In [186…
```
%matplotlib inline
import numpy as np
import pymc as pm
import arviz as az
import matplotlib.pyplot as plt
from scipy import stats
from scipy.optimize import minimize
import pandas as pd
```

Question 2

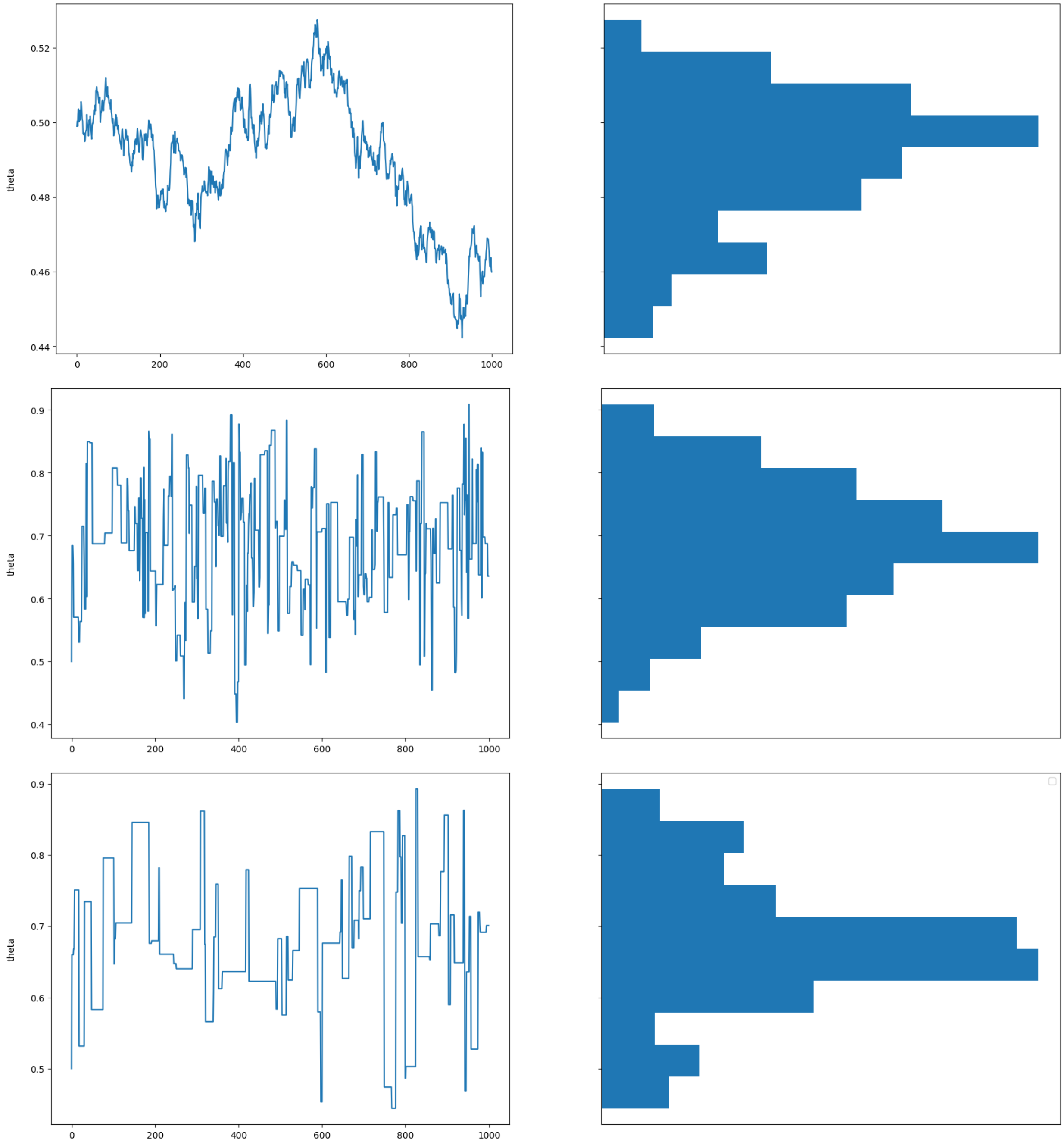
In [187…
```
def post(theta, Y, alpha=1, beta=1):
    if 0 <= theta <= 1:
        prior = stats.beta(alpha, beta).pdf(theta)
        like = stats.bernoulli(theta).pmf(Y).prod()
        prob = like * prior
    else:
        prob = -np.inf
    return prob


Y = stats.bernoulli(0.7).rvs(20)
```

