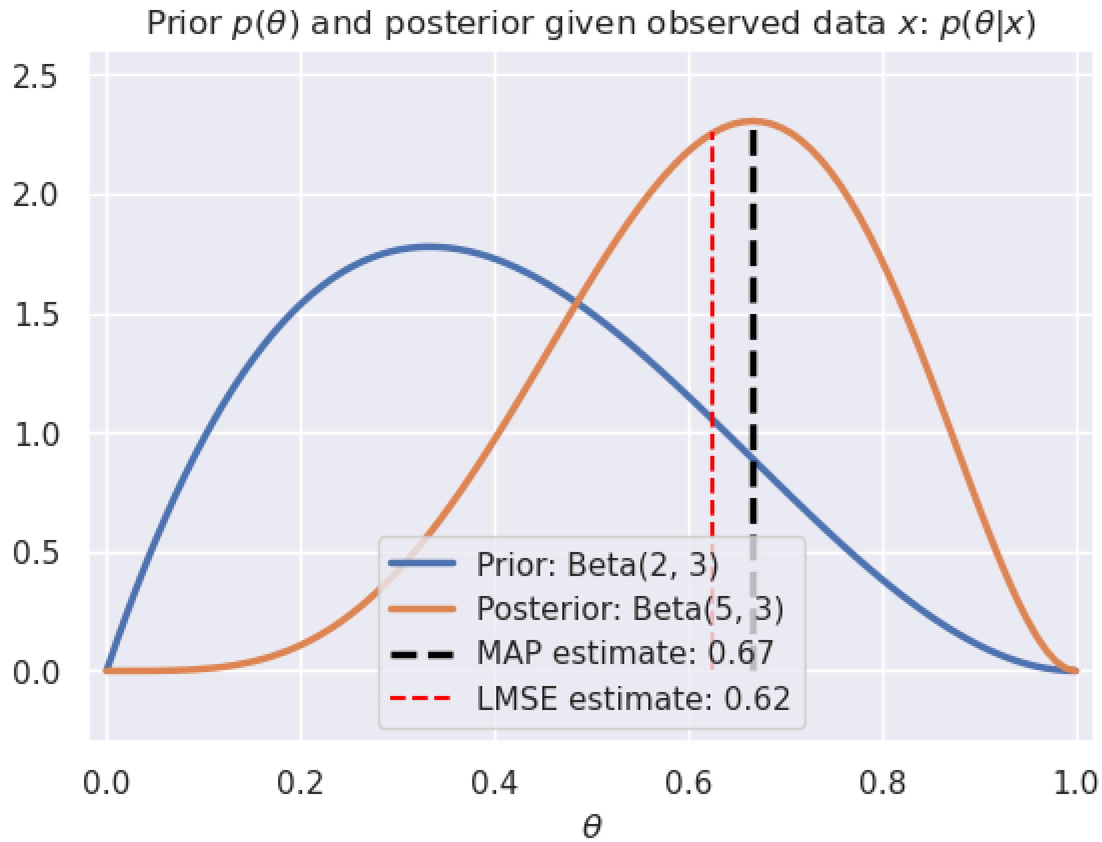
These are copied from last lecture:

In [2]:
```
FIGURE_SIZE = (6.5, 4.5)
def plot_prior_posterior(x, prior, posterior, xlim,
                         prior_label, posterior_label,
                         x_map=None, x_lmse=None,
                        param_name: str = r'\theta'):

    plt.figure(figsize=FIGURE_SIZE, dpi=100)
    plt.plot(x, prior, lw=2.5, label = prior_label)
    plt.plot(x, posterior, lw=2.5, label = posterior_label)
    if x_map is not None:
        map_index = np.argmin(np.abs(x - x_map))
        posterior_map = posterior[map_index]
        label = f'MAP estimate: {x_map:0.2f}'
        plt.plot([x_map, x_map], [0, posterior_map], '--', lw=2.5, color='black', label=
    if x_lmse is not None:
        lmse_index = np.argmin(np.abs(x - x_lmse))
        posterior_lmse = posterior[lmse_index]
        label = f'LMSE estimate: {x_lmse:0.2f}'
        plt.plot([x_lmse, x_lmse], [0, posterior_lmse], '--', lw=1.5, color='red', label
    #plt.legend(bbox_to_anchor=(1.32, 1.02))
    plt.legend()
    ymax = max(max(prior[np.isfinite(prior)]), max(posterior[np.isfinite(posterior)]))
    plt.ylim(-0.3, ymax+0.3)
    plt.xlim(*xlim)
    plt.xlabel(f'${param_name}$')
    plt.title(
        f'Prior $p({param_name})$ and posterior given observed data $x$: $p({param_name}
    );


def plot_beta_prior_and_posterior(alpha, beta, pos_obs, neg_obs, show_map=False, show_lm
    x = np.linspace(0, 1, 100)
    prior = stats.beta.pdf(x, alpha, beta)

    alpha_new = alpha + pos_obs
    beta_new = beta + neg_obs
    posterior = stats.beta.pdf(x, alpha_new, beta_new)

    # You never have to memorize these: when making this notebook,
    # I found them on the wikipedia page for the Beta distribution:
    # https://en.wikipedia.org/wiki/Beta_distribution

    if show_lmse:
        x_lmse = (alpha_new)/(alpha_new + beta_new)
    else:
```

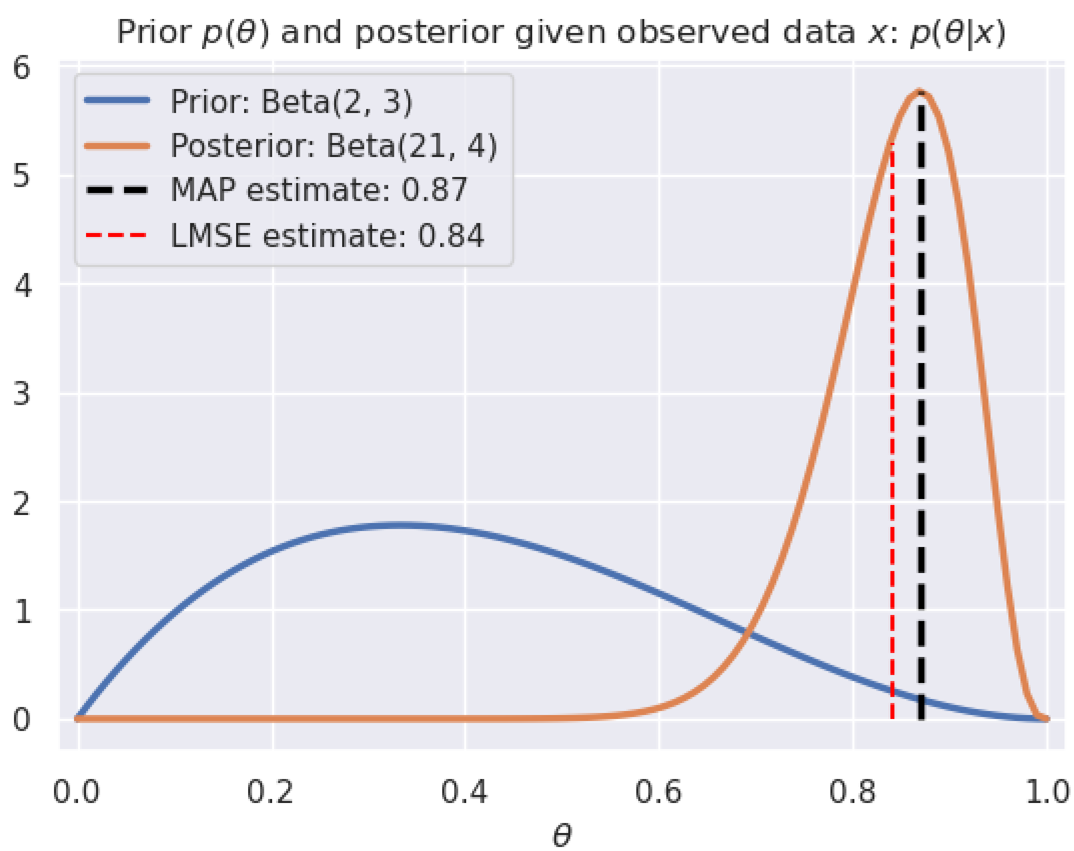
```python
        x_lmse = None

    if show_map:
        x_map = (alpha_new - 1) / (alpha_new + beta_new - 2)
    else:
        x_map = None
    plot_prior_posterior(x, prior, posterior, (-0.02, 1.02),
                         prior_label=f'Prior: Beta({alpha}, {beta})',
                         posterior_label=f'Posterior: Beta({alpha_new}, {beta_new})',
                         x_map=x_map, x_lmse=x_lmse)

# You don't need to understand how this function is implemented.

def plot_gaussian_prior_and_posterior(μ_0, σ_0, xs, σ, range_in_σs=3.5, show_map=False,
    """
    Plots prior and posterior Normaly distribution

    Args:
        μ_0, σ_0: parameters (mean, SD) of the prior distribution
        xs: list or array of observations
        σ: SD of the likelihood
        range_in_σs: how many SDs away from the mean to show on the plot
        show_map: whether or not to show the MAP estimate as a vertical line
        show_lmse: whether or not to show the LMSE/MMSE estimate as a vertical line
    """
    n = len(xs)
    posterior_σ = 1/np.sqrt(1/(σ_0**2) + n/(σ**2))
    posterior_mean = (posterior_σ**2) * (μ_0/(σ_0**2) + np.sum(xs)/(σ**2))

    # Compute range for plot
    posterior_min = posterior_mean - (range_in_σs * posterior_σ)
    posterior_max = posterior_mean + (range_in_σs * posterior_σ)
    prior_min = μ_0 - (range_in_σs * σ)
    prior_max = μ_0 + (range_in_σs * σ)

    xmin = min(posterior_min, prior_min)
    xmax = max(posterior_max, prior_max)
    x = np.linspace(xmin, xmax, 100)
    if show_lmse:
        x_lmse = posterior_mean
    else:
        x_lmse = None

    if show_map:
        x_map = posterior_mean
    else:
        x_map = None


    prior = stats.norm.pdf(x, μ_0, σ_0)
    posterior = stats.norm.pdf(x, posterior_mean, posterior_σ)


    plot_prior_posterior(x, prior, posterior, (xmin, xmax), 'Prior', 'Posterior',
                         x_map=x_map, x_lmse=x_lmse, param_name=r'\mu')
```

Let's look at the posterior and compare the two microwaves that way:

In [3]:
```python
# Microwave A: 3 positive reviews, 0 negative reviews
plot_beta_prior_and_posterior(2, 3, 3, 0, show_map=True, show_lmse=True)
```



In [4]:
```
# Microwave B: 19 positive reviews, 1 negative reviews
plot_beta_prior_and_posterior(2, 3, 19, 1, show_map=True, show_lmse=True)
```

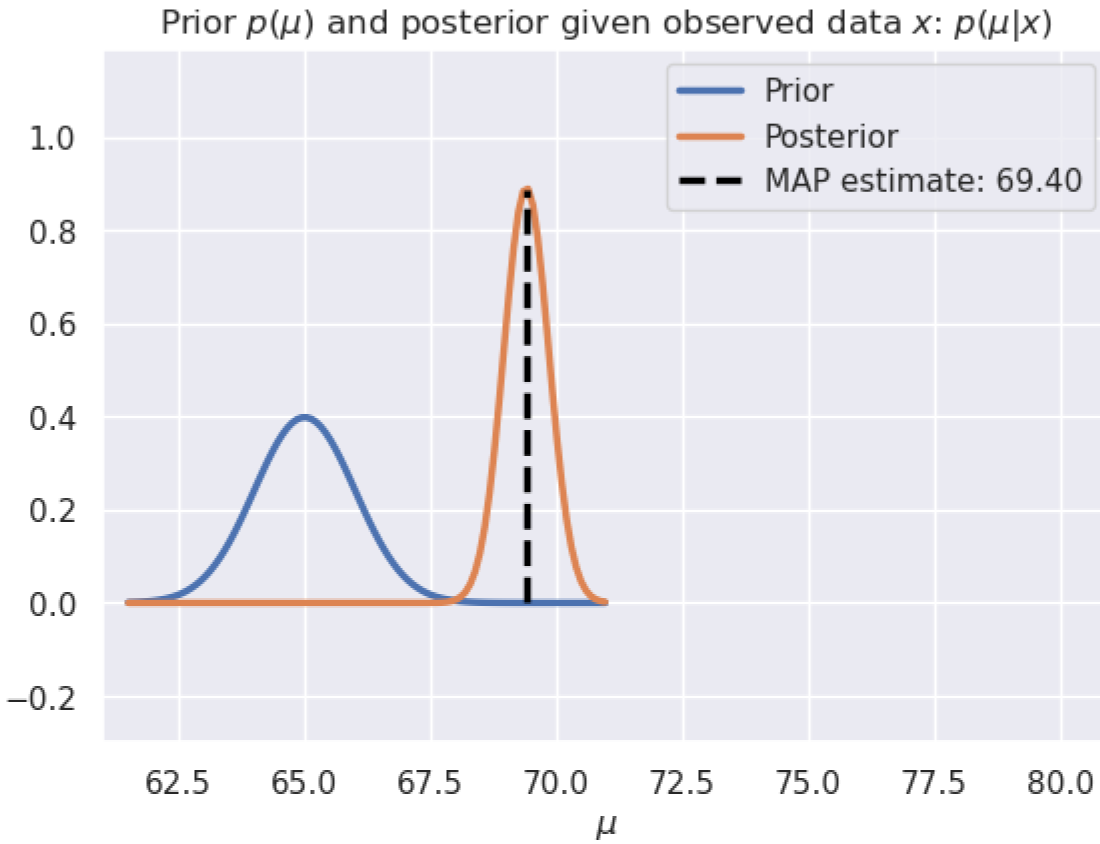


Continuous example: heights

In [5]:
```
observed_heights = [6*12+0, 6*12+1, 5*12+9, 5*12+8]
```

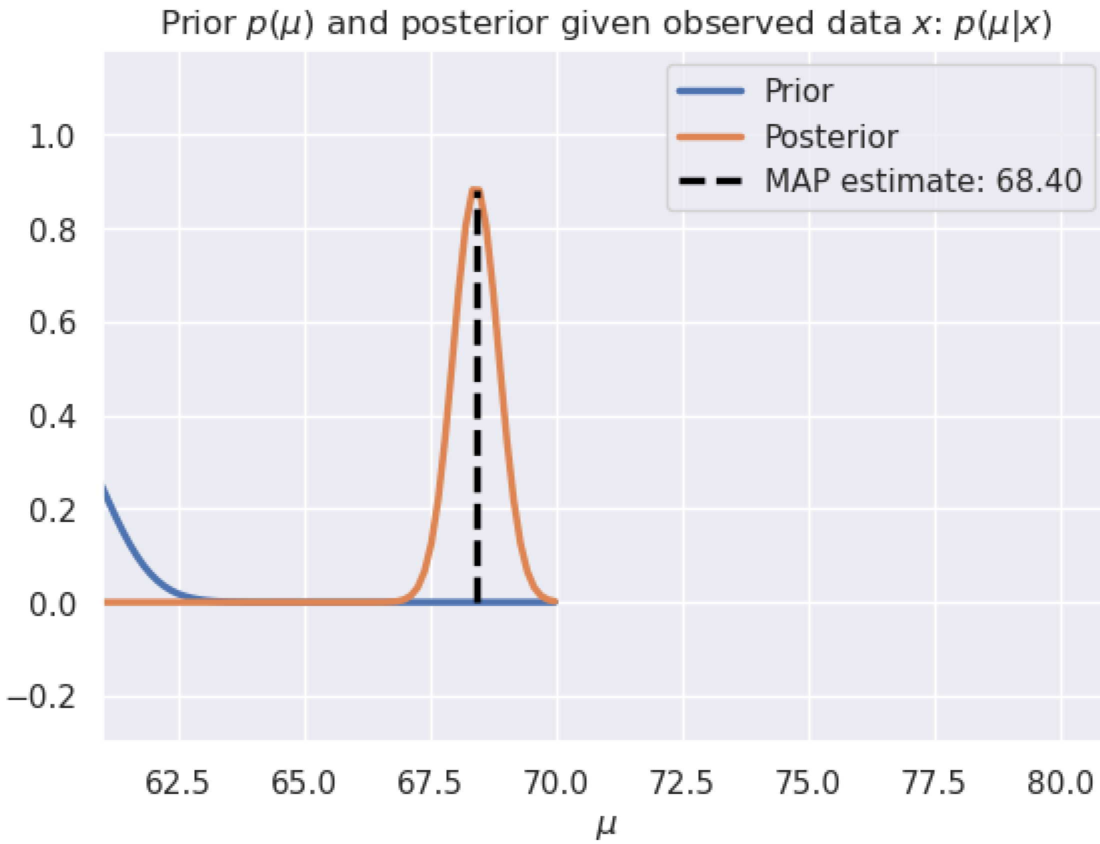
In [6]:
```
plot_gaussian_prior_and_posterior(5*12 + 5, 1, observed_heights, 1, show_map=True)
plt.xlim([61, 81])
plt.xlabel(r'$\mu$')
```

Out[6]: Text(0.5, 0, '$\\mu$')



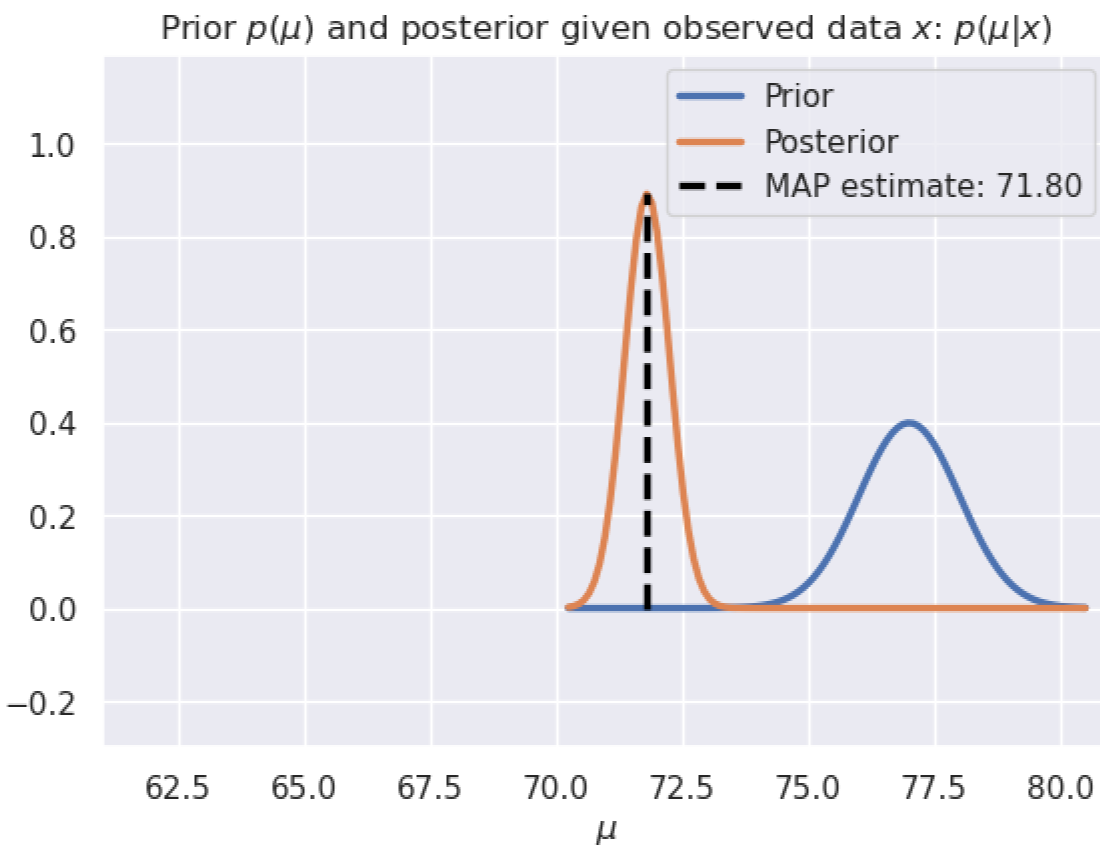
In [7]:
```
plot_gaussian_prior_and_posterior(5*12 + 0, 1, observed_heights, 1, show_map=True)
plt.xlim([61, 81])
plt.xlabel(r'$\mu$')
```

Out[7]: Text(0.5, 0, '$\\mu$')



In [8]:
```
plot_gaussian_prior_and_posterior(6*12 + 5, 1, observed_heights, 1, show_map=True)
plt.xlim([61, 81])
plt.xlabel(r'$\mu$')
```

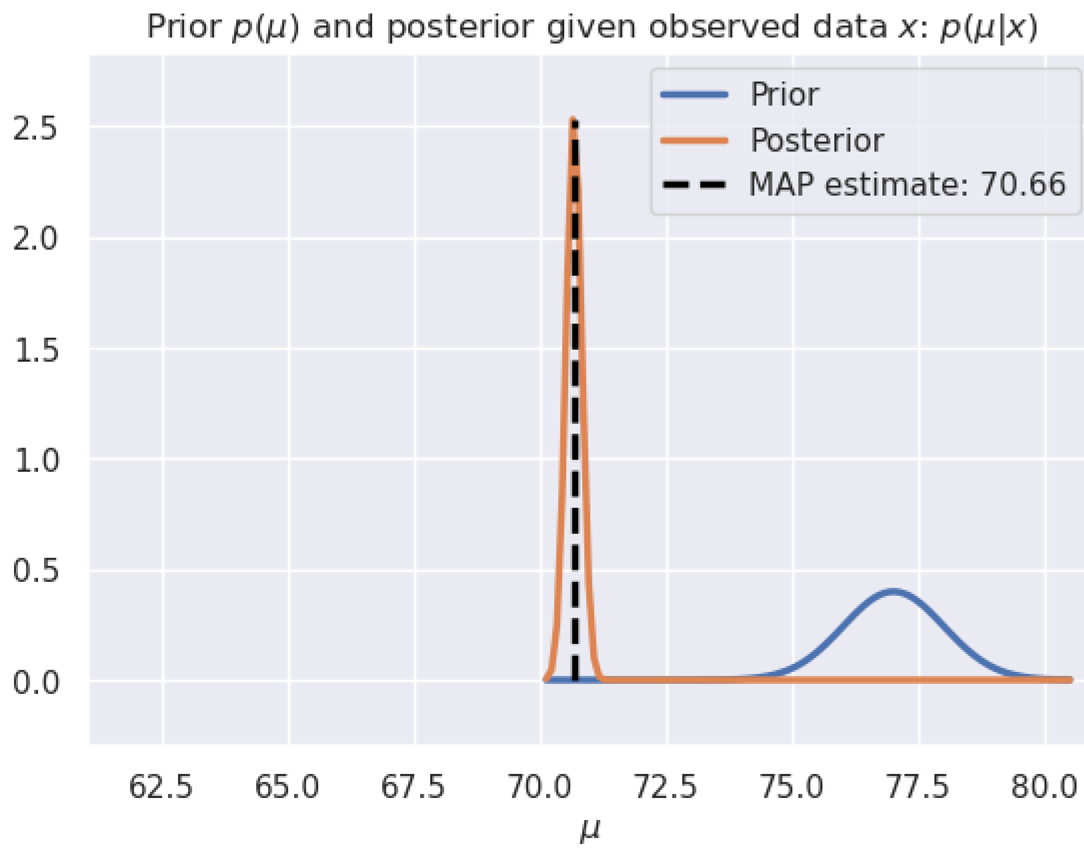
Out[8]:
```
Text(0.5, 0, '$\\mu$')
```



In [9]:
```
plot_gaussian_prior_and_posterior(6*12 + 5, 1, observed_heights * 10, 1, show_map=True)
plt.xlim([61, 81])
```

```
plt.xlabel(r'$\mu$')
```

Out[9]: Text(0.5, 0, '$\\mu$')



A more complex model: exoplanet sizes

Next, we'll look at a dataset of exoplanets: planets outside of our solar system. The `planets` dataframe contains information about 517 exoplanets from NASA. Can we use it to estimate which planets might be able to support life?

In [10]:

```
planets = pd.read_csv('exoplanets.csv')
planets.head()
```

Out[10]:

	name	orbital_period	mass	radius	star_temperature	density
0	2MASS J21402931+1625183 A b	7336.500000	6657.910000	10.312188	2300.0	NaN
1	55 Cnc e	0.736539	8.078476	1.905513	5196.0	6.40
2	BD+20 594 b	41.685500	16.299962	2.230571	5766.0	7.89
3	CoRoT-1 b	1.508956	327.334000	16.701261	5950.0	0.38
4	CoRoT-10 b	13.240600	873.950000	10.872633	5075.0	3.70

It contains the following columns.

- `name` : the name of the exoplanet
- `orbital_period` : how many days it takes for the planet to orbit its star
- `mass` : the mass of the planet, in multiples of Earth's mass (e.g., the second planet, 55 Cnc e, has a mass 73.6% of Earth's)

- `radius` : the radius of the planet, in multiples of Earth's radius (e.g., the second planet, 55 Cnc e, has a radius almost twice the size of Earth's)
- `star_temperature` : the temperature of the star that the planet orbits, in Kelvin
- `density` : the density of the planet, in g/cm^3

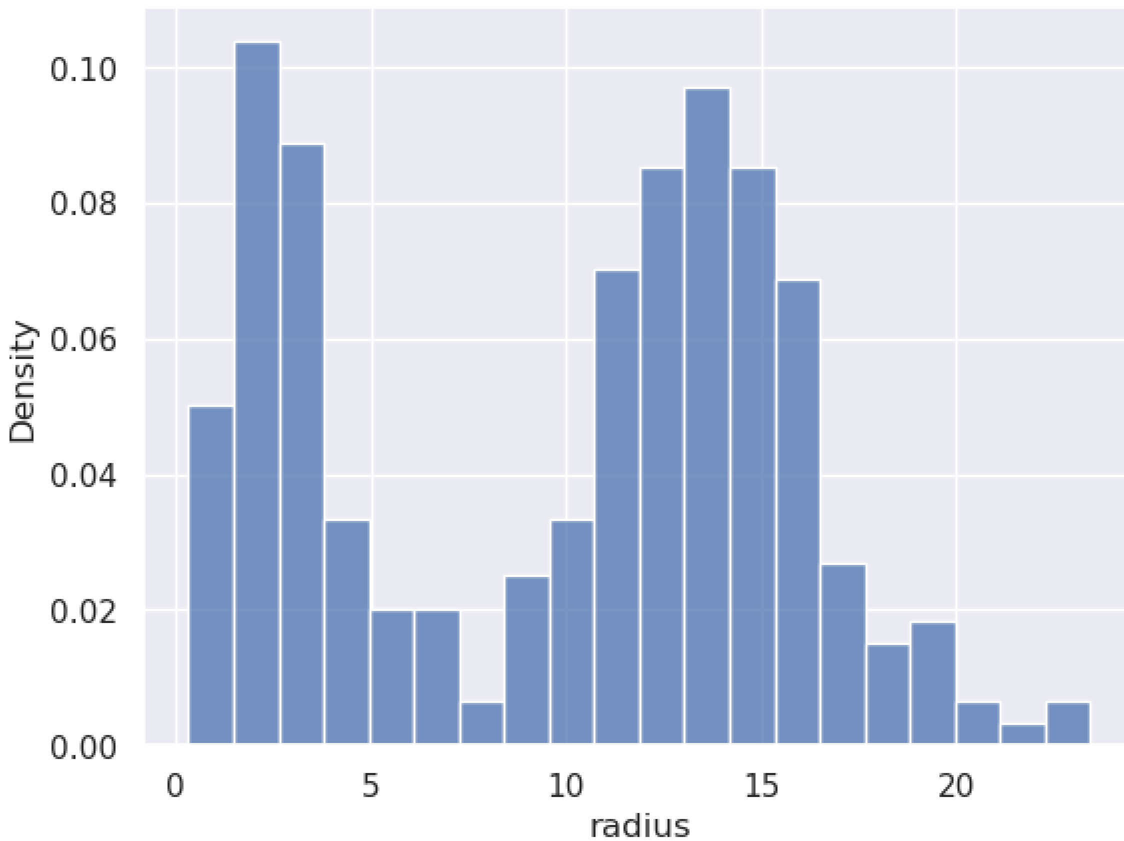
Note that in astronomy, it's more common to measure mass and radius in terms of the planet Jupiter rather than Earth (Jupiter is about 11 times the radius of earth and about 317 times the mass), but we're using Earth-based measurements since we're going to use Earth as a standard for habitability.

In [11]:
```
planets.shape
```

Out[11]:
```
(517, 6)
```

In [12]:
```
sns.histplot(data=planets, x='radius', stat='density', bins=20)
```

Out[12]:
```
<AxesSubplot:xlabel='radius', ylabel='Density'>
```

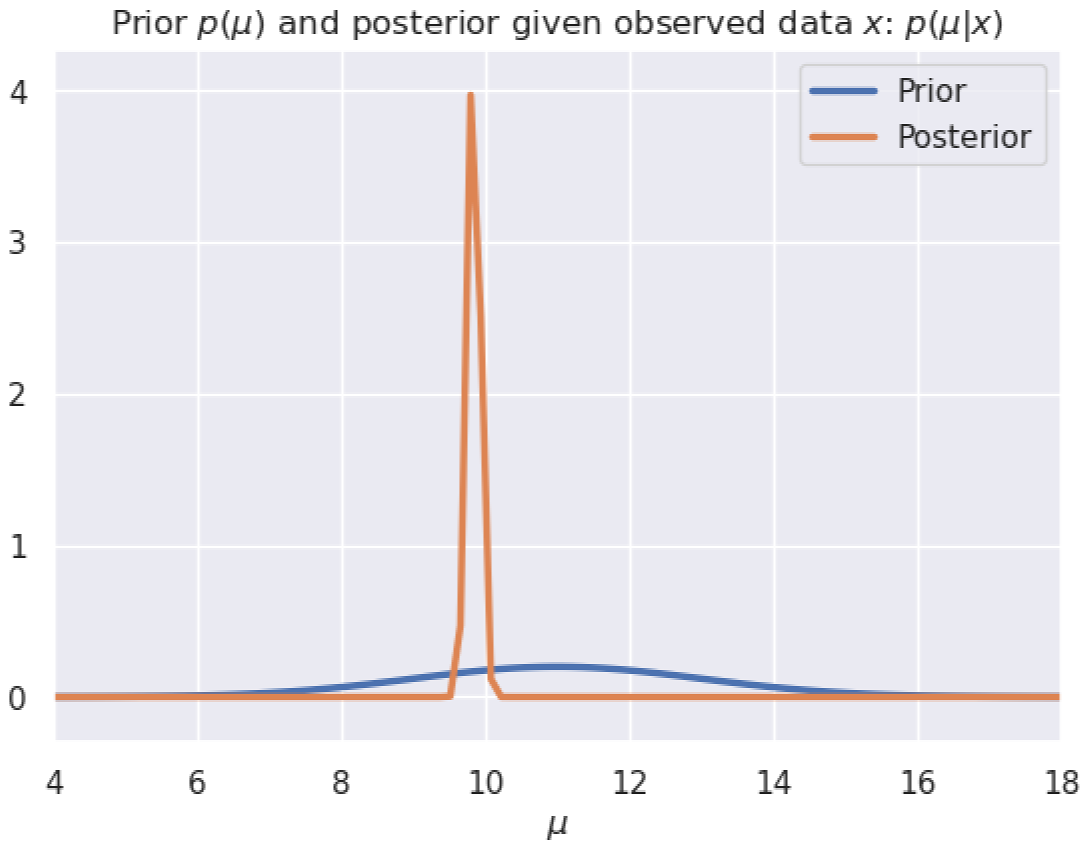


In [13]:
```
planets['radius'].mean()
```

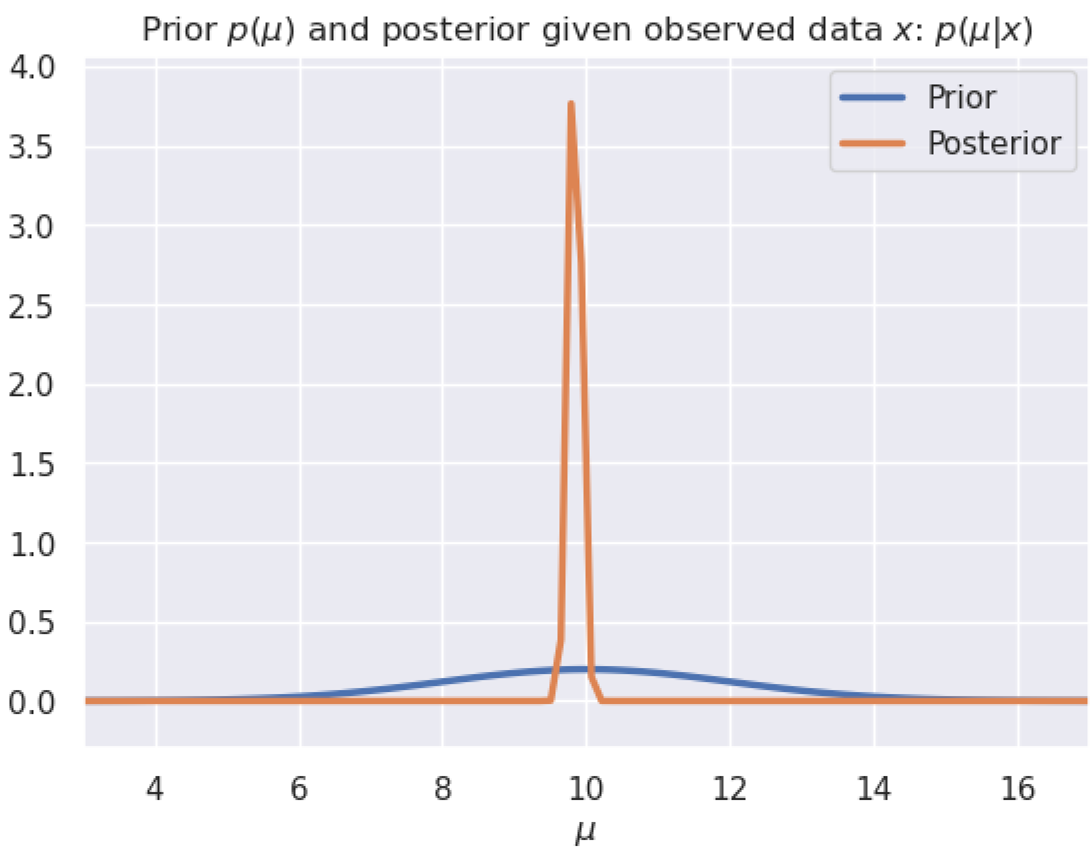
Out[13]:
```
9.84137083868472
```

Gaussian likelihood is not a good fit: the observations have two modes.

In [14]:
```
plot_gaussian_prior_and_posterior(11, 2, planets['radius'], 2)
```

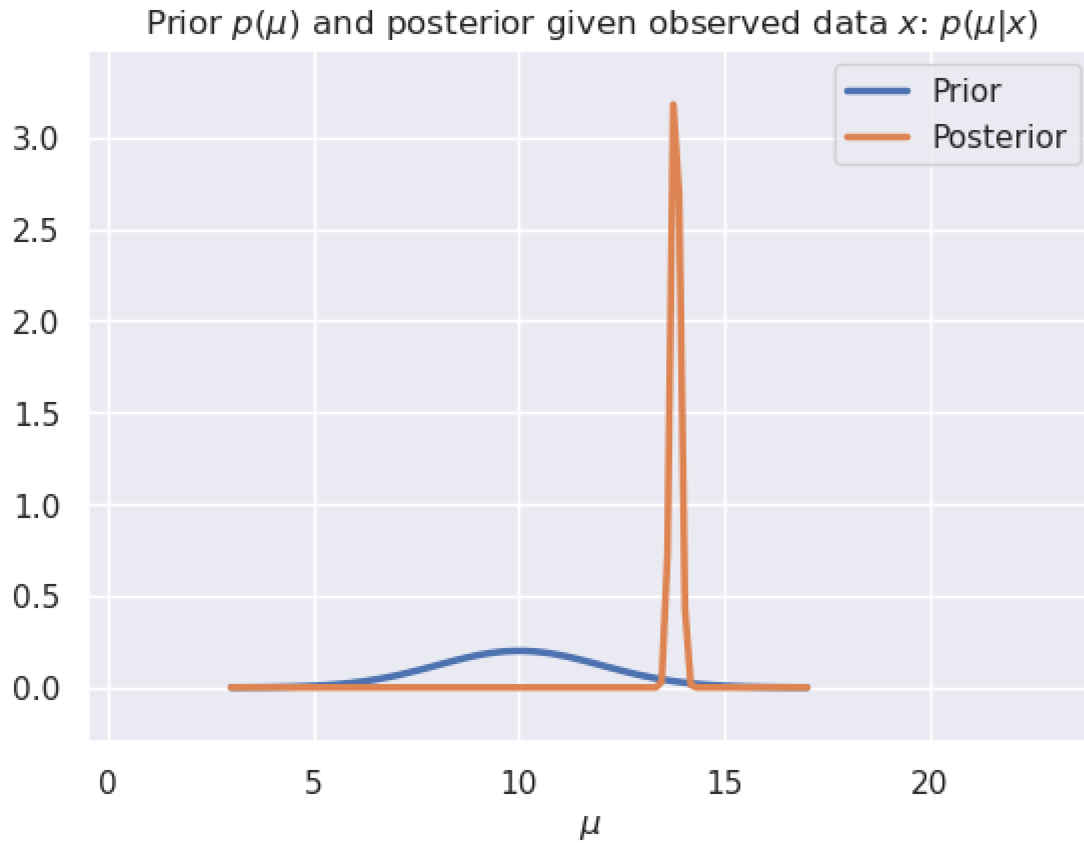


In [15]:
```
plot_gaussian_prior_and_posterior(10, 2, planets['radius'], 2)
```



In [16]:
```
plot_gaussian_prior_and_posterior(10, 2, planets.loc[planets['radius'] > 7.5, 'radius'],
plt.xlim([-0.5, 24])
```

Out[16]: (-0.5, 24.0)



PyMC3

We spent a lot of time doing algebra and computation for the review model. At this point, you might be asking: couldn't we do a lot of that work computationally? It turns out the answer is yes! PyMC3 is a Python library for Bayesian inference. You specify a probabilistic model (like the three we've just seen), and it will compute the posterior distribution over all unknown variables.

Let's try it out on the product review model:

$$x_i|\theta \sim \text{Bernoulli}(\theta) \quad (1)$$
$$\theta \sim \text{Beta}(\alpha, \beta) \quad (2)$$

We'll start by specifying our data: Microwave A has 3 positive reviews and 0 negative reviews, and Microwave B has 19 positive reviews and 1 negative review.

In [10]:
```
reviews_a = np.array([1, 1, 1])
reviews_b = np.array([1] * 19 + [0])
```

Next, we define the model and compute the posterior. Here's what it looks like:

In [11]:
```
import pymc3 as pm
import arviz as az

# Parameters of the prior
alpha = 2
beta = 3

# PyMC3 models should be specified inside a `with pm.Model() as ...:` block, like so:
with pm.Model() as model:
    # Define a Beta-distributed random variable called theta
```

```
    theta = pm.Beta('theta', alpha=alpha, beta=beta)

    # Defines a Bernoulli RV called x. Since x is observed, we
    # pass in the observed= argument to provide our data
    x = pm.Bernoulli('x', p=theta, observed=reviews_b)

    # This line asks PyMC3 to approximate the posterior.
    # Don't worry too much about how it works for now.
    trace = pm.sample(2000, chains=2, tune=1000, return_inferencedata=True)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 4 jobs)
NUTS: [theta]
```

100.00% [6000/6000 00:02<00:00 Sampling 2 chains, 0 divergences]

```
Sampling 2 chains for 1_000 tune and 2_000 draw iterations (2_000 + 4_000 draws total) t
ook 3 seconds.
```

In [12]:

```
trace
```

Out[12]:

arviz.InferenceData

▶ posterior

▶ log_likelihood

▶ sample_stats

▶ observed_data

In [13]:

```
trace.posterior
```

Out[13]:

xarray.Dataset

▶ Dimensions: (**chain**: 2, **draw**: 2000)

▼ Coordinates:

chain	(chain)	int64	0 1
draw	(draw)	int64	0 1 2 3 4 ... 1996 1997 1998 1999

▼ Data variables:

theta	(chain, draw)	float64	0.9505 0.9346 ... 0.882 0.9351

▼ Attributes:

created_at :	2022-09-22T22:48:15.312222
arviz_version :	0.12.1
inference_librar...	pymc3
inference_librar...	3.11.2
sampling_time :	3.3633666038513184
tuning_steps :	1000

In [14]:

```
trace.posterior['theta']
```

Out[14]: xarray.DataArray 'theta' (**chain**: 2, **draw**: 2000)

```
array([[0.95050373, 0.93458985, 0.90299713, ..., 0.88819423, 0.91063615,
        0.66067221],
       [0.71417212, 0.80309123, 0.87842866, ..., 0.88204977, 0.88204977,
        0.9350741 ]])
```

▼ Coordinates:

chain	(chain)	int64	0 1
draw	(draw)	int64	0 1 2 3 4 ... 1996 1997 1998 1999

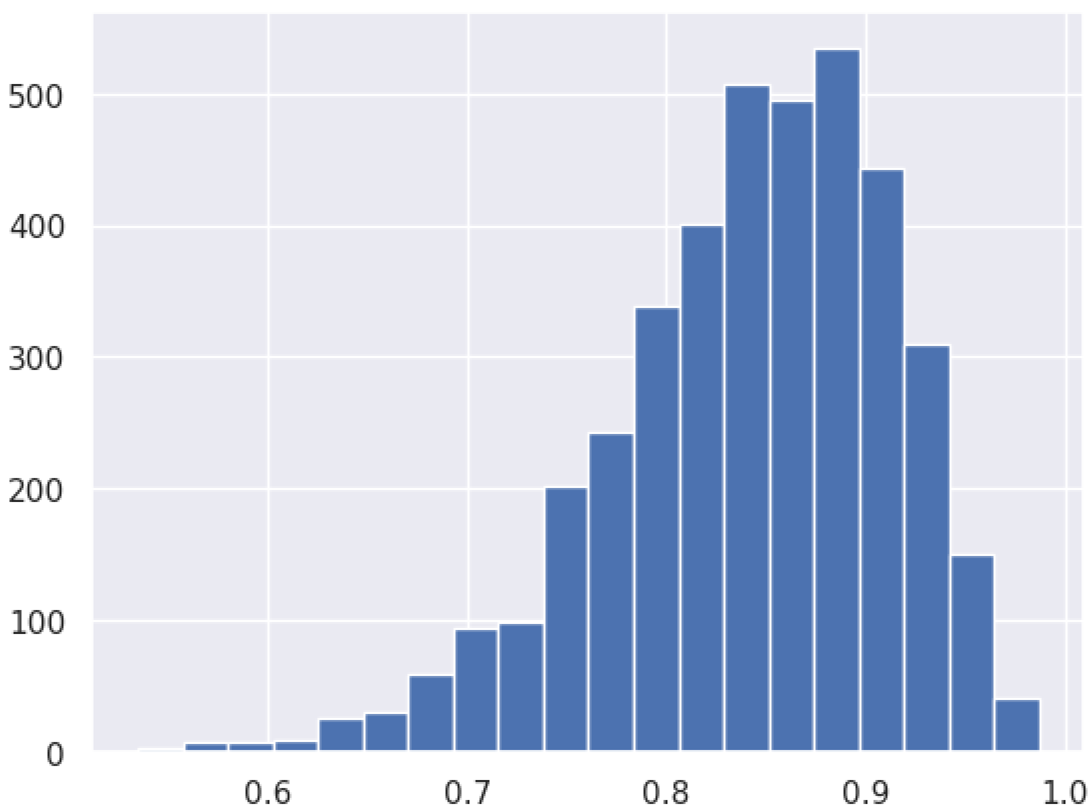
▶ Attributes: (0)

In [15]:

```
plt.hist(trace.posterior['theta'].values.flatten(), bins=20)
```

Out[15]:

```
(array([  3.,   8.,   7.,   9.,  25.,  30.,  59.,  94.,  99., 202., 243.,
        338., 401., 507., 495., 535., 444., 310., 150.,  41.]),
 array([0.53483228, 0.55743491, 0.58003754, 0.60264017, 0.6252428 ,
        0.64784543, 0.67044806, 0.69305069, 0.71565332, 0.73825595,
        0.76085858, 0.78346121, 0.80606384, 0.82866647, 0.8512691 ,
        0.87387173, 0.89647436, 0.91907699, 0.94167962, 0.96428225,
        0.98688488]),
 <BarContainer object of 20 artists>)
```



In [ ]: