[1]:
```
# make figures better:
%config InlineBackend.figure_format = 'retina'

import matplotlib
font = {'weight':'normal','size':22}
matplotlib.rc('font', **font)
matplotlib.rc('figure', figsize=(9.0, 6.0))
matplotlib.rc('xtick.major', pad=10) # xticks too close to border!

#from IPython.display import set_matplotlib_formats
#set_matplotlib_formats('png', 'pdf')

import warnings
warnings.filterwarnings('ignore')
```

DS1 Lecture 23

James Bagrow, james.bagrow@uvm.edu, http://bagrow.com

Today

Intro to Bayesian Inference: checking whether a coin is fair

[2]:
```
import random
import scipy, scipy.stats
```

[Notes on the board]

[3]:
```
scipy.special.comb(100,5) # don't try to compute factorials yourself!
```

[3]:
```
75287520.0
```

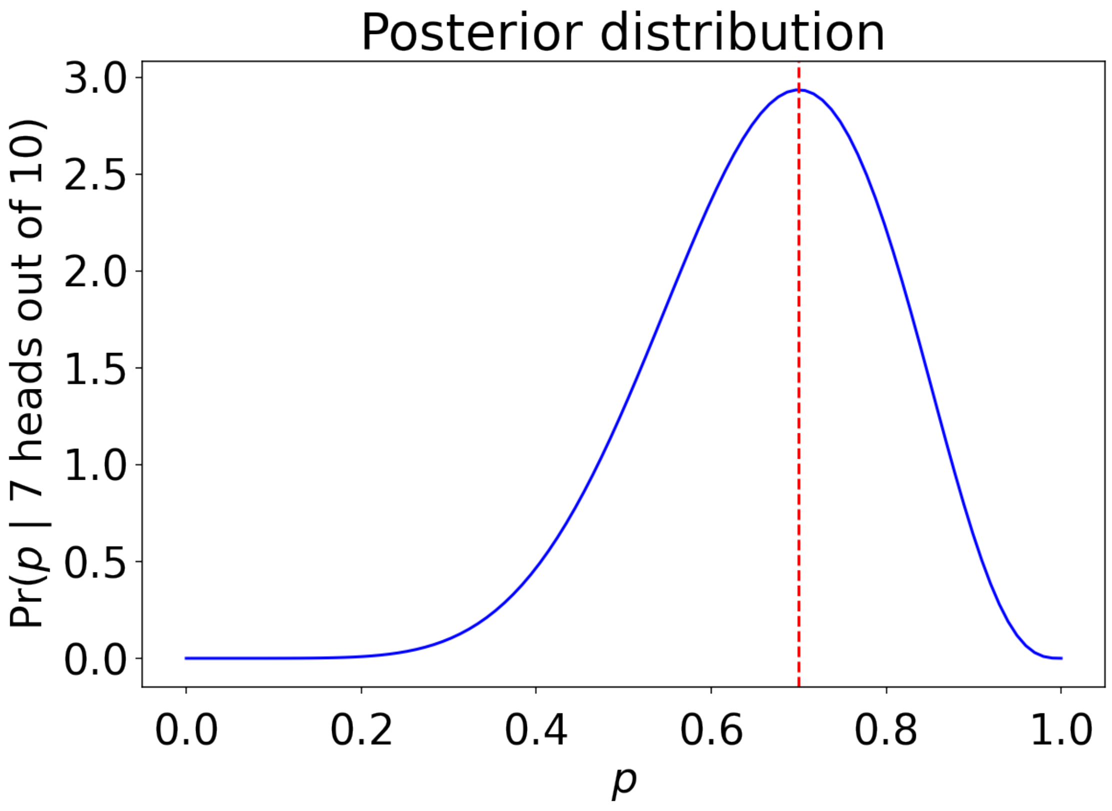
[4]:
```
print(scipy.special.comb.__doc__.strip().split("\n")[0], "\n...") # first line of␣
 ↪docstring
```

```
The number of combinations of N things taken k at a time.
...
```

Here is a plot of the posterior probability distribution for a coin bias of p given we observed 7 heads of 10 trials. The posterior distribution's density (derived on the board) for different values of `p` is the variable `y`; it is called the **beta distribution**.

[5]:
```
f = scipy.special.factorial
c = scipy.special.comb

p = np.linspace(0,1,100)
y = f(10+1)/f(7)/f(10-7) * p**7 * (1-p)**(10-7)

print("if p = 0.5, then Pr(7 heads) =", c(10,7)*0.5**10)

plt.plot(p,y, 'b-');
plt.axvline(x=0.70, color='red', ls='dashed')
plt.xlabel("$p$");
plt.ylabel("Pr($p$ | 7 heads out of 10)")
plt.title("Posterior distribution");
```

```
if p = 0.5, then Pr(7 heads) = 0.1171875
```

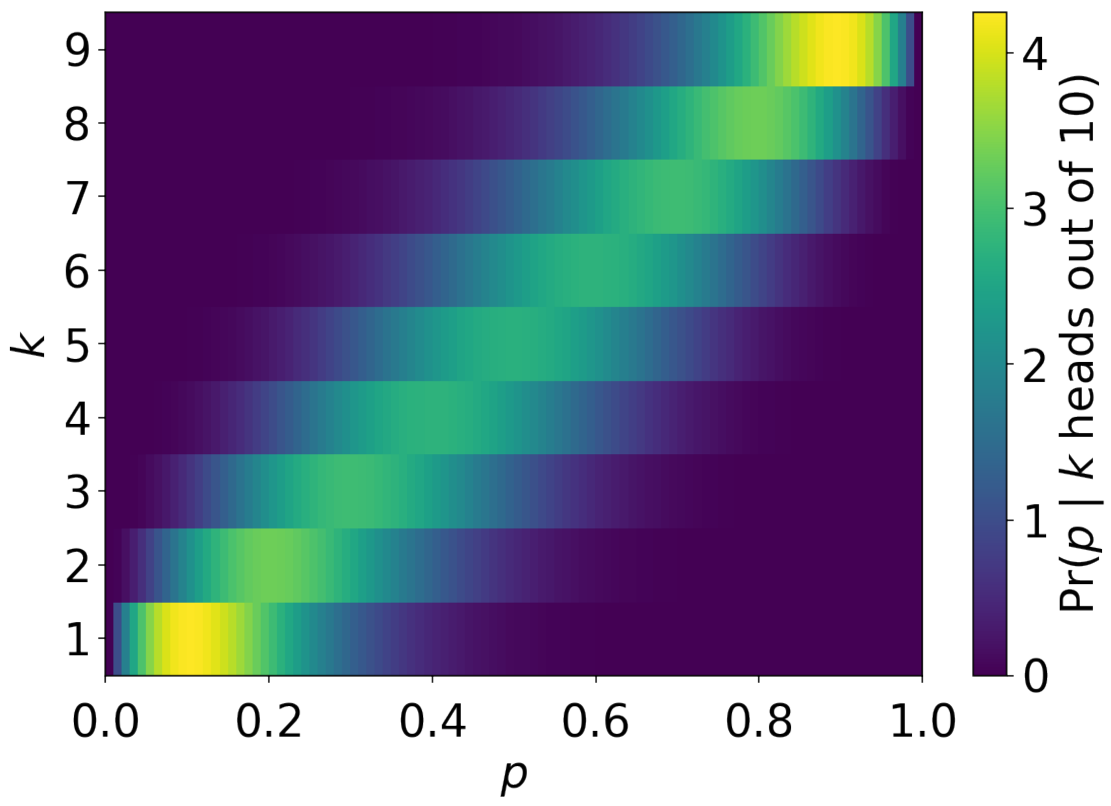


Now let's study the posterior as a function of both p and k.

- Instead of drawing a curve for each value of k, let's **stack them in a matrix** so that each row is a "top-down" view of the curve, with color representing the probability values instead of height.

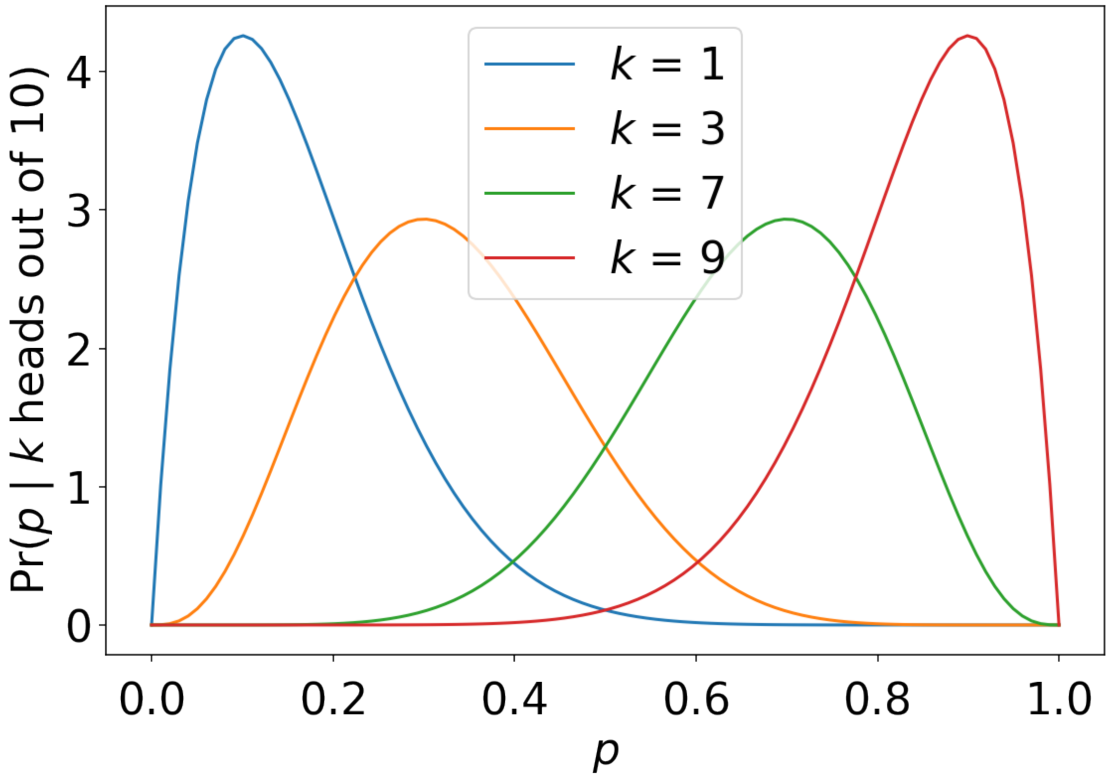
[6]:
```
n = 10
P = np.linspace(0,1,100)
K = range(1,n)
mat = []
for p in P:
    row = []
    for k in K:
        Pr = f(n+1)/f(k)/f(n-k) * p**k * (1-p)**(n-k)
        row.append(Pr)
        #print(p,k,Pr)
    mat.append(row)

K = np.array(K)
Prs = np.array(mat).T
```

```
# imshow is a matrix plotter!
plt.imshow(Prs, extent=(P.min(),P.max(),K.min(),K.max()+1),
           interpolation='nearest', aspect="auto", origin='lower')

cbar = plt.colorbar()
#plt.yticks(K[4::5]+0.5,K[4::5]) # center y-ticks in rows
plt.yticks(K+0.5,K) # center y-ticks in rows
plt.xlabel(r"$p$")
plt.ylabel(r"$k$")
cbar.set_label(r"Pr($p$ | $k$ heads out of 10)");
```



And, just in case, here are the (traditional) curves for a few values of $k$ (it gets a little too crowded if we look at too many values of $k$):

[7]:
```
p = np.linspace(0,1,100)
f = scipy.special.factorial

for k in [1,3,7,9]:
    y = f(10+1)/f(k)/f(10-k) * p**k * (1-p)**(10-k)
    plt.plot(p,y, label="$k$ = {}".format(k))

plt.xlabel("$p$")
plt.ylabel("Pr($p$ | $k$ heads out of 10)")
plt.legend(loc='best')
plt.show()
```



See how symmetric the curves are?

Now let's look at how the **posterior changes** as more unbiased coins are flipped.

- As more data become available we should start to see increasing probability that the coin is fair.

What we will do is flip 100 coins in total, but only look at the posterior for the first flip, then the first two flips, etc. and see how the posterior changes as we add more coin flips.

Here are the coin flips (0 = tails, 1 = heads):

[8]:
```
total_num_flips = 100

data = []
for trial in range(total_num_flips):
    if random.random() < 0.5: # unbiased coin
        data.append(0)
    else:
        data.append(1)
print(data[:10], "...")
```

```
[0, 1, 1, 0, 1, 0, 0, 0, 1, 0] ...
```

Here is a list of how many flips we will look at (we'll plot the posterior for each value in this list):

[9]:
```
list_first_n_flips = [0, 1, 2, 3, 4, 5, 10, 15, 20, total_num_flips]
```

This means we will start by looking at zero flips (!) and go through all the way to looking at all of the flips in the data.

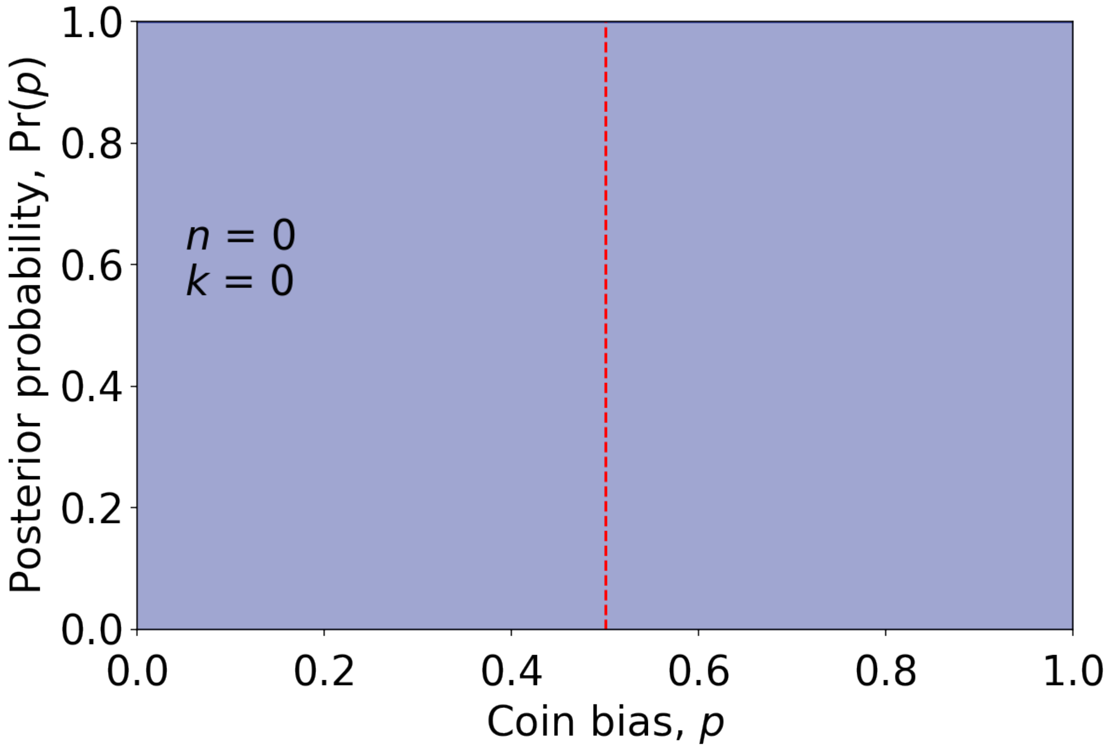
The `scipy.stats` package provides a number of probability distributions, including the beta distribution we can use here

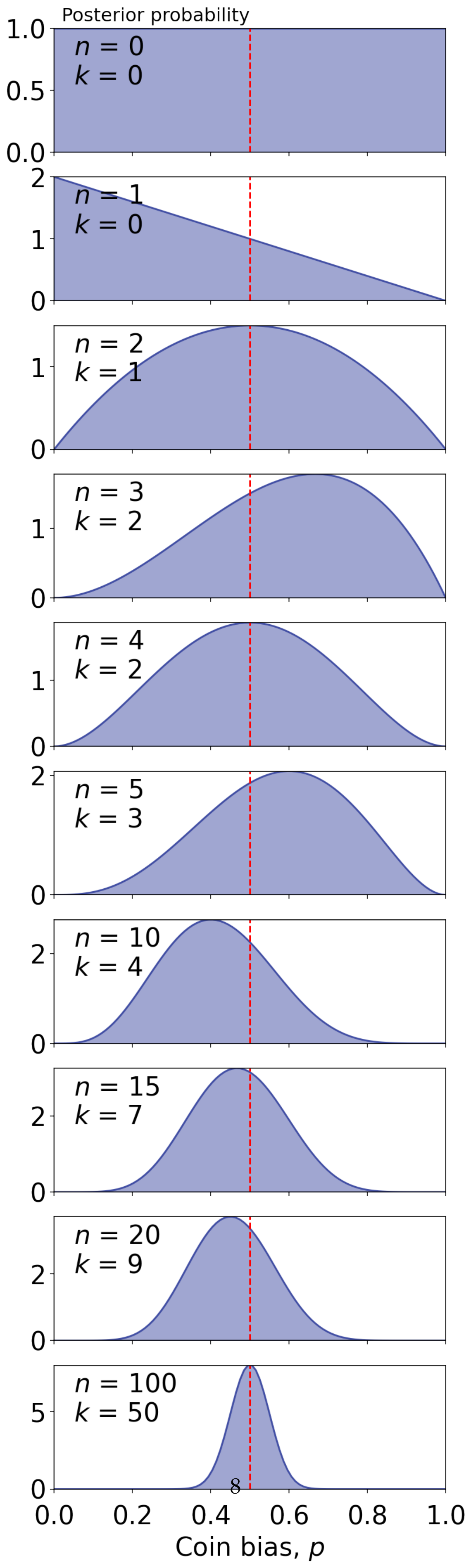
[10]:
```
# posterior distribution for binomial is beta (see board notes):
posterior = scipy.stats.beta
# ^^^ use IPython to explore how scipy.stats distributions work!

# print (excerpt of) docstring:
list_lines = posterior.__doc__.strip().split("\n")
del list_lines[2:47]
del list_lines[32:]
list_lines.insert(1, "\n    [Omitted...]")
list_lines.append("    [Omitted...]")
print("\n".join(list_lines))
```

```
A beta continuous random variable.

    [Omitted...]

        Endpoints of the range that contains fraction alpha [0, 1] of the
        distribution

    Notes
    -----
    The probability density function for `beta` is:

    .. math::

        f(x, a, b) = \frac{\Gamma(a+b) x^{a-1} (1-x)^{b-1}}
                          {\Gamma(a) \Gamma(b)}

    for :math:`0 <= x <= 1`, :math:`a > 0`, :math:`b > 0`, where
    :math:`\Gamma` is the gamma function (`scipy.special.gamma`).

    `beta` takes :math:`a` and :math:`b` as shape parameters.

    The probability density above is defined in the "standardized" form. To
shift
    and/or scale the distribution use the ``loc`` and ``scale`` parameters.
    Specifically, ``beta.pdf(x, a, b, loc, scale)`` is identically
    equivalent to ``beta.pdf(y, a, b) / scale`` with
    ``y = (x - loc) / scale``. Note that shifting the location of a distribution
    does not make it a "noncentral" distribution; noncentral generalizations of
    some distributions are available in separate classes.

    Examples
    --------
    >>> from scipy.stats import beta
    >>> import matplotlib.pyplot as plt
    >>> fig, ax = plt.subplots(1, 1)
    [Omitted...]
```

At last, here is the plot:

```
[11]: # set up two plots, an intro fig to show the axes, and then a stacked
# set of figures to show the posterior for each flip number:
f_intro_fig, axes_intro_fig = plt.subplots(1,1)
f, axes = plt.subplots(len(list_first_n_flips), sharex=True, figsize=(6,23));

# for each set of data, plot the posterior:
p = np.linspace(0, 1, 100)
for i, n in enumerate(list_first_n_flips):

    k = sum(data[:n]) # number of heads
    y = posterior.pdf(p, 1 + k, 1 + n - k)

    if i == 0: # do intro fig and first stacked plot
        this_ax = [f_intro_fig.gca(),axes[i]]
    else: # just a stacked plot
        this_ax = [axes[i]]

    for a in this_ax:
        a.plot(p, y, color="#424FA4");
        a.fill_between(p, 0, y, color="#424FA4", alpha=0.5);

        label="$n$ = {}\n$k$ = {}".format(n, k)
        a.annotate(label, xy=(0.05, 0.55), xycoords='axes fraction')
        a.autoscale(tight=True)
        a.axvline(x=0.5, color='red', linestyle='--')

# label the "intro" plot:
axes_intro_fig.set_xlabel("Coin bias, $p$")
axes_intro_fig.set_ylabel("Posterior probability, Pr($p$)")

# label the multiplot:
axes[0].set_title("Posterior probability", ha='right', fontsize=16)
axes[-1].set_xlabel("Coin bias, $p$")

plt.show()
```


Posterior probability, Pr(p)
Coin bias, p
n = 0
k = 0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Posterior probability
n = 0
k = 0
n = 1
k = 0
n = 2
k = 1
n = 3
k = 2
n = 4
k = 2
n = 5
k = 3
n = 10
k = 4
n = 15
k = 7
n = 20
k = 9
n = 100
k = 50
Coin bias, p

We see that, with more data (higher n and k, the posterior distribution shifts towards the center, revealing the unbiased nature of the coin (the true value of p is $1/2$).

Summary

Examining the posterior probability lets us reason about models and their relationships to data. This is the fundamental power of Bayesian inference