In [188…
```
def mcmc(can_sd=0.05):
    n_iters = 1000
    alpha = beta = 1
    theta = 0.5
    trace = {"theta":np.zeros(n_iters)}
    p2 = post(theta, Y, alpha, beta)
    for iter in range (n_iters):
        theta_can = stats.norm(theta, can_sd).rvs(1)
        p1 = post(theta_can, Y, alpha, beta)
        pa = p1 / p2
        if pa > stats.uniform(0, 1).rvs(1):
            theta = theta_can
            p2 = p1
        trace["theta"][ iter ] = theta
    return trace
```

In [189…
```
traces = []
sds = [0.002, 0.5, 1.5]
for can_sd in sds:
    traces.append(mcmc(can_sd))
```

In [190…
```
for idx,trace in enumerate(traces):
    _, axes = plt.subplots(1,2, sharey=True)
    axes[0].plot(trace["theta"])
    axes[0].set_ylabel("theta", rotation=90, labelpad=15)
    axes[1].hist(trace['theta'], orientation="horizontal", density=True)
    axes[1].set_xticks([])
plt.legend();
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

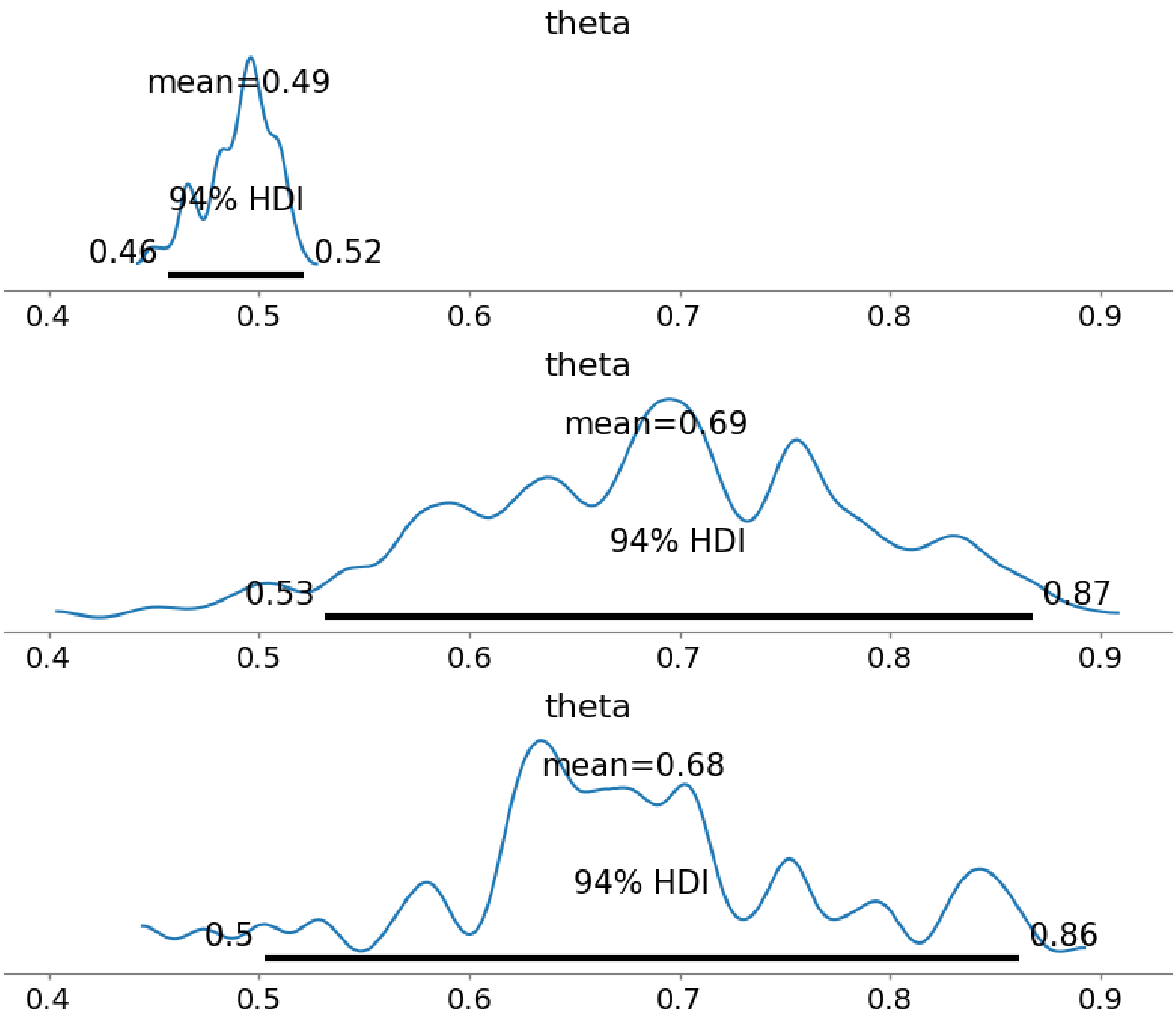


Question 2.A Answer

Comparing the histograms with sd 0.002, 0.05, and 1.5, it seems as though the one with a sd that was really low (0.002), the markov chain builds up slowly so it's less efficient while the ones with higher sd seem to be more efficient on sampling.

In [191...

```
_, axes = plt.subplots(3, 1, figsize=(10, 8), sharex=True)
plt.subplots_adjust(wspace=0.4,hspace=0.4)
for trace, ax in zip(traces, axes.ravel()):
    az.plot_posterior(trace, ax=ax)
```



Question 2.B Answer

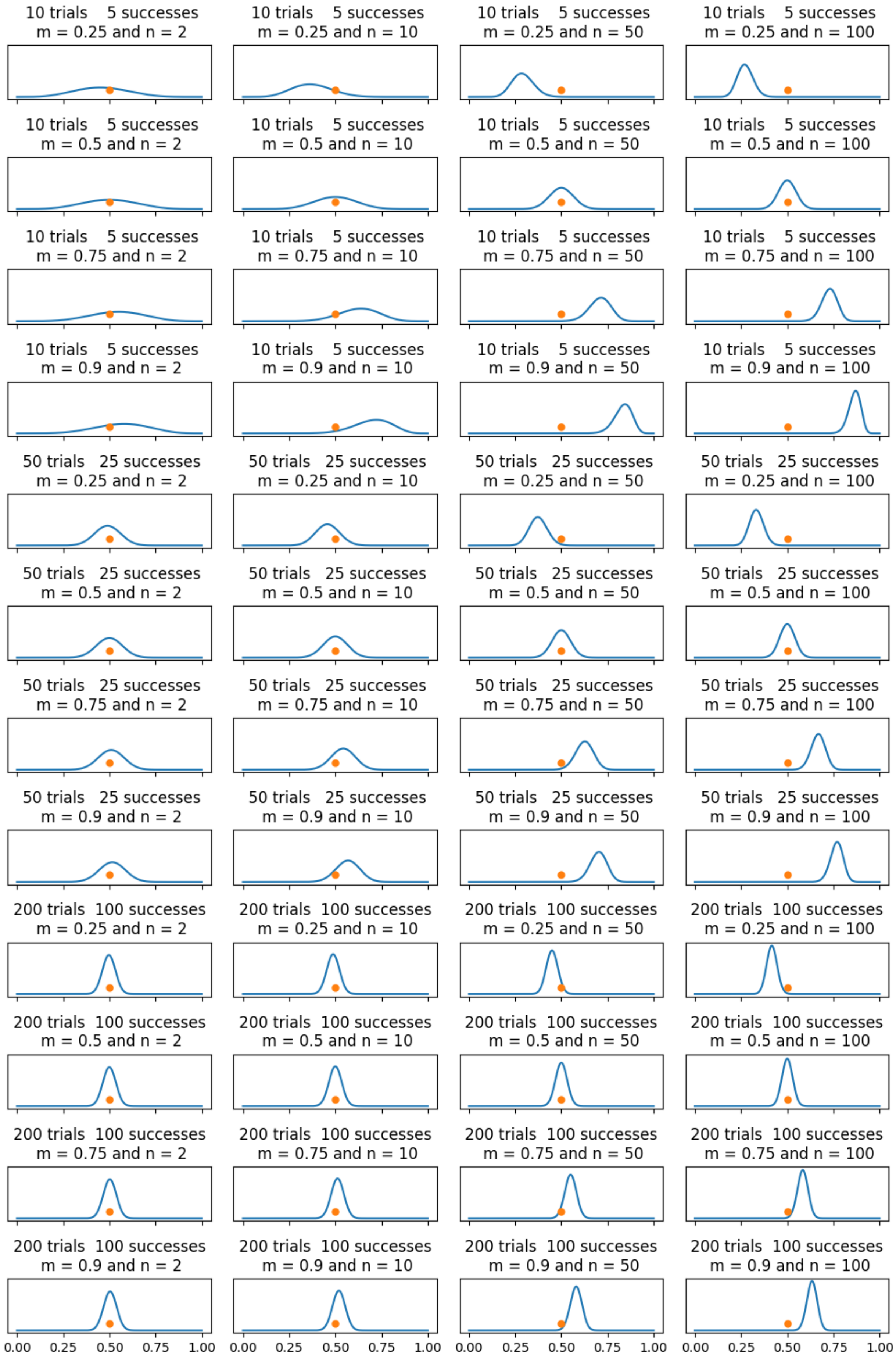
Sorry, the graph looks wanky, but we can see the mean increasing as the sd increases.

Question 3

In [192…

```
ms = [0.25, 0.5, 0.75, 0.9]
ns = [2, 10, 50, 100]

pairs = [(5,10), (25,50), (100,200)]

_, axes = plt.subplots(len(ns)*len(pairs), len(ms), figsize=(10, 15), sharex=True, sha
plt.subplots_adjust(wspace=0.6,hspace=0.6)
axes = np.ravel(axes)

for i, (X,N) in enumerate(pairs):
    ## m and n are prior params
    for j, m in enumerate(ms):
        for k, n in enumerate(ns):
            alpha = m*n
            beta = (1-m)*n

            theta = np.linspace(0,1,1500)
```

```
            ## P(theta|Y)
            p = stats.beta.pdf(theta, alpha+X, beta+N-X)

            ## val X/N
            val = X/N

            id_plot = i*(len(ms)*len(ns))+j*len(ns)+k

            ## ploting the posterior distributions
            ## one plot for every data pair and every different prior param combinatic
            axes[id_plot].plot(theta, p)
            axes[id_plot].set_yticks([])
            axes[id_plot].plot(val, 2, marker="o", ms=5)
            axes[id_plot].set_title(f"{N:4d} trials {X:4d} successes \n m = {m} and n

plt.tight_layout()
plt.show()
```



Question 3.A Answer

Comparing the posterior distributions for each data pair, we see the peaks getting higher as the number of trials increase.

Answer 3.B Answer

Comparing the posterior distributions for each given prior parameters, not only are the peeks getting higher, but we see the peaks move (for the most part).

Question 4

In [193…

```
data = pd.read_csv("C:/Users/jacqu/OneDrive/Documents/MSDS/datasets/ArtHistBooks.csv")
```

In [194…

```
data.head()
```

Out[194]:

	ArtBooks	HistoryBooks	TableBooks	Purchase
0	0	0	1	0
1	0	1	0	0
2	0	0	0	0
3	1	0	1	0
4	1	1	1	0

In [195…

```
data.ArtBooks.replace({1:1, 2:1, 3:1}, inplace=True)
```

Part 1

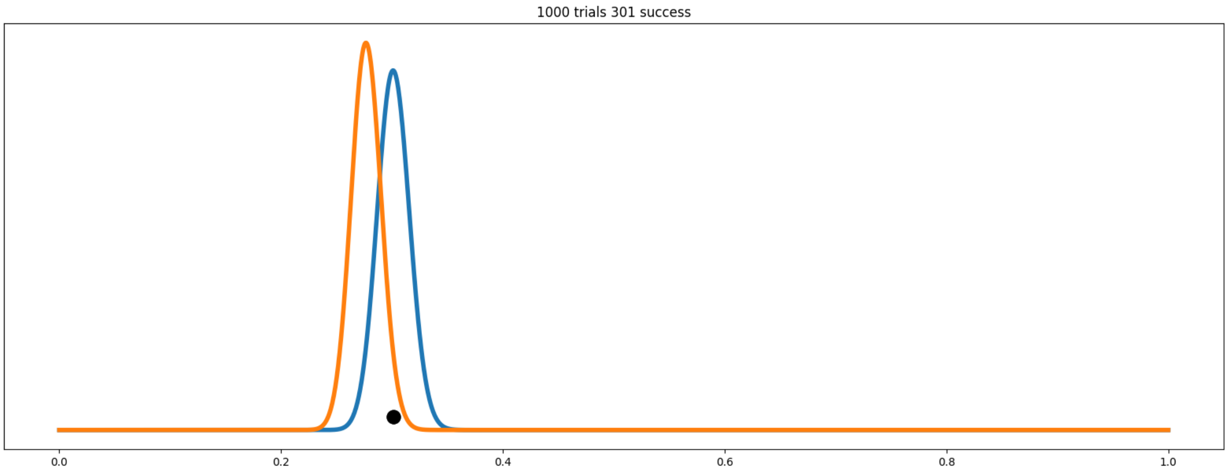
Use beta-binomial conjugate analysis to determine the posterior probabilities for purchases of art books, history books and coffee table books, first by using beta priors with values of the hypterparameters that represent lack of information. Then compute these probabilities again with beta priors that show strong weighting for low likelihood of a book purchase...

In [230…

```
## ArtBooks
_, axes = plt.subplots(1,1)
axes = np.ravel(axes)

## number of trials
N = len(data)
## y = the number of successes in current data
y = data.ArtBooks.value_counts()[1]

post_p = []

## beta priors where we start with 1,1
## then go with low likelihood of book purchase
beta_params = [(1, 1), (5, 100)]
theta = np.linspace(0, 1, 1500)
for jdx, (a_prior, b_prior) in enumerate(beta_params):
```

```
    ## args are x, alpha, and beta where
    ## alpha_posterior = alpha_prior + y
    ## beta_posterior = beta_prior + N - y
    p_theta_given_y = stats.beta.pdf(theta, a_prior + y, b_prior + N - y)
    axes[0].plot(theta, p_theta_given_y, lw=4)
    axes[0].set_yticks([])
    axes[0].plot(np.divide(y, N), 1, color="k", marker="o", ms=12)
    axes[0].set_title(f"{N} trials {y} success")

    max_p = theta[np.argmax(p_theta_given_y)]
    post_p.append(max_p)
print(post_p)
```

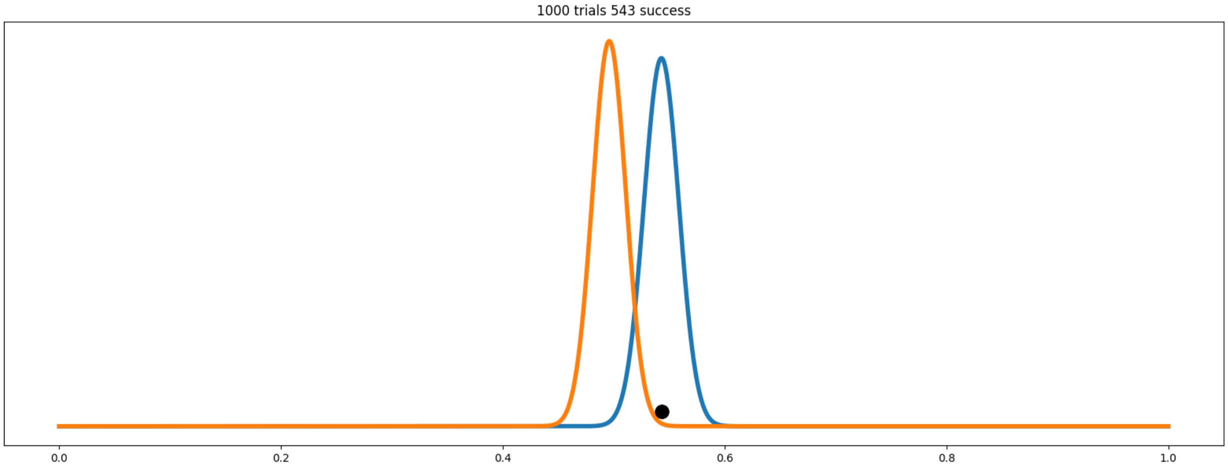
```
[0.3008672448298866, 0.276851234156104]
```



In [229…

```
## HistoryBooks
_, axes = plt.subplots(1,1)
axes = np.ravel(axes)

N = len(data)
y = data.HistoryBooks.value_counts()[1]

post_p = []

beta_params = [(1, 1), (5, 100)]
theta = np.linspace(0, 1, 1500)
for jdx, (a_prior, b_prior) in enumerate(beta_params):
    p_theta_given_y = stats.beta.pdf(theta, a_prior + y, b_prior + N - y)
    axes[0].plot(theta, p_theta_given_y, lw=4)
    axes[0].set_yticks([])
    axes[0].plot(np.divide(y, N), 1, color="k", marker="o", ms=12)
    axes[0].set_title(f"{N} trials {y} success")

    max_p = theta[np.argmax(p_theta_given_y)]
    post_p.append(max_p)
print(post_p)
```

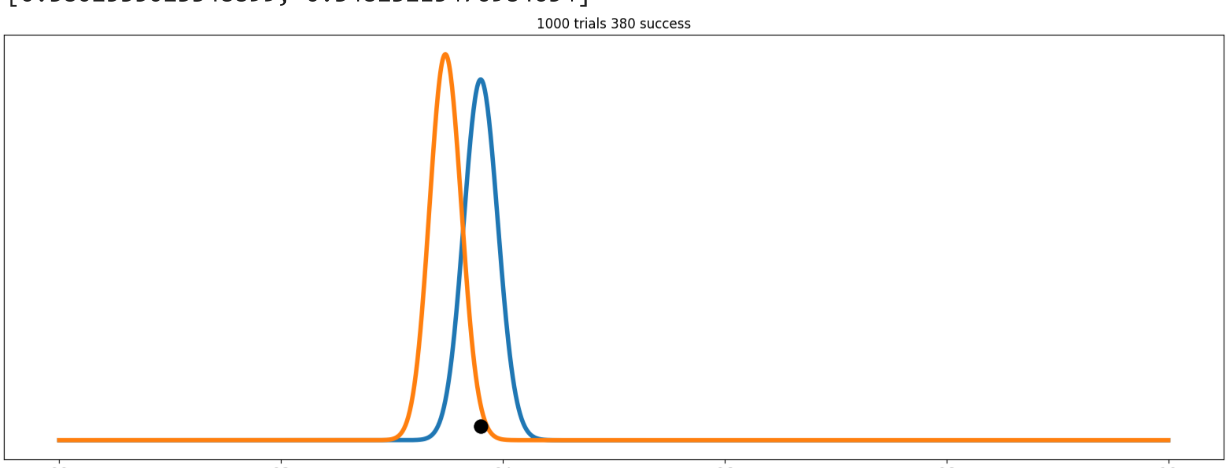
```
[0.543028685790527, 0.495663775850567]
```



In [228…

```
## TableBooks
_, axes = plt.subplots(1,1)
axes = np.ravel(axes)

post_p = []

N = len(data)
y = data.TableBooks.value_counts()[1]

beta_params = [(1, 1), (5, 100)]
theta = np.linspace(0, 1, 1500)
for jdx, (a_prior, b_prior) in enumerate(beta_params):
    p_theta_given_y = stats.beta.pdf(theta, a_prior + y, b_prior + N - y)
    axes[0].plot(theta, p_theta_given_y, lw=4)
    axes[0].set_yticks([])
    axes[0].plot(np.divide(y, N), 1, color="k", marker="o", ms=12)
    axes[0].set_title(f"{N} trials {y} success")

    max_p = theta[np.argmax(p_theta_given_y)]
    post_p.append(max_p)
print(post_p)
```

```
[0.3802535023348899, 0.34823215476984654]
```



Part 2

Use beta-binomial conjugate analysis to determine the separate probabilities for purchases of the new book given each possible combination of prior purchases of art books, history books and coffee table books, first by using beta priors with values of the hypterparameters that represent lack of information. Then compute these probabilities again with beta priors that show strong weighting for low likelihood of a book purchase...

In [199...

```
combos = data[["ArtBooks","HistoryBooks","TableBooks"]].value_counts().reset_index(lev
```

In [200...

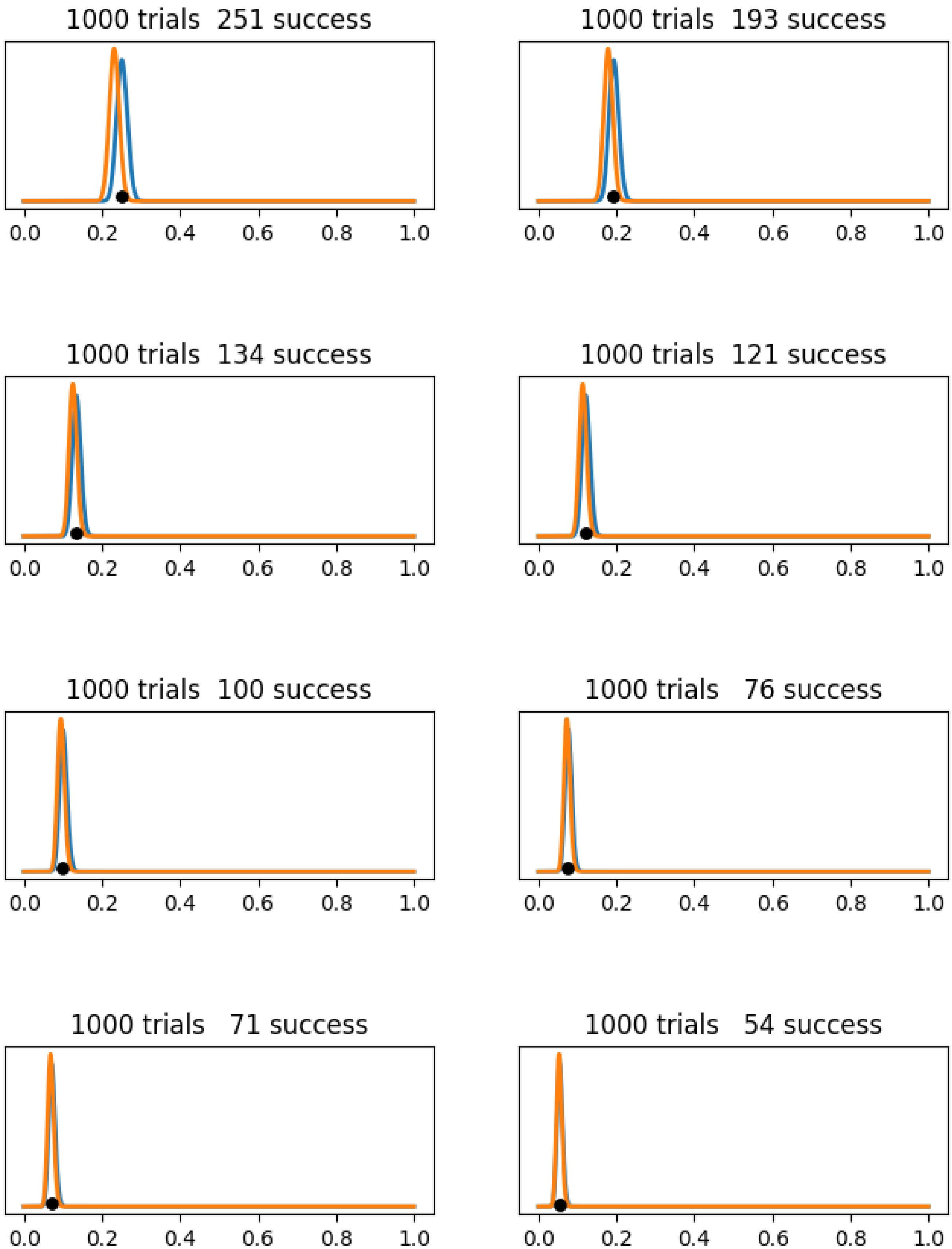
```
combos
```

Out[200]:

	ArtBooks	HistoryBooks	TableBooks	count
0	0	1	0	251
1	0	0	0	193
2	0	0	1	134
3	0	1	1	121
4	1	1	0	100
5	1	0	0	76
6	1	1	1	71
7	1	0	1	54

In [226...

```
_, axes = plt.subplots(4,2, figsize=(8,10))
plt.subplots_adjust(hspace=1)
axes = np.ravel(axes)

labels = ["010","010","000","000","001","001","011","011","110","110","100","100","11
post_p = []

for idx, each_combo in enumerate(combos["count"]):
    N = len(data)
    y = each_combo

    beta_params = [(1, 1), (5, 100)]
    theta = np.linspace(0, 1, 1500)
    for jdx, (a_prior, b_prior) in enumerate(beta_params):
        ## alpha_posterior = alpha_prior + y
        ## beta_posterior = beta_prior + N - y
        max_p = 0
        p_theta_given_y = stats.beta.pdf(theta, a_prior + y, b_prior + N - y)
        axes[idx].plot(theta, p_theta_given_y, lw=2)
        axes[idx].set_yticks([])
        axes[idx].plot(np.divide(y, N), 1, color="k", marker="o", ms=5)
        axes[idx].set_title(f"{N:4d} trials {y:4d} success")

        max_p = theta[np.argmax(p_theta_given_y)]
        post_p.append(max_p)

post = pd.DataFrame({"combos":labels,"posterior probability (unweighted vs. weighted)'
```


1000 trials 251 success
1000 trials 193 success
1000 trials 134 success
1000 trials 121 success
1000 trials 100 success
1000 trials 76 success
1000 trials 71 success
1000 trials 54 success
0.0 0.2 0.4 0.6 0.8 1.0

In [227…

```
post
```

Out[227]:

	combos	posterior probability (unweighted vs. weighted)
0	010	0.250834
1	010	0.231488
2	000	0.192795
3	000	0.178786
4	001	0.134089
5	001	0.125417
6	011	0.120747
7	011	0.113409
8	110	0.100067
9	110	0.094063
10	100	0.076051
11	100	0.072715
12	111	0.070714
13	111	0.068045
14	101	0.054036
15	101	0.052702

In [ ]: