In [20]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_probability as tfp
tfd = tfp.distributions
```

Exercises 1

1.

Combinations of two dice that add to a given number:

In [21]:

```
die = np.arange(1, 7)
dice = np.tile(die, (6, 1))
dice = pd.DataFrame(dice + dice.transpose(), columns=die, index=die)
dice
```

Out[21]:

	1	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

In [22]:

```
n_10 = (dice == 10).values.sum()
n_10
```

Out[22]:

```
3
```

In [23]:

```
n_11 = (dice == 11).values.sum()
n_11
```

Out[23]:

```
2
```

a)

Prior probability of craps and roulette:

In [24]:

```
prior = np.array([1, 2])
prior = prior / prior.sum()
prior
```

Out[24]:

```
array([0.33333333, 0.66666667])
```

Likelihood for craps and roulette:

In [25]:

```
likelihood = np.array([n_11/36, 1/38])
likelihood
```

Out[25]:

```
array([0.05555556, 0.02631579])
```

Posterior probability for craps and roulette:

In [26]:

```
posterior = prior * likelihood
posterior = posterior / posterior.sum()
posterior
```

Out[26]:

```
array([0.51351351, 0.48648649])
```

In [27]:

```
posterior[0].round(2)
```

Out[27]:

```
0.51
```

b)

Prior probability of craps and roulette:

In [28]:

```
prior = np.array([2, 1])
prior = prior / prior.sum()
prior
```

Out[28]:

```
array([0.66666667, 0.33333333])
```

Likelihood for craps and roulette:

In [29]:

```
likelihood = np.array([n_10/36, 1/38])
likelihood
```

Out[29]:

```
array([0.08333333, 0.02631579])
```

Posterior probability for craps and roulette:

In [30]:

```
posterior = prior * likelihood
posterior = posterior / posterior.sum()
posterior
```

Out[30]:

```
array([0.86363636, 0.13636364])
```

In [31]:

```
posterior[1].round(2)
```

Out[31]:

```
0.14
```

2.

Assuming you pick door 1 and the host opens door 3, the prior probabilities are following:

In [32]:

```
prior = np.ones(3) / 3
prior
```

Out[32]:

```
array([0.33333333, 0.33333333, 0.33333333])
```

Likelihood counts the ways the car can have ended up behind doors 1 and 2 given that door 3 has a goat.

In [33]:

```
likelihood = np.array([1, 2, 0])
```

Probabilities for each door are:

In [34]:

```
posterior = prior * likelihood
posterior = posterior / posterior.sum()
posterior.round(2)
```

Out[34]:

```
array([0.33, 0.67, 0.  ])
```

Door 2 has higher posterior probability, so the contestant should always switch. This can be explained by the fact that the host always gives the contestant new and reliable information on the game setup. The contestant should update their beliefs because they always have more information when choosing between the doors after the goat reveal than before it.

3.

In [35]:

```
fruits = pd.DataFrame(
    [[3, 4, 3],
    [1, 1, 0],
    [3, 3, 4]],
    columns=['apples', 'oranges', 'limes'],
    index=['r', 'b', 'g'],
)
p_box = pd.Series(dict(r=0.2, b=0.2, g=0.6))
```

```
fruits_p = fruits / fruits.values.sum(axis=1, keepdims=True)
fruits_p
```

Out[35]:

	apples	oranges	limes
r	0.3	0.4	0.3
b	0.5	0.5	0.0
g	0.3	0.3	0.4

a)

In [36]:

```
combined_p = fruits_p * np.tile(p_box.values[..., np.newaxis], (1, 3))
combined_p.apples.sum().round(2)
```

Out[36]:

```
0.34
```

Probability of an apple is 1/3.

b)

In [37]:

```
prior = p_box.values
likelihood = fruits_p.oranges.values
posterior = prior * likelihood
posterior = posterior / posterior.sum()
posterior[1].round(2)
```

Out[37]:

```
0.28
```

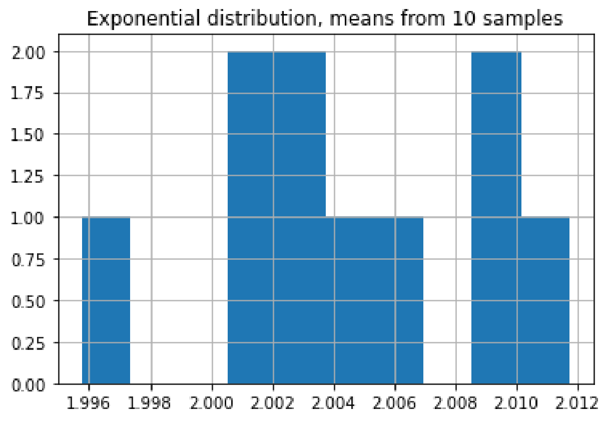
P(g|apple) = 0.28

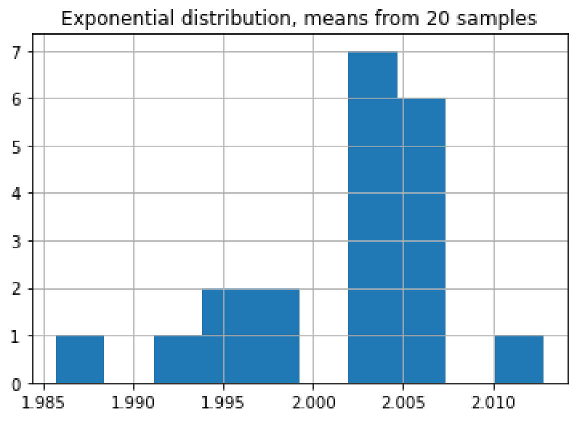
4.

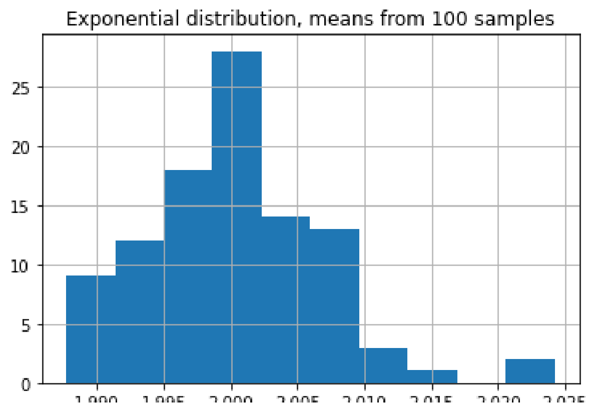
In [38]:

```
mu = 0.5
sigma = 0.25

distributions = {
    'exponential': tfd.Exponential(np.repeat(mu, repeats=100_000)),
    'normal': tfd.Normal(np.repeat(mu, repeats=100_000), scale=np.repeat(sigma, repeats=100_000)),
    'poisson': tfd.Poisson(np.repeat(mu, repeats=100_000)),
    'uniform': tfd.Uniform(np.repeat(0., repeats=100_000), np.repeat(1., repeats=100_000)),
    'bernoulli': tfd.Bernoulli(probs=np.repeat(mu, repeats=100_000)),
}

for name, dist in distributions.items():
    for n in 10, 20, 100:
        sample = dist.sample(n).numpy()
        means = pd.Series(sample.mean(axis=1))

        log_prob = tfd.Normal(mu, np.sqrt(sigma**2 / n)).log_prob(means).numpy()
        print(f"Log prob ({name.capitalize()}, {n}): {log_prob}")

        fig = means.hist()
        fig.set_title(f'{name.capitalize()} distribution, means from {n} samples')
        plt.show()
```

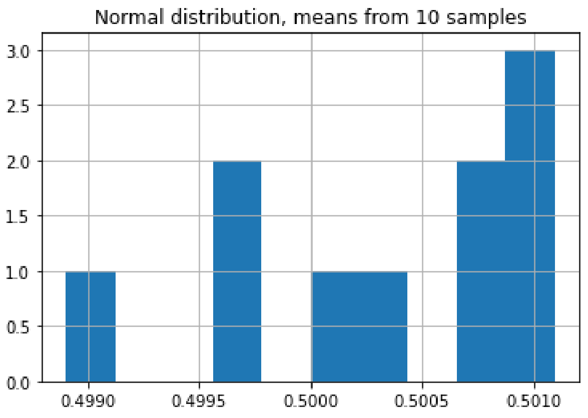
```
Log prob (Exponential, 10): [-179.72829628 -177.36380748 -181.21075245 -179.09689191 -179.10188611
 -179.31664852 -180.63619142 -178.75823164 -178.84290862 -180.59429028]
Log prob (Exponential, 20): [-361.38768003 -355.84557551 -351.1952306  -359.75477232 -354.89120611
 -359.23424227 -356.60495173 -360.42650309 -357.05823732 -355.45014724
 -361.06773538 -364.21198581 -359.46874669 -359.16195388 -361.18476978
 -361.14689332 -359.98271008 -359.16511168 -359.32410596 -360.34117463]
Log prob (Exponential, 100): [-1792.01635771 -1795.02071825 -1804.64256169 -1814.51987421
 -1777.10175611 -1776.2994927  -1800.17401025 -1796.23688418
 -1809.75574491 -1781.82306687 -1782.796502   -1797.7201053
 -1805.52910168 -1781.52421841 -1785.03070492 -1797.51366983
 -1806.36238731 -1775.76781099 -1793.0658722  -1795.75258086
 -1779.43479906 -1783.07304462 -1810.03568481 -1821.47443732
 -1793.26418752 -1787.79064341 -1809.6330061  -1772.55919994
 -1819.44081006 -1819.03816546 -1774.55759632 -1790.65658715
 -1784.78169717 -1792.13481436 -1793.32672122 -1802.39292141
 -1801.43535754 -1802.91974501 -1813.90373551 -1817.47841715
 -1811.7741992  -1818.68152526 -1797.41786326 -1779.65301451
 -1807.30904549 -1801.02532463 -1847.62504192 -1799.35224198
 -1808.15080278 -1802.0741555  -1801.50969536 -1785.34981503
 -1768.12690463 -1811.52543123 -1783.40651264 -1823.46655199
 -1794.73174798 -1799.98948466 -1792.5077051  -1786.68510081
 -1798.9196636  -1797.45253317 -1798.56416061 -1767.83960064
 -1786.89974564 -1821.17682027 -1818.93797316 -1793.68349987
 -1800.65801214 -1795.13537647 -1798.99118181 -1792.15484485
 -1804.43029176 -1791.11424184 -1815.89547451 -1796.7965484
 -1810.11482093 -1796.83816572 -1785.4341802  -1780.897105
 -1803.48630556 -1802.5012313  -1804.79670881 -1800.19344867
 -1855.88217068 -1775.4280232  -1770.22465908 -1777.50726585
 -1814.53822569 -1788.24376362 -1772.74870539 -1795.0438406
 -1789.41066666 -1794.99209718 -1812.93941245 -1812.26773187
 -1785.89973641 -1830.78203711 -1795.9437711  -1818.41769699]
Log prob (Normal, 10): [1.61858774 1.61863863 1.6185523  1.61855399 1.6185564  1.61863912
 1.61863794 1.61864823 1.61859532 1.61855269]
Log prob (Normal, 20): [1.96497369 1.96500308 1.96517343 1.96519767 1.96511413 1.96510754
 1.96519332 1.96521895 1.96515296 1.96521923 1.96518913 1.9651041
 1.96522062 1.96520813 1.96520442 1.9650086  1.9650886  1.96515554
 1.96520288 1.96517336]
Log prob (Normal, 100): [2.76938097 2.7699006  2.76885587 2.76993153 2.7699409  2.76968374
 2.76969768 2.76992334 2.7694816  2.76986591 2.76893642 2.76991714
 2.76976709 2.76985394 2.76948302 2.76962244 2.76993866 2.76883321
 2.76984804 2.76970805 2.76965851 2.76993755 2.76841595 2.76991227
 2.76948656 2.76843988 2.76970119 2.76992191 2.76983157 2.76980018
 2.7685402  2.76711177 2.76877971 2.76965324 2.76981159 2.76936942
 2.76865388 2.76993964 2.76719582 2.76993396 2.76991785 2.76881932
 2.76905104 2.76993528 2.76938587 2.7699354  2.76992854 2.76976867
 2.76993591 2.76978419 2.76987361 2.76992571 2.76732822 2.76976752
 2.76994089 2.7699     2.7695302  2.76946241 2.76813544 2.76973627
 2.7697832  2.76928149 2.76988913 2.7684726  2.76972798 2.76944377
 2.7682472  2.76924213 2.76993432 2.76978226 2.76985022 2.76976763
 2.76991143 2.76992204 2.76923295 2.76992974 2.7697055  2.76992272
 2.76994092 2.76923715 2.7681105  2.76979031 2.76960019 2.76931708
 2.76955025 2.76994086 2.76925937 2.76847996 2.76981972 2.76972768
 2.76905016 2.76940433 2.7699335  2.76949082 2.76981477 2.76882652
 2.76892806 2.769915   2.76892769 2.76944613]
Log prob (Poisson, 10): [1.61862417 1.61785457 1.61858213 1.61808233 1.61862238 1.61814034
 1.61864834 1.61785961 1.61729042 1.61600337]
Log prob (Poisson, 20): [1.96451636 1.96350062 1.96239956 1.96482254 1.96424582 1.96522118
 1.96395859 1.96511699 1.96491726 1.96498771 1.96495982 1.96470931
 1.96519374 1.96513902 1.96243974 1.96484742 1.96448236 1.96457555
 1.96363436 1.96453638]
Log prob (Poisson, 100): [2.7637582  2.7692638  2.75366884 2.76994084 2.76986892 2.744131
 2.7698538  2.7691086  2.76677284 2.76801892 2.769249   2.7680926
 2.76912484 2.75783524 2.76992524 2.76362404 2.7593994  2.76990892
 2.75904804 2.7671434  2.76949092 2.75473324 2.7654474  2.76674092
 2.7591658  2.76956004 2.76469804 2.76966244 2.76786724 2.767683
 2.76746284 2.7680926  2.769571   2.76477964 2.7694666  2.7698826
 2.7693214  2.76984844 2.767683   2.76869092 2.76835044 2.7681886
 2.7694666  2.76867084 2.76449124 2.7698254  2.76749092 2.76641292
 2.7682122  2.75951524 2.76914092 2.7696334  2.76765604 2.7698986
 2.76949092 2.7695146  2.76921892 2.76895524 2.76984844 2.76816484
 2.7698826  2.7690922  2.76816484 2.76956004 2.76983724 2.76765604
 2.76498084 2.763073   2.76981924 2.7693898  2.76825892 2.769571
 2.76436524 2.76773644 2.76986404 2.76765604 2.76654604 2.7686506
 2.76993804 2.7693898  2.7699358  2.7659626  2.73778444 2.75739564
 2.76556044 2.76749092 2.75445292 2.76970764 2.76705292 2.75802124
 2.76786724 2.768827   2.76860964 2.76749092 2.76871084 2.7687502
 2.769681   2.7690078  2.76886444 2.7691086 ]
Log prob (Uniform, 10): [1.6186097  1.61860219 1.61860837 1.61864797 1.61864738 1.61857861
 1.61850025 1.6185966  1.61857682 1.61864816]
Log prob (Uniform, 20): [1.96519247 1.96514552 1.96508045 1.96521921 1.96520959 1.96462259
 1.96521609 1.96507242 1.96517716 1.96504853 1.96522137 1.96519003
 1.96507293 1.96451099 1.96465403 1.96465099 1.96521525 1.96521964
 1.96480939 1.96515989]
Log prob (Uniform, 100): [2.76913085 2.76635209 2.76954787 2.76959623 2.76988388 2.76969082
 2.76989095 2.76955486 2.76994066 2.76962059 2.76971558 2.76994052
 2.76992666 2.76785957 2.76772298 2.76992641 2.76866023 2.76993034
 2.76790633 2.76994091 2.76978307 2.76975648 2.76988901 2.7693946
 2.76992077 2.7681122  2.7696879  2.76993846 2.76927351 2.76992721
 2.76993337 2.76993421 2.76976242 2.76983743 2.76993546 2.76965834
 2.76984021 2.76799102 2.76925427 2.76706893 2.76962372 2.76948641
 2.76889429 2.76978122 2.76950468 2.76984577 2.76903476 2.76781396
 2.76976666 2.76960204 2.7698736  2.76933795 2.76989737 2.76992892
 [illegible]
 2.76885322 2.76641264 2.76904662 2.76830496 2.76993476 2.7699187
 2.7698782  2.76723167 2.76947954 2.76959493 2.76978684 2.76964926
 2.76957213 2.76993344 2.76972708 2.76991687 2.76951638 2.76993639
 2.76985765 2.76985552 2.76969096 2.76910019 2.7685515  2.7696531
 2.76980401 2.76994069 2.76961804 2.76932772 2.76991383 2.76950898
 2.76983728 2.765728   2.76954672 2.76993828 2.76989277 2.76918198
 2.76976171 2.76988308 2.76993322 2.76991812 2.76993989 2.76985222
 2.76955557 2.76960484 2.76968418 2.76991658]
Log prob (Bernoulli, 10): [1.61854802 1.61862505 1.61864117 1.61817209 1.61863801 1.61770169
 1.61809081 1.61782917 1.61863621 1.6184755 ]
Log prob (Bernoulli, 20): [1.96449606 1.96522139 1.96319419 1.96474862 1.9652135  1.96514131
 1.96517531 1.96519099 1.96518035 1.96516998 1.96504891 1.96503534
 1.96407278 1.96283803 1.96522171 1.96517702 1.96505222 1.9650683
 1.96479686 1.96422196]
Log prob (Bernoulli, 100): [2.76961324 2.7663114  2.7655978  2.7682122  2.769681   2.76641292
 2.75745892 2.76974092 2.76848292 2.76574564 2.76905892 2.7682122
 2.76980644 2.76899044 2.765217   2.75714092 2.76984292 2.76993892
 2.7696234  2.76876964 2.7695818  2.76992292 2.7678154  2.7659626
 2.7689374  2.76846124 2.76567204 2.75963044 2.76942892 2.76984292
 2.7697642  2.76654604 2.76984844 2.766961   2.766961   2.7690078
 2.7693898  2.7696334  2.76195244 2.75549092 2.7686302  2.76825892
 2.76620844 2.769571   2.7666442  2.767683   2.7697246  2.76762892
 2.76816484 2.7652558  2.7680682  2.76814092 2.76835044 2.7610698
 2.769915   2.76927844 2.76825892 2.7694666  2.76984844 2.7699358
 2.76624292 2.7688458  2.76737764 2.7696234  2.7699406  2.76878892
 2.76984292 2.76804364 2.7527914  2.76689892 2.7663114  2.76941604
 2.76765604 2.7696718  2.7697566  2.7674062  2.76984292 2.769249
 2.769025   2.76453292 2.7698826  2.7687306  2.76961324 2.7698538
 2.7685258  2.767969   2.76760164 2.76644644 2.76993292 2.7688458
 2.7691086  2.7588702  2.76988684 2.7659626  2.7699274  2.76804364
 2.7698782  2.7699358  2.76989092 2.76552292]
```

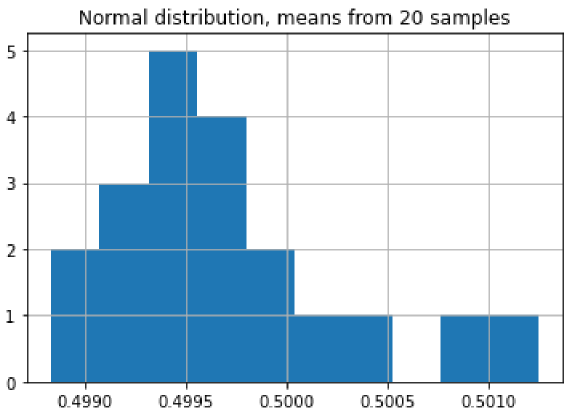


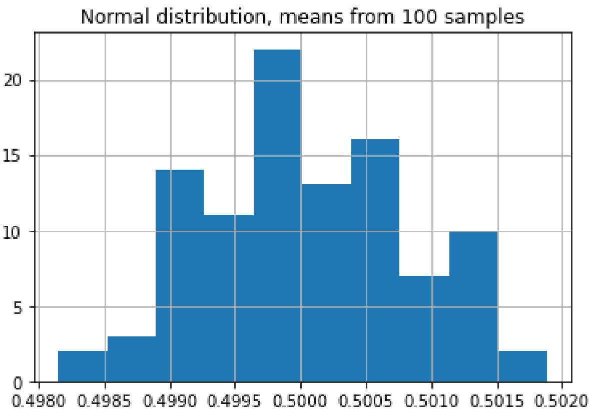


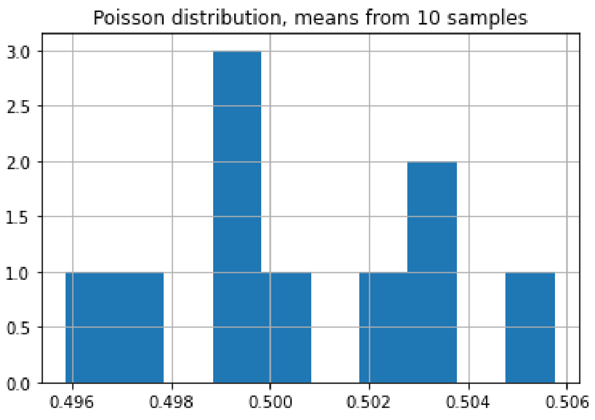


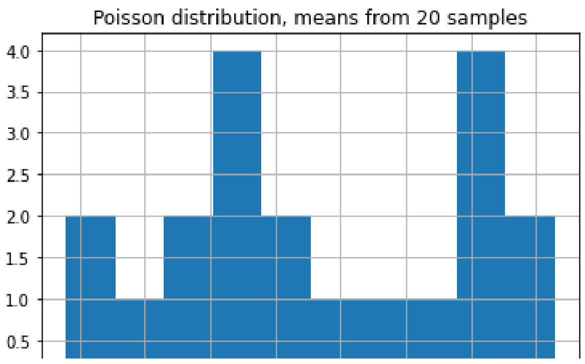
1.990
1.995
2.000
2.005
2.010
2.015
2.020
2.025

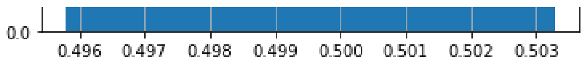

Normal distribution, means from 10 samples
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.4990
0.4995
0.5000
0.5005
0.5010

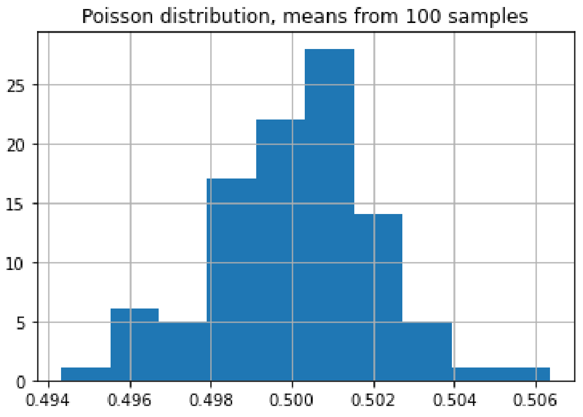

Normal distribution, means from 20 samples
0
1
2
3
4
5
0.4990
0.4995
0.5000
0.5005
0.5010

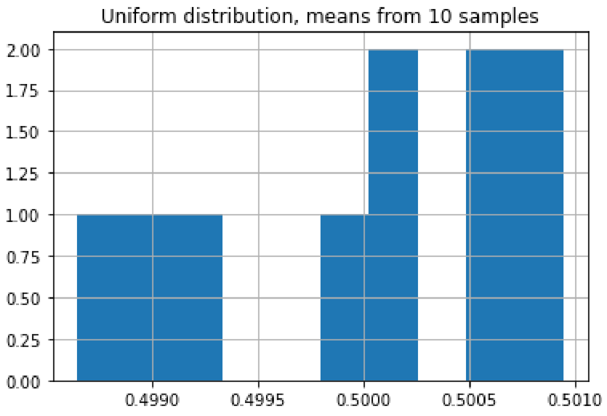

Normal distribution, means from 100 samples
0
5
10
15
20
0.4980
0.4985
0.4990
0.4995
0.5000
0.5005
0.5010
0.5015
0.5020

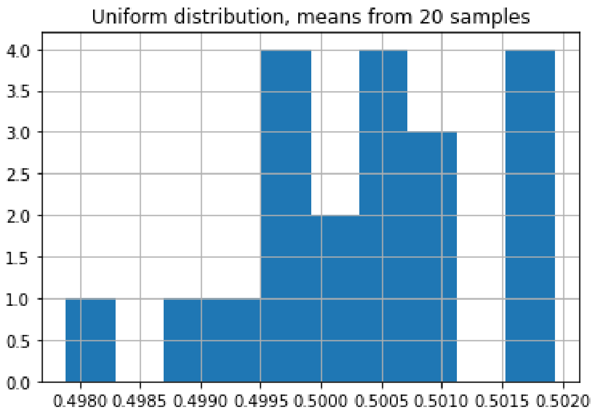

Poisson distribution, means from 10 samples
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.496
0.498
0.500
0.502
0.504
0.506

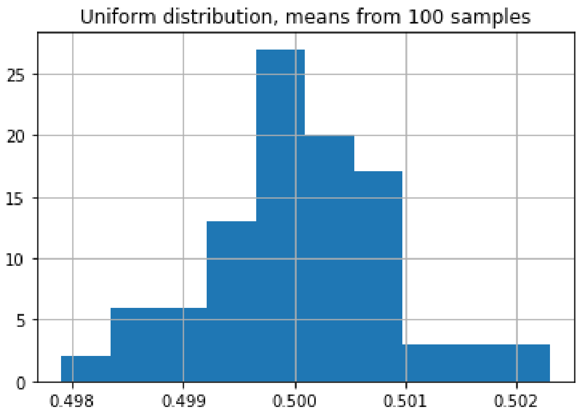

Poisson distribution, means from 20 samples
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

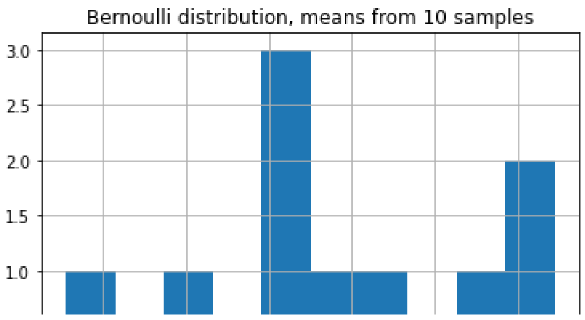

0.0
0.496
0.497
0.498
0.499
0.500
0.501
0.502
0.503

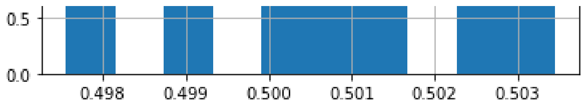

Poisson distribution, means from 100 samples
0
5
10
15
20
25
0.494
0.496
0.498
0.500
0.502
0.504
0.506

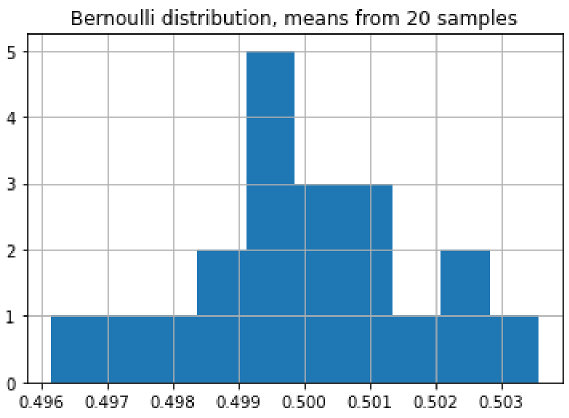

Uniform distribution, means from 10 samples
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.4990
0.4995
0.5000
0.5005
0.5010

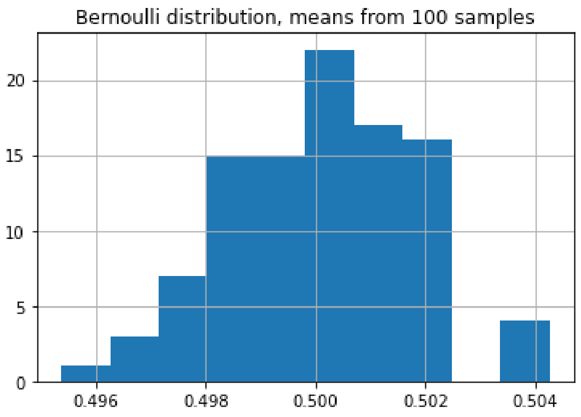

Uniform distribution, means from 20 samples
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.4980
0.4985
0.4990
0.4995
0.5000
0.5005
0.5010
0.5015
0.5020


Uniform distribution, means from 100 samples
0
5
10
15
20
25
0.498
0.499
0.500
0.501
0.502


Bernoulli distribution, means from 10 samples
1.0
1.5
2.0
2.5
3.0







5.

My research problem would be this: Is there association between the spatial segregation of populations and wealth? I would like inspect the connection between ethnicity and wealth by taking into account spatial clusters of ethnic groups and high income.

The generative process might work like this:

- certain ethnic groups are more likely to be wealthy
- wealthy people tend to live in better parts of the town
- therefore, income segregation causes spatial segregation
- however, there are other processes that may cause the spatial segregation of ethnic groups

My ultimate aim is to test empirically whether income differences explain all of the ethnic segregation.

Latent variables that might be needed for modelling:

- the other, hidden variables that affect segregation