privacy

December 9, 2021

[ ]:

[5]:
```
from covid.simulator import Population
from covid.auxilliary import symptom_names
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from covid.policy import Policy
```

[6]:
```
## Baseline simulator parameters
n_genes = 128
n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
n_treatments = 4
n_population = 10000
n_symptoms = 10
#batch_size = 2000

#assert n_population/batch_size == n_population//batch_size, 'the batch size must evenly divide the number of people'
```

[7]:
```
population = Population(n_genes, n_vaccines, n_treatments)
```

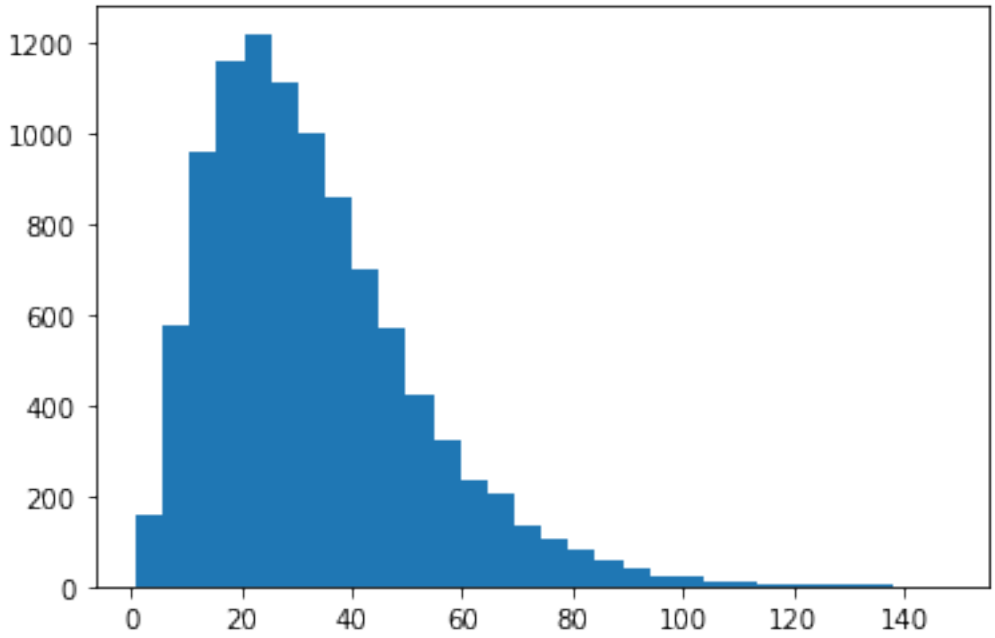
[8]:
```
X = population.generate(n_population)
n_features = X.shape[1]
```

[9]:
```
X
```

[9]:
```
array([[0., 1., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

[10]:
```
plt.hist(X[:,10], bins=30)
```

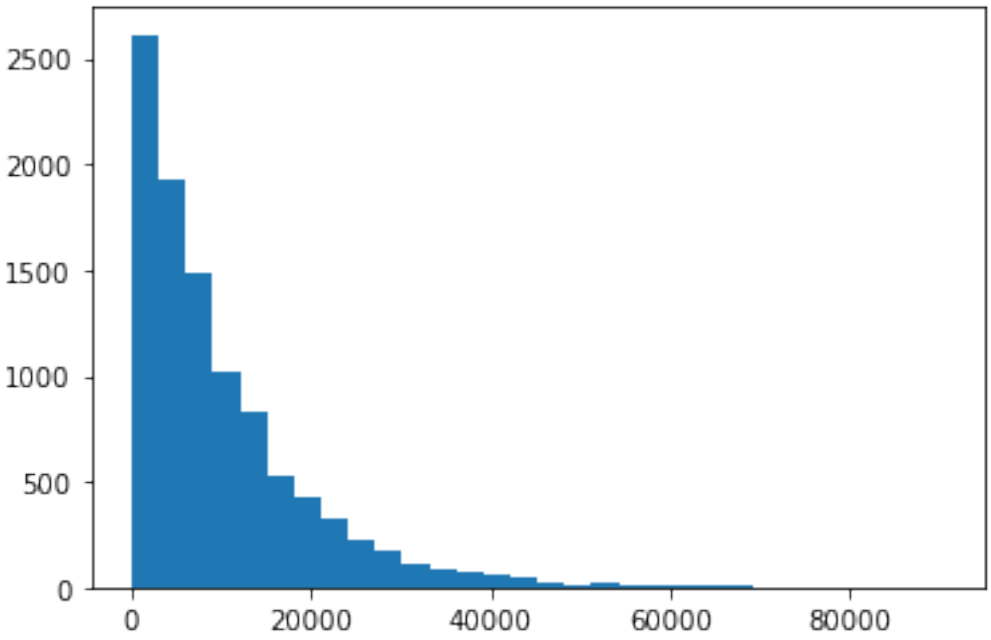
[10]:
```
(array([1.570e+02, 5.770e+02, 9.610e+02, 1.158e+03, 1.220e+03, 1.111e+03,
        1.001e+03, 8.600e+02, 7.000e+02, 5.730e+02, 4.230e+02, 3.210e+02,
        2.330e+02, 2.050e+02, 1.350e+02, 1.020e+02, 8.300e+01, 5.800e+01,
        3.900e+01, 2.000e+01, 2.300e+01, 1.300e+01, 1.100e+01, 5.000e+00,
        2.000e+00, 2.000e+00, 2.000e+00, 4.000e+00, 0.000e+00, 1.000e+00]),
 array([  0.87620595,   5.77366804,  10.67113012,  15.56859221,
         20.46605429,  25.36351638,  30.26097846,  35.15844055,
         40.05590263,  44.95336472,  49.8508268 ,  54.74828889,
         59.64575097,  64.54321306,  69.44067514,  74.33813723,
         79.23559931,  84.1330614 ,  89.03052348,  93.92798557,
         98.82544765, 103.72290973, 108.62037182, 113.5178339 ,
        118.41529599, 123.31275807, 128.21022016, 133.10768224,
        138.00514433, 142.90260641, 147.8000685 ]),
 <BarContainer object of 30 artists>)
```



[11]:
```
plt.hist(X[:,12], bins=30)
```

[11]:
```
(array([2.612e+03, 1.925e+03, 1.480e+03, 1.015e+03, 8.310e+02, 5.210e+02,
        4.210e+02, 3.240e+02, 2.220e+02, 1.750e+02, 1.100e+02, 9.200e+01,
        6.900e+01, 5.900e+01, 4.400e+01, 2.500e+01, 1.200e+01, 1.800e+01,
        8.000e+00, 7.000e+00, 9.000e+00, 7.000e+00, 6.000e+00, 2.000e+00,
        1.000e+00, 3.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 2.000e+00]),
 array([2.50887924e-01, 3.01226853e+03, 6.02428617e+03, 9.03630381e+03,
        1.20483214e+04, 1.50603391e+04, 1.80723567e+04, 2.10843744e+04,
        2.40963920e+04, 2.71084096e+04, 3.01204273e+04, 3.31324449e+04,
        3.61444626e+04, 3.91564802e+04, 4.21684978e+04, 4.51805155e+04,
```

```
        4.81925331e+04, 5.12045508e+04, 5.42165684e+04, 5.72285860e+04,
        6.02406037e+04, 6.32526213e+04, 6.62646390e+04, 6.92766566e+04,
        7.22886742e+04, 7.53006919e+04, 7.83127095e+04, 8.13247272e+04,
        8.43367448e+04, 8.73487624e+04, 9.03607801e+04]),
 <BarContainer object of 30 artists>)
```



[12]:
```python
def u(x, value):

    v_cnts = np.unique(x, return_counts=True)

    return v_cnts[1][value] / v_cnts[1].sum()

def exponential(x, R, u, sensitivity, epsilon, n=1):
    scores = u(x, R) # score each element in R
    probs = np.exp(epsilon*scores / 2 / sensitivity)
    probs /= probs.sum()
    return np.random.choice(R, n, p=probs)

X_new = np.zeros((n_population, n_features))

for i in range(n_features):
    leng = len(np.unique(X[:,i].astype(int)))
    e = exponential(X[:,i].astype(int), np.arange(0,leng), u, 1, 100,␣
 ↪n=n_population)
```

```
    X_new[:,i] = e

print(np.shape(X))
print(np.shape(X_new))
```

```
(10000, 150)
(10000, 150)
```

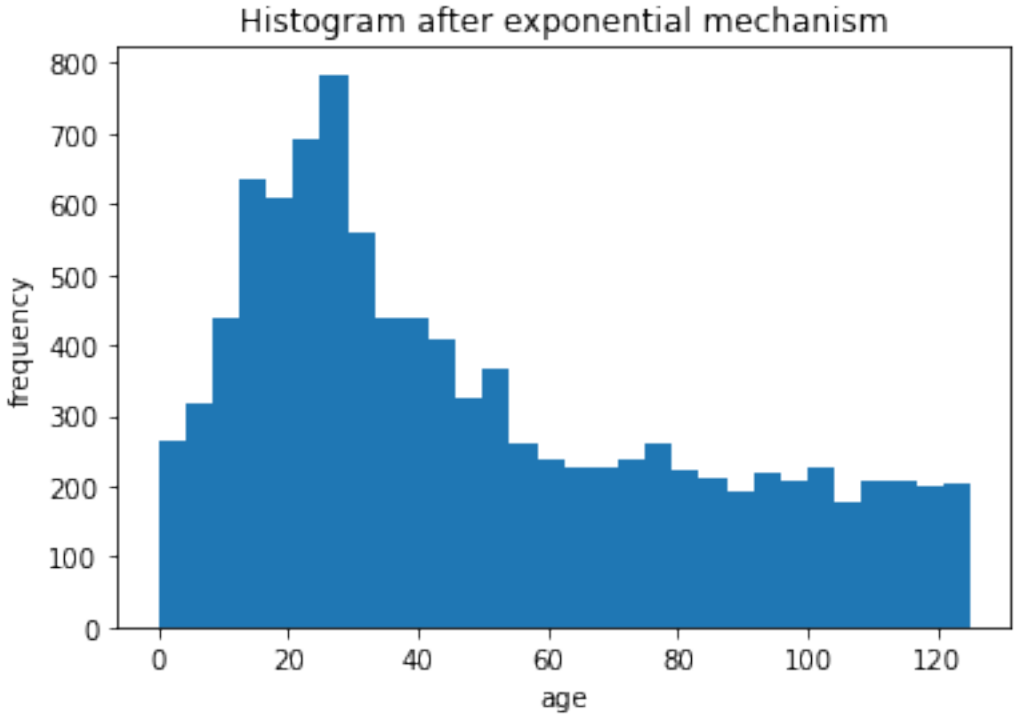
[13]:
```
def exp_data(data, epsilon):
    n_features = data.shape[1]

    for i in range(n_features):
        leng = len(np.unique(data[:,i].astype(int)))
        e = exponential(data[:,i].astype(int), np.arange(0,leng), u, 1,␣
↪epsilon, n=n_population)

        X_new[:,i] = e

    return X_new
```

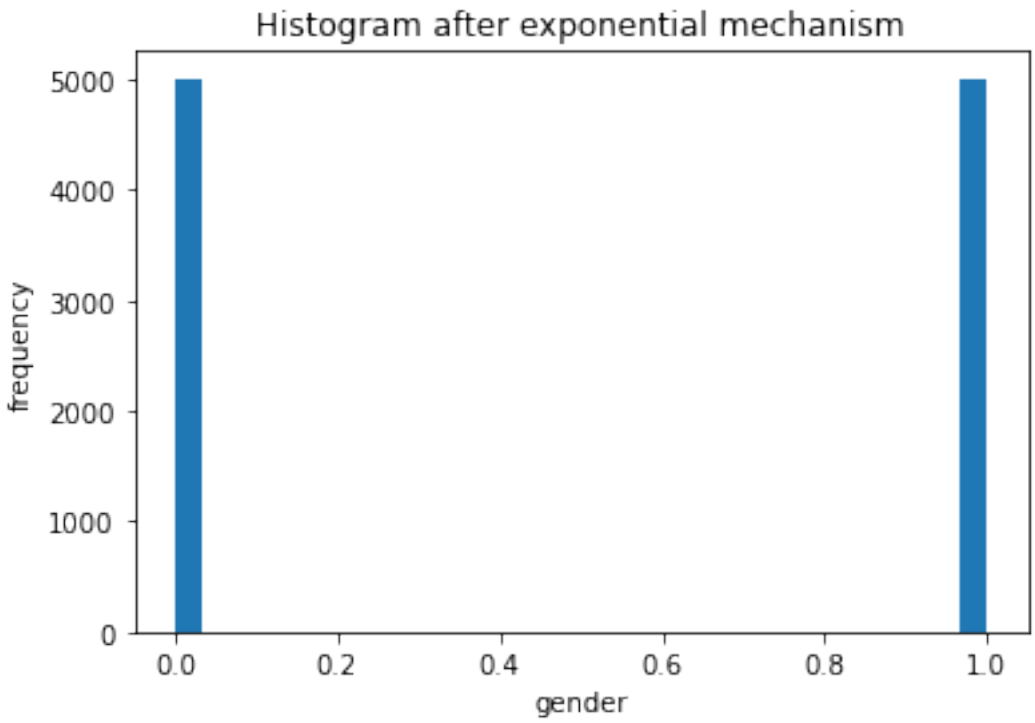
[33]:
```
varia = ['age', 'gender','income']
var_num = [10,11,12]
```

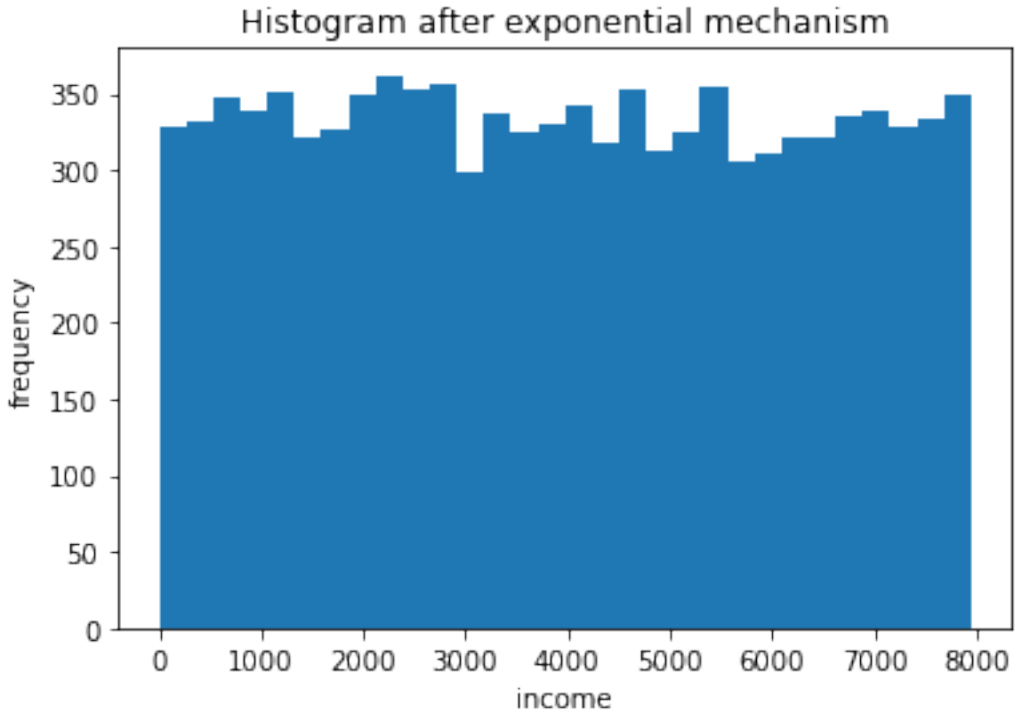
[34]:
```
def plot_expo(data, varia, var_num):
    for i in range(len(varia)):
        plt.hist(data[:,var_num[i]],bins=30)
        plt.title("Histogram after exponential mechanism")
        plt.xlabel(varia[i])
        plt.ylabel("frequency")
        plt.savefig('figures/histogram_exp_mech_' + varia[i] + '.png')
        plt.show()
```

[35]:
```
X_exp = exp_data(X, epsilon=100)
plot_expo(X_exp, varia, var_num)


"""
plot_expo(X_exp, feature=10)
plt.xlabel(varia[i])
plt.savefig('figures/hist_exp_mech_age.png')
plt.show()
plot_expo(X_exp, feature=11)
plt.savefig('figures/hist_exp_mech_gender.png')
plt.show()
plot_expo(X_exp, feature=12)
```

```
plt.savefig('figures/hist_exp_mech_salary.png')
plt.show()
"""
```




Histogram after exponential mechanism
frequency
gender
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0

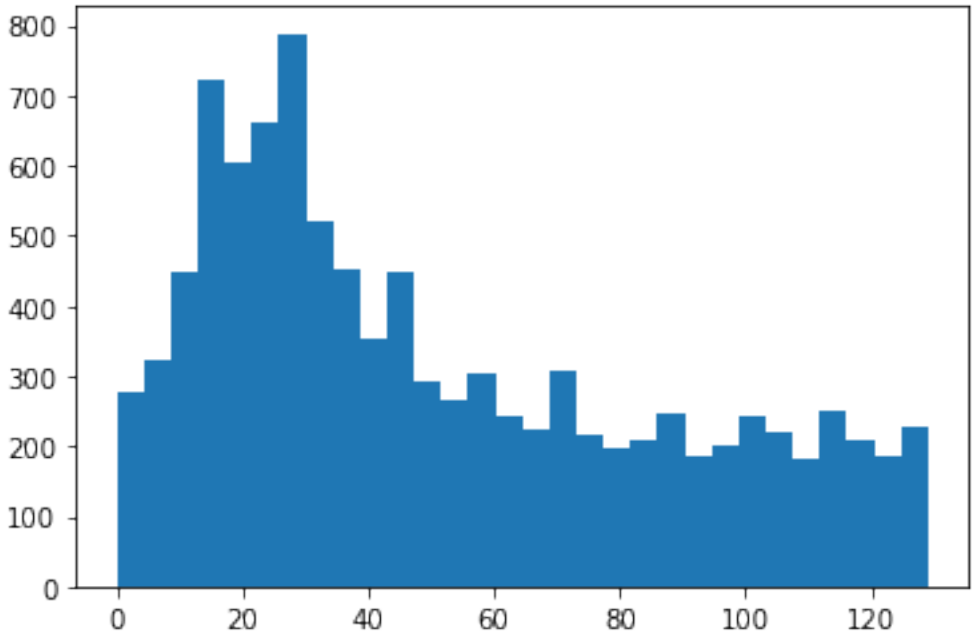

Histogram after exponential mechanism
frequency
income
0
50
100
150
200
250
300
350
0
1000
2000
3000
4000
5000
6000
7000
8000

[35]:
```
"\nplot_expo(X_exp, feature=10)\nplt.xlabel(varia[i])\nplt.savefig('figures/hist
_exp_mech_age.png')\nplt.show()\nplot_expo(X_exp, feature=11)\nplt.savefig('figu
res/hist_exp_mech_gender.png')\nplt.show()\nplot_expo(X_exp,
feature=12)\nplt.savefig('figures/hist_exp_mech_salary.png')\nplt.show()\n"
```

[ ]:
```
```

[302]:
```
plt.hist(X_new[:,10],bins=30)
```

[302]:
```
(array([276., 321., 447., 723., 603., 661., 788., 520., 451., 355., 450.,
        291., 265., 305., 244., 223., 309., 216., 198., 208., 248., 185.,
        201., 241., 220., 183., 249., 207., 184., 228.]),
 array([  0. ,   4.3,   8.6,  12.9,  17.2,  21.5,  25.8,  30.1,  34.4,
         38.7,  43. ,  47.3,  51.6,  55.9,  60.2,  64.5,  68.8,  73.1,
         77.4,  81.7,  86. ,  90.3,  94.6,  98.9, 103.2, 107.5, 111.8,
        116.1, 120.4, 124.7, 129. ]),
 <BarContainer object of 30 artists>)
```



[ ]:
```
```

2b

December 9, 2021

[1]:
```
from covid.simulator import Population
from covid.auxilliary import symptom_names
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from covid.policy import Policy
```

[5]:
```
## Baseline simulator parameters
n_genes = 128
n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
n_treatments = 4
n_population = 10_000
n_symptoms = 10
batch_size = 2000

assert n_population/batch_size == n_population//batch_size, 'the batch size␣
 ↪must evenly divide the number of people'


# symptom names for easy reference
from covid.auxilliary import symptom_names
```

[6]:
```
population = Population(n_genes, n_vaccines, n_treatments)
```

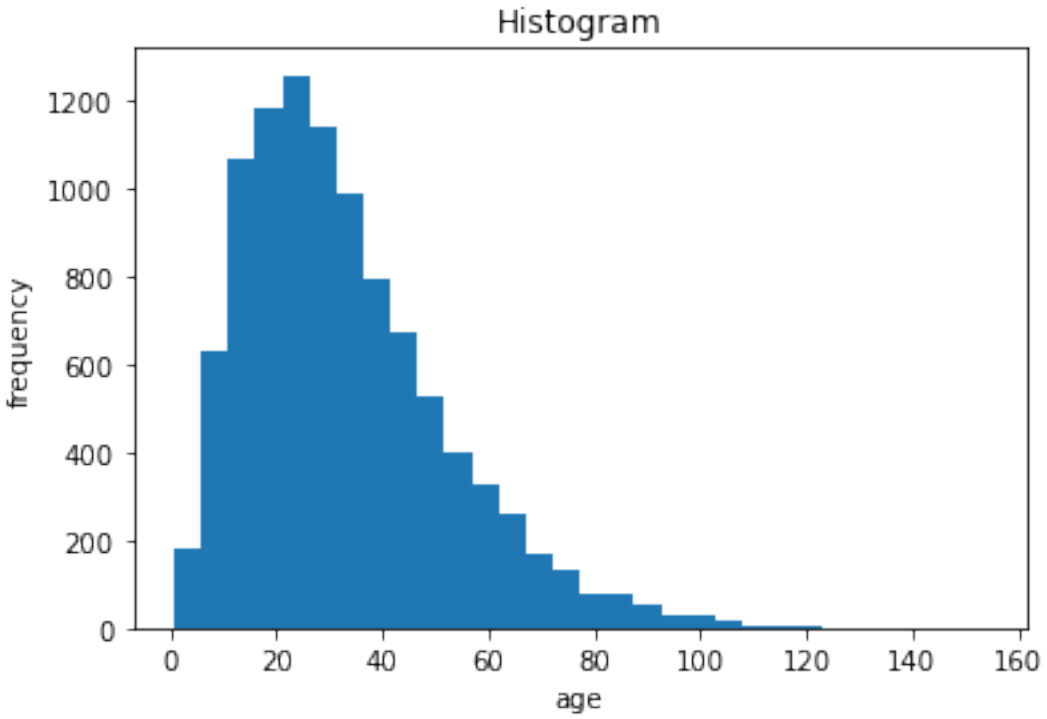
[7]:
```
X = population.generate(n_population)
n_features = X.shape[1]
```

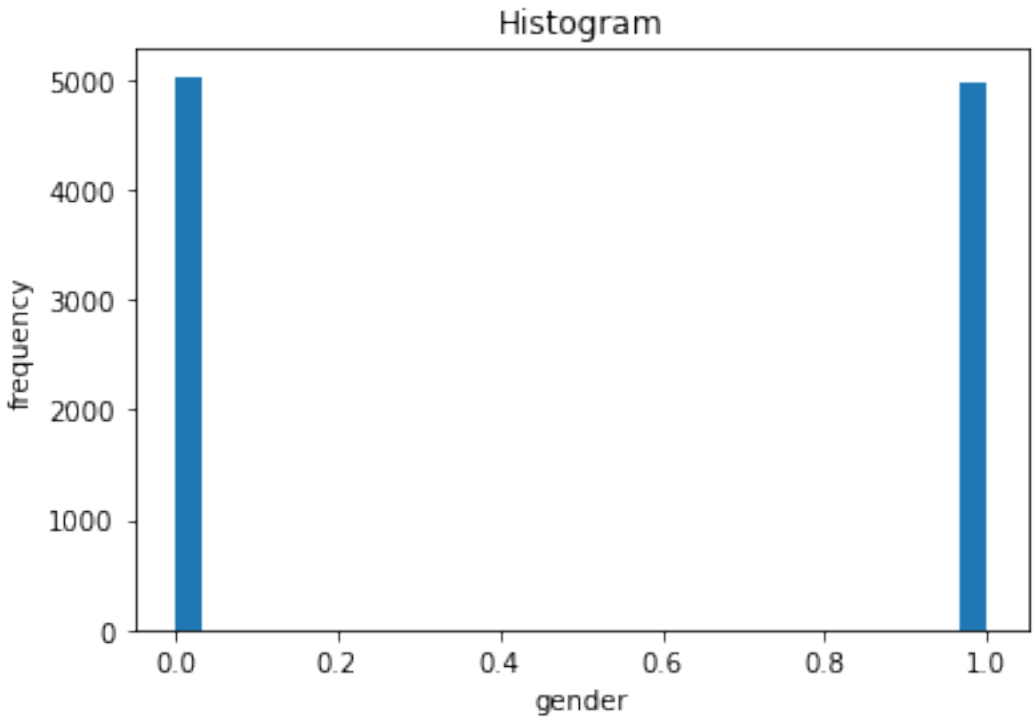
[8]:
```
varia = ['age', 'gender','income']
var_num = [10,11,12]
```

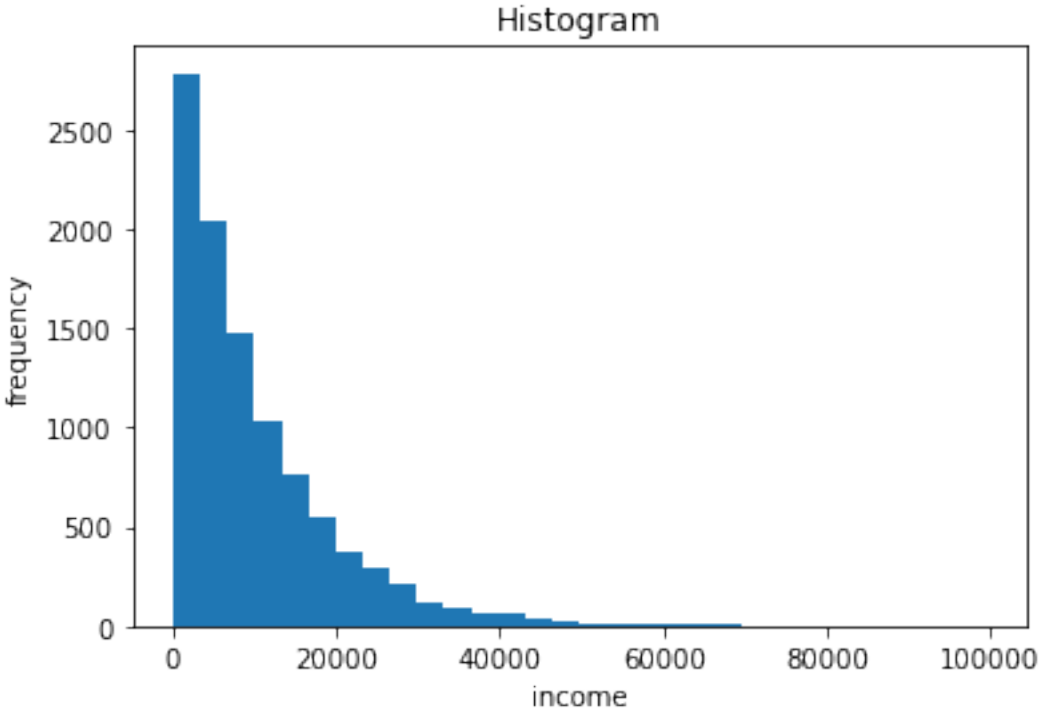
[43]:
```
def hist_data(data, varia,var_num):
    for i in range(len(varia)):
        plt.hist(data[:,var_num[i]], bins=30)
        plt.title("Histogram")
        plt.xlabel(varia[i])
        plt.ylabel("frequency")
```

```
        plt.savefig('figures/'+ varia[i] + '_histogram.png')
        plt.show()
```

[44]:
```
hist_data(data=X, varia=varia, var_num=var_num)
```



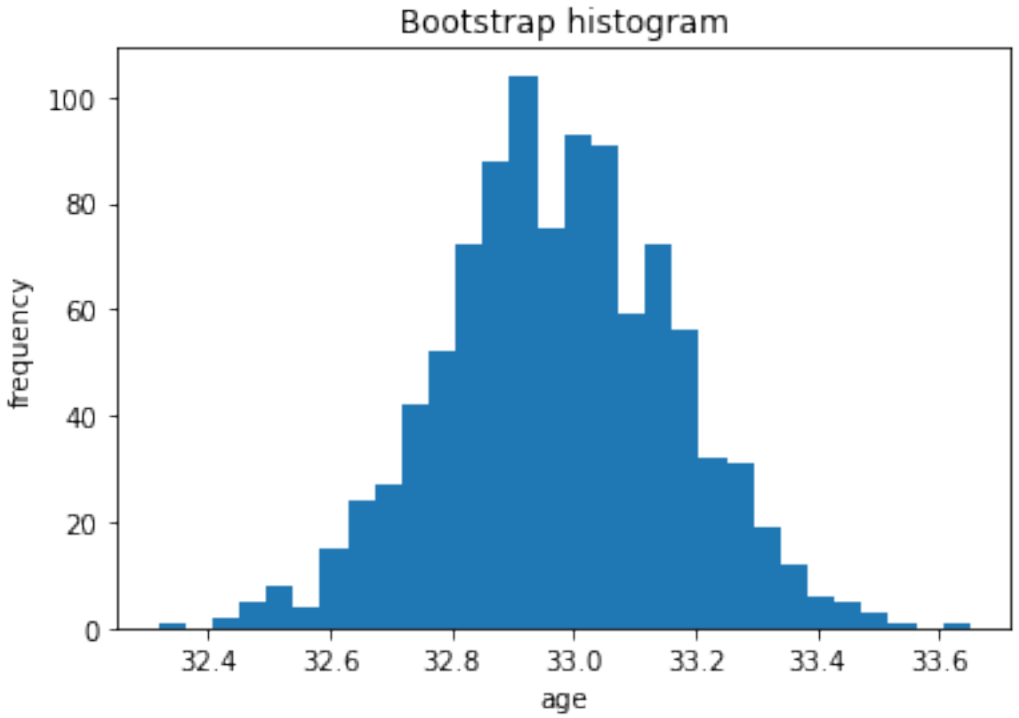

Histogram
5000
4000
3000
2000
1000
0
frequency
0.0
0.2
0.4
0.6
0.8
1.0
gender

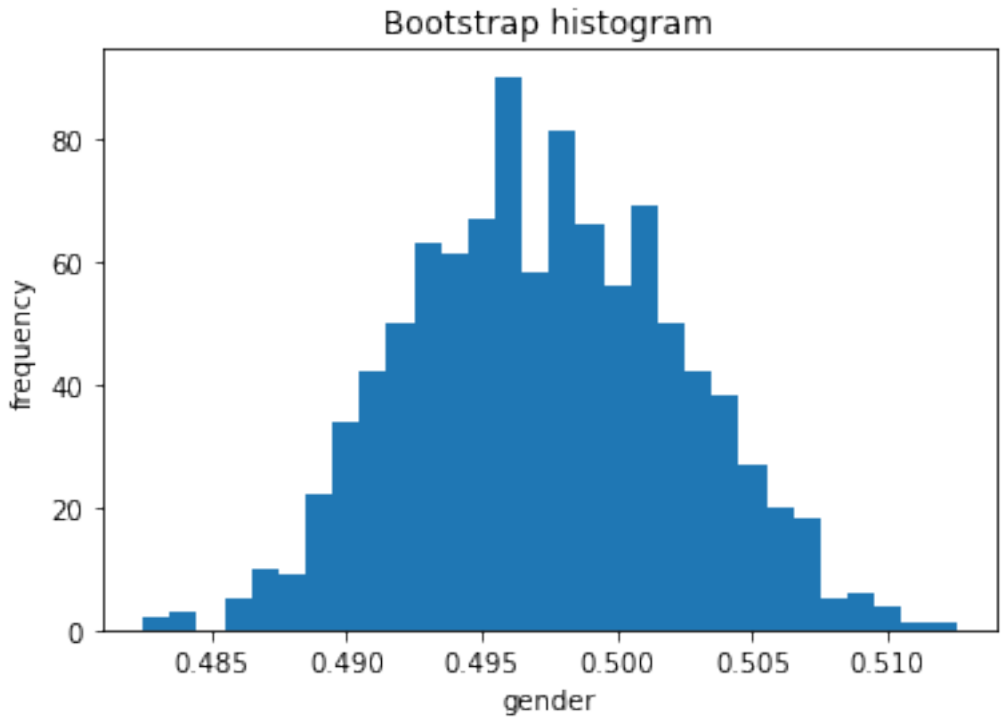

Histogram
2500
2000
1500
1000
500
0
frequency
0
20000
40000
60000
80000
100000
income

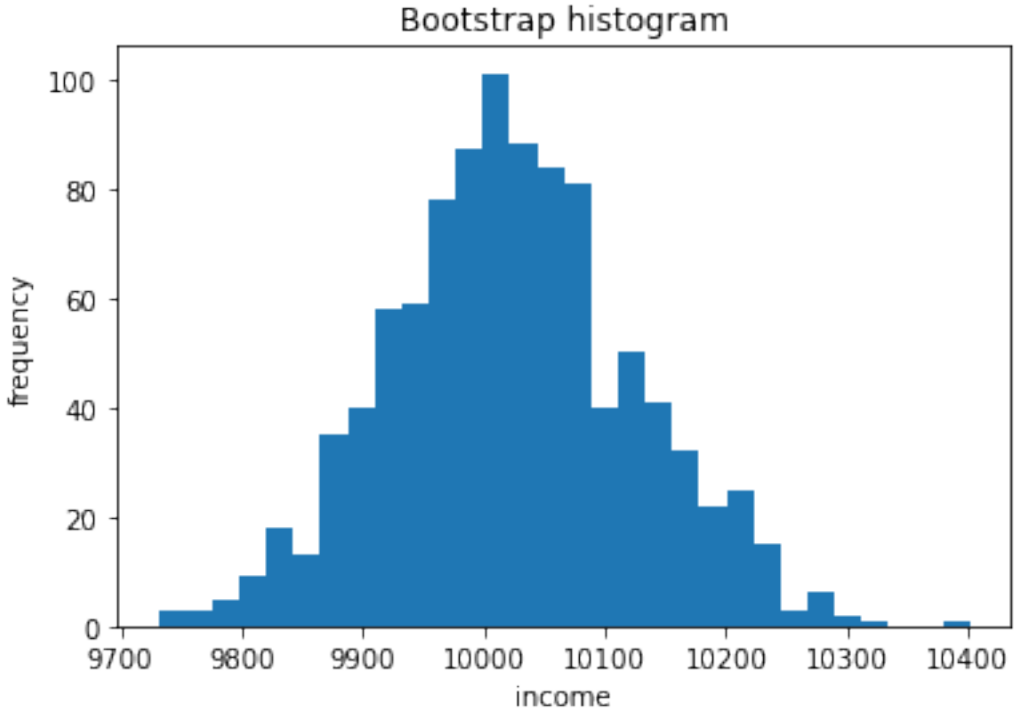
[ ]:

[41]:
```
def boot_var(data, varia, var_num, B):

    for i in range(len(varia)):
        var_mean = []
        for _ in range(B):
            X_boot = np.random.choice(data[:,var_num[i]],replace=True,␣
↪size=data.shape[0])
            var_mean.append(np.mean(X_boot))

        plt.hist(var_mean, bins=30)
        plt.title("Bootstrap histogram")
        plt.xlabel(varia[i])
        plt.ylabel("frequency")
        plt.savefig('figures/bootstrap' + varia[i] + '.png')
        plt.show()
```

[42]:
```
boot_var(data=X, varia=varia, var_num=var_num, B=1000)
```



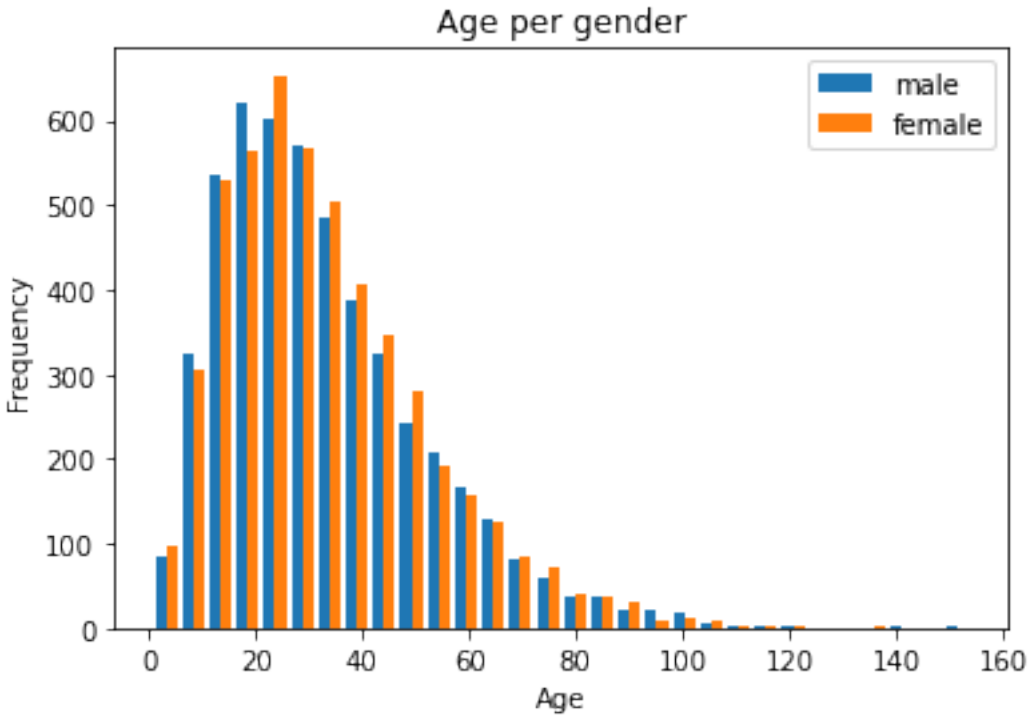

Bootstrap histogram
frequency
0
20
40
60
80
0.485
0.490
0.495
0.500
0.505
0.510
gender


Bootstrap histogram
frequency
0
20
40
60
80
100
9700
9800
9900
10000
10100
10200
10300
10400
income

[91]:
```
###
```

[34]:
```
def plot_age_gender(data):
    mm = np.where(data[:,11]==1) #male
    ff = np.where(data[:,11]==0) #female

    sal_m = np.take(data[:,10], mm)
    sal_m = sal_m.flatten()

    sal_f = np.take(data[:,10], ff)
    sal_f = sal_f.flatten()
    plt.hist([sal_m,sal_f],bins=30, label=['male','female'])
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.title('Age per gender')
    plt.legend(loc='upper right')
    plt.savefig('figures/age_per_gender.png')
    plt.show()
```

[35]:
```
plot_age_gender(X)
```

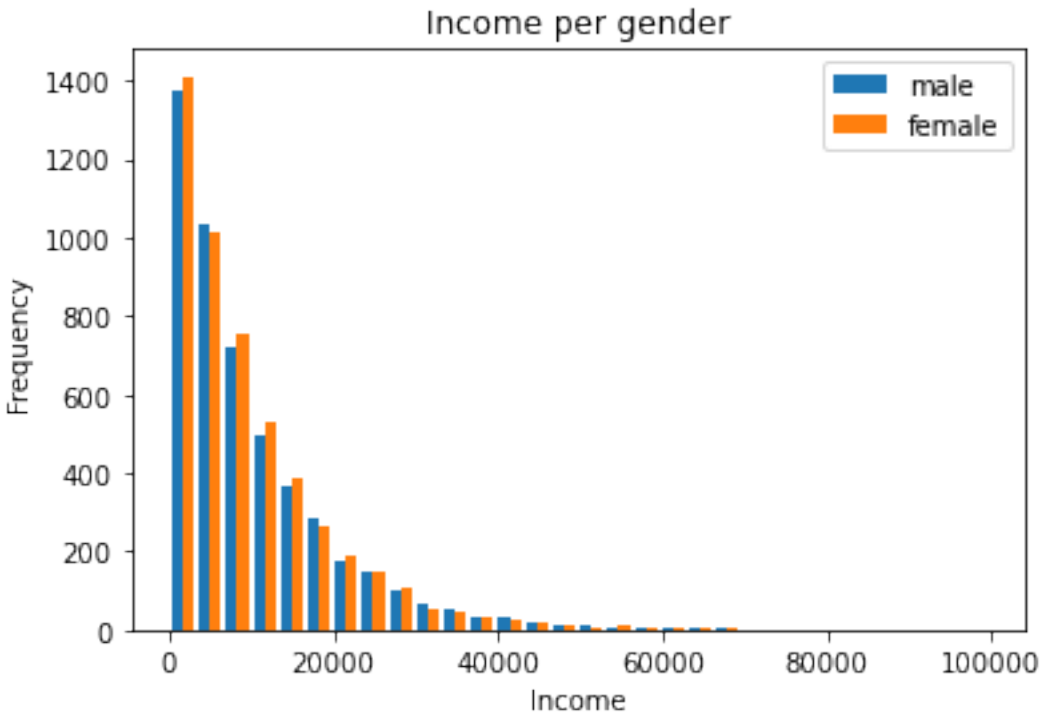


[24]:
```

```

[ ]:

[39]:
```
def plot_salary_gender(data):
    mm = np.where(data[:,11]==1) #male
    ff = np.where(data[:,11]==0) #female

    inc_m = np.take(data[:,12], mm)
    inc_m = inc_m.flatten()

    inc_f = np.take(data[:,12], ff)
    inc_f = inc_f.flatten()
    plt.hist([inc_m,inc_f],bins=30, label=['male','female'])
    plt.xlabel('Income')
    plt.ylabel('Frequency')
    plt.title('Income per gender')
    plt.legend(loc='upper right')
    plt.savefig('figures/salary_per_gender.png')
    plt.show()
```

[40]:
```
plot_salary_gender(X)
```



[ ]:

[ ]:

From the first plots we look at the distribution of some of the variables that could tell us aloot about the population. We see that the age of our data is centered between 20-50 years old, which can be a realistic assumption. One problem os however that there is not that much young people, and we also have some very old people (200 years old) which is not realsitic. We also see that our data is balanced in respect to the genders. When it comes to the income of our data, this also looks to reflect a general population.

We also bootsrap our data to see if there is big variation, but it doesnt look to be any big variation in the data.

In the report we also want to mention that the histogram of age is very unreasitically. The birth rate has dropped exponentially in a few years and people die with exponential rate after above 25 years.

Info from office hours: * How does the original trainig data affect the fairness of your policy * Perhaps look at unfair points in the simulator. * Do the original features already imply some biased in data collection: * Strange variabls * More relevant from earlier task * Unbalanced sample * Analysis of the decision function: * How are the actions distributed. * F.ex age, how are the vaccine rate among young and old people

1 IMPORTANT

- Chrisos said equality of oppurtunity score is not a good critiria.
  - P(a|x,?) = P(a|x)

functions

December 9, 2021

1 Functions for plotting and outputs

1.0.1 Comment

This file originally contained both functions and scripts.

To simplify we put all the code into functions. Some of then doesn't run since some code inbetween was remove.

[39]:
```
from covid.simulator import Population
from covid.auxilliary import symptom_names
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from covid.policy import Policy
from scipy.stats import beta, bernoulli, uniform
```

[40]:
```
class ThompsonSampling:
    """Thompson sampling"""
    def __init__(self, nvacc):
        # Priors for the beta-bernoulli model
        self.a = np.ones(nvacc) # uniform prior
        self.b = np.ones(nvacc) # uniform prior
        self.nvacc = nvacc

    def update(self, action, outcome):
        self.a[action] += outcome
        self.b[action] += 1 - outcome

    def update_with_laplace(self, action, outcome, epsilon=1):
        """Alternative update model with added Laplace noise"""
        outcome = outcome + np.random.laplace(0, 1/epsilon)
        self.a[action] += outcome
        self.b[action] += 1 - outcome

    def get_params(self):
        # Returns the parameters of all the beta distrobutions.
```

```
        return self.a, self.b

    def get_prob(self):
        for i in range(self.nvacc):
            if self.a[i] <= 1:
                self.a[i] = 1
            if self.b[i] <= 1:
                self.b[i] = 1
        return beta.rvs(self.a, self.b)
```

[41]:
```
class Naive(Policy):
    def set_model(self, model):
        self.model = model

    def get_action(self):
        probs = self.model.get_prob()

        #print(probs)
        ret_val = np.argmin(probs)
        #print(ret_val)
        #print(ret_val)
        return ret_val

    def observe(self, action, outcome):
        self.model.update(action, outcome)

    def observe_with_laplace(self, action, outcome, epsilon):
        self.model.update_with_laplace(action, outcome, epsilon)
```

[42]:
```
def compare_laplace():
    """Comparing different privacy guarantees."""

    action_space = np.array([-1,0,1,2])
    n_actions = action_space.shape[0]

    epsilons = [100, 10, 1, 0.1, 0.01, 0.001]

    alphas = np.zeros((len(epsilons),4))
    betas = np.zeros((len(epsilons),4))

    for i in range(len(epsilons)):
        n_genes = 128
        n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
        n_treatments = 4
        n_population = 10_000
        n_symptoms = 10
```

```
        #np.random.seed(2)
        population = Population(n_genes, n_vaccines, n_treatments)
        X = population.generate(n_population)
        n_features = X.shape[1]

        policy = Naive(n_actions, action_space)
        model = ThompsonSampling(n_actions)
        policy.set_model(model)
        action_arr = np.zeros(n_population)
        symptom_arr = np.zeros((n_population, n_symptoms+1))

        for t in range(n_population):
            a_t = policy.get_action()
            y_t = population.vaccinate([t], a_t.reshape((1, 1)))
            new_col = y_t[:,[5,7,8]].sum(axis=1)
            new_col0 = new_col > 0
            new_col01 = new_col0.astype(int)
            new_col01 = np.reshape(new_col01,(1,-1))
            y_t_new = np.hstack((y_t,new_col01))
            policy.observe_with_laplace(a_t, y_t_new[:,10], epsilon=epsilons[i])
            action_arr[t] = a_t
            symptom_arr[t] = y_t_new

        alpha, beta = policy.model.get_params()

        print(i, alpha, beta)
        alphas[i,:] = alpha
        betas[i,:] = beta
    return alphas, betas
```

[43]:
```
def privacy_output():
    alphas_comp, betas_comp = compare_laplace()
    epsilons = [100, 10, 1, 0.1, 0.01, 0.001]
    utility_comp = alphas_comp.sum(axis=1)
    out = f'========= Utility when privacy guarantee changes ======\n'
    out += f'Epsilon: {100:5}, Accumulated utility: {utility_comp[0]:11.1f}\n'
    out += f'Epsilon: {10:5}, Accumulated utility: {utility_comp[1]:11.1f}\n'
    out += f'Epsilon: {1:5}, Accumulated utility: {utility_comp[2]:11.1f}\n'
    out += f'Epsilon: {0.1:5}, Accumulated utility: {utility_comp[3]:11.1f}\n'
    out += f'Epsilon: {0.01:5}, Accumulated utility: {utility_comp[4]:11.1f}\n'
    out += f'Epsilon: {0.001:5}, Accumulated utility: {utility_comp[5]:11.1f}\n'
    out += f'=========================================================\n'
    with open('outputs/utility_comp.txt', 'w') as outfile:
        outfile.write(out)
```

[44]:
```
def expected_utility(samples=10):
    action_space = np.array([-1,0,1,2])
```

```
    n_actions = action_space.shape[0]

    alphas = np.zeros((samples,4))
    betas = np.zeros((samples,4))

    for i in range(samples):
        n_genes = 128
        n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
        n_treatments = 4
        n_population = 10_000
        n_symptoms = 10
        #np.random.seed(2)
        population = Population(n_genes, n_vaccines, n_treatments)
        X = population.generate(n_population)
        n_features = X.shape[1]

        policy = Naive(n_actions, action_space)
        model = ThompsonSampling(n_actions)
        policy.set_model(model)
        action_arr = np.zeros(n_population)
        symptom_arr = np.zeros((n_population, n_symptoms+1))

        for t in range(n_population):
            a_t = policy.get_action()
            y_t = population.vaccinate([t], a_t.reshape((1, 1)))
            new_col = y_t[:,[5,7,8]].sum(axis=1)
            new_col0 = new_col > 0
            new_col01 = new_col0.astype(int)
            new_col01 = np.reshape(new_col01,(1,-1))
            y_t_new = np.hstack((y_t,new_col01))
            policy.observe(a_t, y_t_new[:,10])
            action_arr[t] = a_t
            symptom_arr[t] = y_t_new

        alpha, beta = policy.model.get_params()

        print(i, alpha, beta)
        alphas[i,:] = alpha
        betas[i,:] = beta
    return alphas, betas
```

[45]:
```
def expected_utility_laplace(samples=100):
    action_space = np.array([-1,0,1,2])
    n_actions = action_space.shape[0]

    alphas = np.zeros((samples,4))
    betas = np.zeros((samples,4))
```

```
    for i in range(samples):
        n_genes = 128
        n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
        n_treatments = 4
        n_population = 10_000
        n_symptoms = 10
        #np.random.seed(2)
        population = Population(n_genes, n_vaccines, n_treatments)
        X = population.generate(n_population)
        n_features = X.shape[1]

        policy = Naive(n_actions, action_space)
        model = ThompsonSampling(n_actions)
        policy.set_model(model)
        action_arr = np.zeros(n_population)
        symptom_arr = np.zeros((n_population, n_symptoms+1))

        for t in range(n_population):
            a_t = policy.get_action()
            y_t = population.vaccinate([t], a_t.reshape((1, 1)))
            new_col = y_t[:,[5,7,8]].sum(axis=1)
            new_col0 = new_col > 0
            new_col01 = new_col0.astype(int)
            new_col01 = np.reshape(new_col01,(1,-1))
            y_t_new = np.hstack((y_t,new_col01))
            policy.observe_with_laplace(a_t, y_t_new[:,10], epsilon=1)
            action_arr[t] = a_t
            symptom_arr[t] = y_t_new

        alpha, beta = policy.model.get_params()

        print(i, alpha, beta)
        alphas[i,:] = alpha
        betas[i,:] = beta
    return alphas, betas
```

[46]:
```
def laplace_histogram():
    alphas_laplace, betas_laplace = expected_utility_laplace(100)
    utility_laplace = alphas_laplace.sum(axis=1)
    plt.hist(-utility_laplace, bins=30)
    plt.title('Histogram utility policy with laplace noise (1/epsilon)')
    mean_laplace = np.mean(utility_laplace)
    std_laplace = np.std(utility_laplace)
    xlabel_str_laplace = f"Utility: Mean {-mean_laplace:.4f}, Std {std_laplace:.
↪4f}"
    plt.xlabel(xlabel_str_laplace)
```

```
    plt.ylabel('Frequency')
    plt.savefig('figures/histogram_utility_laplace.png')
    plt.show()
```

[47]:
```
def beta_comparision_short_scale_plot():
    from scipy.stats import beta
    x = np.linspace(0.034,0.038,500)
    y0 = beta.pdf(x, a[0], b[0])
    y1 = beta.pdf(x, a[1], b[1])
    y2 = beta.pdf(x, a[2], b[2])
    y3 = beta.pdf(x, a[3], b[3])
    plt.plot(x, y0, x, y1, x, y2, x, y3)
    plt.title('Beta distribution with average parameters over 100 runs')
    plt.xlabel('x')
    plt.ylabel('Density')
    plt.legend(['No vaccine', 'Vaccine 1', 'Vaccine 2', 'Vaccine 3'])
    plt.show()
```

[48]:
```
def beta_comparison_plot_long_scale():
    x = np.linspace(0,1,500)
    y0 = beta.pdf(x, a[0], b[0])
    y1 = beta.pdf(x, a[1], b[1])
    y2 = beta.pdf(x, a[2], b[2])
    y3 = beta.pdf(x, a[3], b[3])
    plt.plot(x, y0, x, y1, x, y2, x, y3)
    plt.title('Beta distribution with average parameters over 100 runs')
    plt.xlabel('x')
    plt.ylabel('Density')
    plt.legend(['No vaccine', 'Vaccine 1', 'Vaccine 2', 'Vaccine 3'])
    plt.show()
```

[49]:
```
def vaccination_comparison_plot():
    expectation = alphas/(alphas+betas)
    x0 = expectation[:,0]
    x1 = expectation[:,1]
    x2 = expectation[:,2]
    x3 = expectation[:,3]

    out = '=====================================================\n'
    out += f'Expected utility per vaccine averaged over 100 runs:\n'
    out += f'No vaccine: {np.mean(x0):.4f}, std: {np.std(x0):.4f}\n'
    out += f'Vaccine 1:  {np.mean(x1):.4f}, std: {np.std(x1):.4f}\n'
    out += f'Vaccine 2:  {np.mean(x2):.4f}, std: {np.std(x2):.4f}\n'
    out += f'Vaccine 3:  {np.mean(x3):.4f}, std: {np.std(x3):.4f}\n'
    out += '====================================================='

    with open('outputs/expectation_per_vaccine.txt', 'w') as outfile:
```

```
        outfile.write(out)
```

[50]:
```
def utility_histogram():
    utility = alphas.sum(axis=1)
    plt.hist(-utility, bins=30)
    plt.title('Histogram utility improved policy')
    mean = np.mean(utility)
    std = np.std(utility)
    xlabel_str = f"Utility: Mean {-mean:.4f}, Std {std:.4f}"
    plt.xlabel(xlabel_str)
    plt.ylabel('Frequency')
    plt.savefig('figures/histogram_utility_improved_policy.png')
    plt.show()
```

[51]:
```
def plot_fair_balance(actions, data, symptom=0, save=False, name="figures/
↪Default.png"):
    mm_y = np.where(np.logical_and(data[:,11]==1, symptom_arr[:,10]==symptom))
    ff_y = np.where(np.logical_and(data[:,11]==0, symptom_arr[:,10]==symptom))

    m_a_y = np.take(actions, mm_y)
    m_a_y = m_a_y.flatten()

    f_a_y = np.take(actions, ff_y)
    f_a_y = f_a_y.flatten()

    title = "Vaccines per gender y=0" if not symptom else "Vaccines per gender␣
↪y=1"
    plt.hist([m_a_y, f_a_y],label=['male','female'])
    plt.xlabel('Vaccine')
    plt.ylabel('Frequency')
    plt.title(title)
    plt.legend(loc='upper right')
    if save:
        plt.savefig(name)
    plt.show()
```

[52]:
```
def fair_get_arrays(actions, data, symptom_arr):
    y0 = np.where(symptom_arr[:,10]==0)
    y1 = np.where(symptom_arr[:,10]==1)
    y0_m = np.where(np.logical_and(data[:,11]==1, symptom_arr[:,10]==0))
    y1_m = np.where(np.logical_and(data[:,11]==1, symptom_arr[:,10]==1))
    y0_f = np.where(np.logical_and(data[:,11]==0, symptom_arr[:,10]==0))
    y1_f = np.where(np.logical_and(data[:,11]==0, symptom_arr[:,10]==1))

    a_y0 = np.take(actions, y0).flatten()
    a_y1 = np.take(actions, y1).flatten()
    a_y0_m = np.take(actions, y0_m).flatten()
```

```
    a_y1_m = np.take(actions, y1_m).flatten()
    a_y0_f = np.take(actions, y0_f).flatten()
    a_y1_f = np.take(actions, y1_f).flatten()

    y0 = []
    y1 = []
    y0_m = []
    y1_m = []
    y0_f = []
    y1_f = []
    for i in range(4):
        y0.append(np.sum(a_y0==i))
        y1.append(np.sum(a_y1==i))
        y0_m.append(np.sum(a_y0_m==i))
        y1_m.append(np.sum(a_y1_m==i))
        y0_f.append(np.sum(a_y0_f==i))
        y1_f.append(np.sum(a_y1_f==i))
    return y0, y1, y0_m, y1_m, y0_f, y1_f
```

[53]:
```
def fair_get_F_metric(y0, y1, y0_m, y1_m, y0_f, y1_f):
    p_y0 = y0/sum(y0)
    p_y1 = y1/sum(y1)
    p_y0_m = y0_m/sum(y0_m)
    p_y1_m = y1_m/sum(y1_m)
    p_y0_f = y0_f/sum(y0_f)
    p_y1_f = y1_f/sum(y1_f)

    squared_diff_arr = []
    # y = 0
    for i in range(4):
        squared_diff_arr.append((p_y0[i]-p_y0_m[i])**2)
        squared_diff_arr.append((p_y0[i]-p_y0_f[i])**2)
    # y = 1
    for i in range(4):
        squared_diff_arr.append((p_y1[i]-p_y1_m[i])**2)
        squared_diff_arr.append((p_y1[i]-p_y1_f[i])**2)

    return sum(squared_diff_arr)
```

[54]:
```
def fair_get_output(to_file=False, name='outputs/default.txt'):
    y0, y1, y0_m, y1_m, y0_f, y1_f = fair_get_arrays(actions=action_arr,␣
 ↪data=X, symptom_arr=symptom_arr)
    F = fair_get_F_metric(y0, y1, y0_m, y1_m, y0_f, y1_f)
    output_str = ""
    output_str += f'============================== y=0␣
 ↪==================================\n'
```

```
    output_str += f'             |       P(a|y=0)  | P(a|y=0,z=male) |␣
↪P(a|y=0,z=female) |\n'
    output_str += f'No vaccine: | {y0[0]:4}/{sum(y0):4}={y0[0]/sum(y0):.3f} |␣
↪{y0_m[0]:4}/{sum(y0_m):4}={y0_m[0]/sum(y0_m):.3f} | {y0_f[0]:4}/{sum(y0_f):
↪4}={y0_f[0]/sum(y0_f):.3f}   |\n'
    output_str += f'Vaccine 1:  | {y0[1]:4}/{sum(y0):4}={y0[1]/sum(y0):.3f} |␣
↪{y0_m[1]:4}/{sum(y0_m):4}={y0_m[1]/sum(y0_m):.3f} | {y0_f[1]:4}/{sum(y0_f):
↪4}={y0_f[1]/sum(y0_f):.3f}   |\n'
    output_str += f'Vaccine 2:  | {y0[2]:4}/{sum(y0):4}={y0[2]/sum(y0):.3f} |␣
↪{y0_m[2]:4}/{sum(y0_m):4}={y0_m[2]/sum(y0_m):.3f} | {y0_f[2]:4}/{sum(y0_f):
↪4}={y0_f[2]/sum(y0_f):.3f}   |\n'
    output_str += f'Vaccine 3:  | {y0[3]:4}/{sum(y0):4}={y0[3]/sum(y0):.3f} |␣
↪{y0_m[3]:4}/{sum(y0_m):4}={y0_m[3]/sum(y0_m):.3f} | {y0_f[3]:4}/{sum(y0_f):
↪4}={y0_f[3]/sum(y0_f):.3f}   |\n'
    output_str += f'============================== y=1␣
↪================================\n'
    output_str += f'             |       P(a|y=0)  | P(a|y=0,z=male) |␣
↪P(a|y=0,z=female) |\n'
    output_str += f'No vaccine: | {y1[0]:4}/{sum(y1):4}={y1[0]/sum(y1):.3f} |␣
↪{y1_m[0]:4}/{sum(y1_m):4}={y1_m[0]/sum(y1_m):.3f} | {y1_f[0]:4}/{sum(y1_f):
↪4}={y1_f[0]/sum(y1_f):.3f}   |\n'
    output_str += f'Vaccine 1:  | {y1[1]:4}/{sum(y1):4}={y1[1]/sum(y1):.3f} |␣
↪{y1_m[1]:4}/{sum(y1_m):4}={y1_m[1]/sum(y1_m):.3f} | {y1_f[1]:4}/{sum(y1_f):
↪4}={y1_f[1]/sum(y1_f):.3f}   |\n'
    output_str += f'Vaccine 2:  | {y1[2]:4}/{sum(y1):4}={y1[2]/sum(y1):.3f} |␣
↪{y1_m[2]:4}/{sum(y1_m):4}={y1_m[2]/sum(y1_m):.3f} | {y1_f[2]:4}/{sum(y1_f):
↪4}={y1_f[2]/sum(y1_f):.3f}   |\n'
    output_str += f'Vaccine 3:  | {y1[3]:4}/{sum(y1):4}={y1[3]/sum(y1):.3f} |␣
↪{y1_m[3]:4}/{sum(y1_m):4}={y1_m[3]/sum(y1_m):.3f} | {y1_f[3]:4}/{sum(y1_f):
↪4}={y1_f[3]/sum(y1_f):.3f}   |\n'
    output_str += f'================== Fairness metric balance␣
↪==========================\n'
    output_str += f'Sum(|P(a_j|y_i) - P(a_j|y_i,z_k)|^2 for all i,j,k: {F:.
↪5f}\n'
    output_str +=␣
↪f'=================================================================\n'
    if to_file:
        with open(name, 'w') as outfile:
            outfile.write(output_str)
    print(output_str)
```

[55]:
```
def fair_F_simulation(samples=10):
    F_values = []
    for i in range(samples):
        n_genes = 128
        n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
```

```
        n_treatments = 4
        n_population = 10_000
        n_symptoms = 10
        #np.random.seed(2)
        population = Population(n_genes, n_vaccines, n_treatments)
        X = population.generate(n_population)
        n_features = X.shape[1]

        policy = Naive(n_actions, action_space)
        model = ThompsonSampling(n_actions)
        policy.set_model(model)
        action_arr = np.zeros(n_population)
        symptom_arr = np.zeros((n_population, n_symptoms+1))

        for t in range(n_population):
            a_t = policy.get_action()
            y_t = population.vaccinate([t], a_t.reshape((1, 1)))
            new_col = y_t[:,[5,7,8]].sum(axis=1)
            new_col0 = new_col > 0
            new_col01 = new_col0.astype(int)
            new_col01 = np.reshape(new_col01,(1,-1))
            y_t_new = np.hstack((y_t,new_col01))
            policy.observe(a_t, y_t_new[:,10])
            action_arr[t] = a_t
            symptom_arr[t] = y_t_new

        y0, y1, y0_m, y1_m, y0_f, y1_f = fair_get_arrays(actions=action_arr,␣
↪data=X, symptom_arr=symptom_arr)
        F = fair_get_F_metric(y0, y1, y0_m, y1_m, y0_f, y1_f)
        print(i, F)
        F_values.append(F)
    return F_values
```

[56]:
```
def fair_F_simulation_histogram(samples=100, save=False, name='figures/
↪fair_F_simulation_histogram.png'):
    F_values = fair_F_simulation(samples=100)
    plt.hist(F_values)
    title = 'Histogram of unfairness in balance, ' + str(samples) + 'runs'
    plt.title(title)
    plt.xlabel('F_balance metric: unfairness')
    plt.ylabel('frequency')
    if save:
        plt.savefig(name)
    plt.show()
```

[57]:
```
def get_gender_fraction(to_file=False, name='outputs/gender_average_100_runs.
↪txt'):
    n_genes = 128
    n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
    n_treatments = 4
    n_population = 10_000
    n_symptoms = 10

    avg = []
    for i in range(100):
        population = Population(n_genes, n_vaccines, n_treatments)
        X = population.generate(n_population)
        m = np.mean(X[:,11])
        print(m)
        avg.append(np.mean(X[:,11]))
    np.mean(avg)

    if to_file:
        with open('outputs/gender_average_100_runs.txt', 'w') as outfile:
            out =␣
↪f'============================================================\n'
            out += f'Percentage males, 100 runs with 10_000 individuals: {np.
↪mean(avg):.4f}\n'
            out += f'Standard deviation: {np.std(avg):.4f}\n'
            out += f'============================================================'
            outfile.write(out)
```

thompson

December 9, 2021

[2]:
```
from covid.simulator import Population
from covid.auxilliary import symptom_names
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from covid.policy import Policy
from scipy.stats import beta, bernoulli, uniform
```

[3]:
```
np.random.laplace(0,0.1)
```

[3]:
```
-0.19614104708356772
```

[4]:
```
class ThompsonSampling:
    def __init__(self, nvacc):
        # Priors for the beta-bernoulli model
        self.a = np.ones(nvacc) # uniform prior
        self.b = np.ones(nvacc) # uniform prior
        self.nvacc = nvacc

    def update(self, action, outcome):
        self.a[action] += outcome
        self.b[action] += 1 - outcome

    def update_with_laplace(self, action, outcome):
        outcome = outcome + np.random.laplace(0, 0.1)
        self.a[action] += outcome
        self.b[action] += 1 - outcome

    def get_params(self):
        # Returns the parameters of all the beta distrobutions.
        return self.a, self.b

    def get_prob(self):
        return beta.rvs(self.a, self.b)
```

[5]:
```
class Naive(Policy):
    def set_model(self, model):
        self.model = model

    def get_action(self):
        probs = self.model.get_prob()

        #print(probs)
        ret_val = np.argmin(probs)
        #print(ret_val)
        #print(ret_val)
        return ret_val

    def observe(self, action, outcome):
        self.model.update(action, outcome)

    def observe_with_laplace(self, action, outcome):
        self.model.update_with_laplace(action, outcome)
```

[6]:
```
class NaiveForExample(Policy):
    def set_model(self, model):
        self.model = model

    def get_action(self):
        probs = self.model.get_prob()

        #print(probs)
        ret_val = np.argmax(probs)
        #print(ret_val)
        #print(ret_val)
        return ret_val

    def observe(self, action, outcome):
        self.model.update(action, outcome)

    def observe_with_laplace(self, action, outcome):
        self.model.update_with_laplace(action, outcome)
```

[7]:
```
thetas = np.array([0.8, 0.75, 0.85])
```

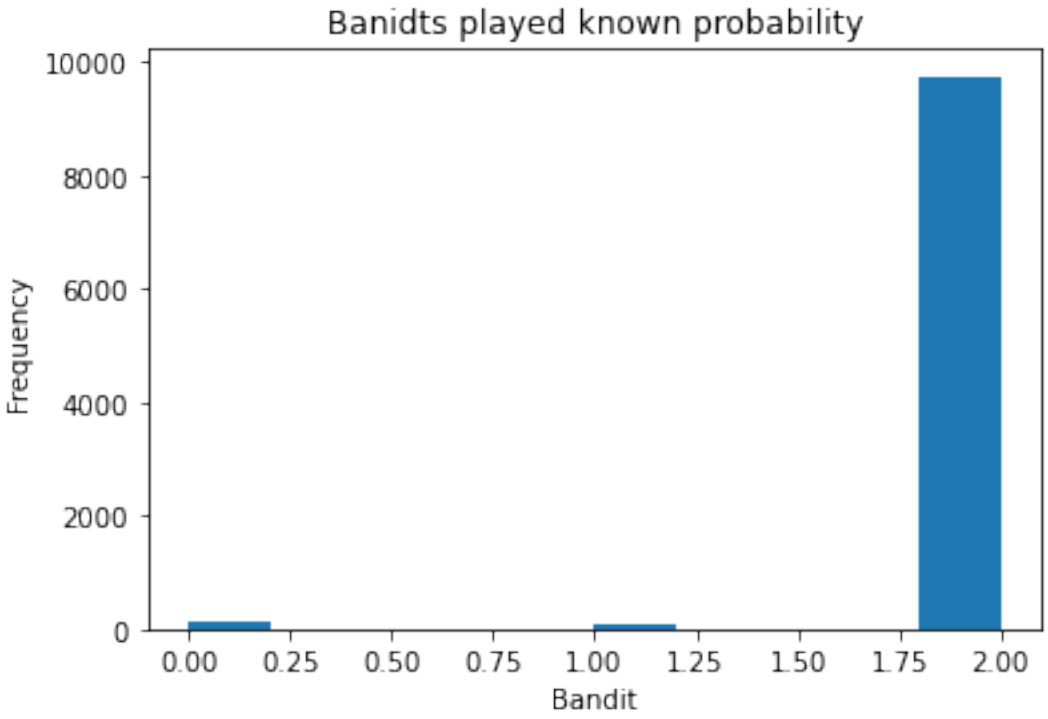
[8]:
```
n_genes = 128
n_vaccines = 3
n_treatment = 4
#population = Population(n_genes, n_vaccines, n_treatment)
N = 10000
#X = population.generate(N)
```

[9]:
```
policy = NaiveForExample(n_actions=3, action_set=[0, 1, 2])
model = ThompsonSampling(nvacc=3)
policy.set_model(model)

action_arr = np.zeros(N)
symptom_arr = np.zeros(N)

for i in range(N):
    action = policy.get_action()
    #print(thetas[action])
    reward = bernoulli.rvs(thetas[action])
    #print(reward)
    policy.observe(action, reward)
    action_arr[i] = action
    symptom_arr[i] = reward

    #print(action, reward)
    #policy.observe(action, reward)
    #print(action)

alphas, betas = policy.model.get_params()
for i, j in zip(alphas, betas):
    print(i,j)
print(alphas[0]/(alphas[0] + betas[0]))
print(alphas[1]/(alphas[1] + betas[1]))
print(alphas[2]/(alphas[2] + betas[2]))
```

```
Initialising policy with  3 actions
A = { [0, 1, 2] }
114.0 34.0
76.0 28.0
8338.0 1416.0
0.7702702702702703
0.7307692307692307
0.8548287881894607
```

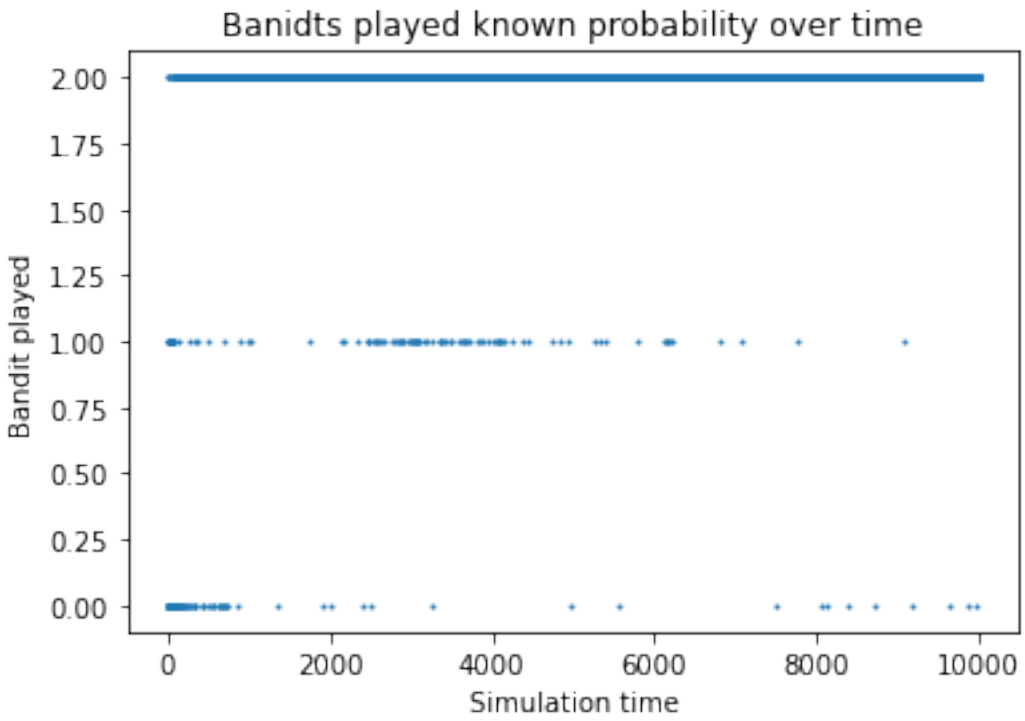
[10]:
```
#print(action_arr)
#plt.scatter(range(N), action_arr)
def plot_action_known_theta(actions):
    plt.hist(actions)
    plt.xlabel('Bandit')
    plt.ylabel('Frequency')
    plt.title('Banidts played known probability')
    #plt.legend(loc='upper right')
    plt.savefig('figures/banidts_played_known_prob_hist.png')
    plt.show()
```

[11]:
```
plot_action_known_theta(action_arr)
```



[12]:
```
def plot_bandits_played_time(actions, N):
    plt.scatter(range(N), actions, s=1)
    plt.ylabel('Bandit played')
    plt.xlabel('Simulation time')
    plt.title('Banidts played known probability over time')
    #plt.legend(loc='upper right')
    plt.savefig('figures/banidts_played_known_prob_over_time.png')
    plt.show()
```

[13]:
```
plot_bandits_played_time(action_arr, N)
```



[14]:
```
n_genes = 128
n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
n_treatments = 4
n_population = 10_000
n_symptoms = 10

# symptom names for easy reference
from covid.auxilliary import symptom_names

np.random.seed(1)

population = Population(n_genes, n_vaccines, n_treatments)
X = population.generate(n_population)
n_features = X.shape[1]
```

[15]:
```
#new_col = X[:,[5,7,8]].sum(axis=1)
#new_col0 = new_col > 0
#new_col01 = new_col0.astype(int)
#new_col01 = np.reshape(new_col01,(n_population,-1))
#X_new = np.hstack((X,new_col01))
```

[16]:
```
action_space = np.array([-1,0,1,2])
n_actions = action_space.shape[0]
```

[17]:
```
policy = Naive(n_actions, action_space)
model = ThompsonSampling(n_actions)
policy.set_model(model)
```

```
Initialising policy with  4 actions
A = { [-1  0  1  2] }
```

[18]:
```
action_arr = np.zeros(n_population)
symptom_arr = np.zeros((n_population, n_symptoms+1))
symptom_arr.shape
```

[18]:
```
(10000, 11)
```

[19]:
```
print("With a for loop")
# The simplest way to work is to go through every individual in the population
for t in range(n_population):
    a_t = policy.get_action()
    #print(type(a_t))
    #a_t = np.int(3)
    # Then you can obtain results for everybody
    y_t = population.vaccinate([t], a_t.reshape((1, 1)))
    #print(y_t.shape)
    new_col = y_t[:,[5,7,8]].sum(axis=1)

    #print(y_t)
    new_col0 = new_col > 0
    new_col01 = new_col0.astype(int)

    #new_col01 = np.reshape(new_col01,(n_population,-1))

    new_col01 = np.reshape(new_col01,(1,-1))
    y_t_new = np.hstack((y_t,new_col01))

    # Feed the results back in your policy. This allows you to fit the
    # statistical model you have.
    #print( y_t_new[:,10])

    policy.observe(a_t, y_t_new[:,10])
    #policy.observe(a_t, y_t_new[:,1])

    action_arr[t] = a_t
    symptom_arr[t] = y_t_new
```

```
With a for loop
```

[20]:
```
print(action_arr)
```

```
[3. 3. 3. … 1. 0. 1.]
```

[21]:
```
alphas, betas = policy.model.get_params()
for i, j in zip(alphas, betas):
    print(i,j)
print(alphas[0]/(alphas[0] + betas[0]))
print(alphas[1]/(alphas[1] + betas[1]))
print(alphas[2]/(alphas[2] + betas[2]))
print(alphas[3]/(alphas[3] + betas[3]))
```

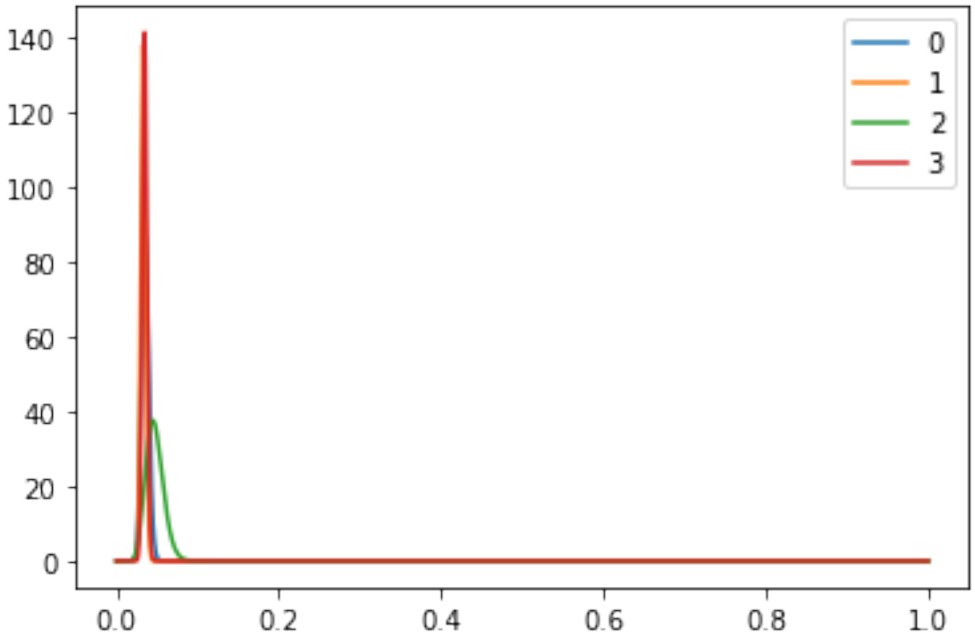
```
58.0 1579.0
126.0 3676.0
18.0 364.0
146.0 4041.0
0.0354306658521686
0.03314045239347712
0.04712041884816754
0.03486983520420349
```

[22]:
```
utility_normal = -sum(alphas)
utility_normal
```

[22]:
```
-348.0
```

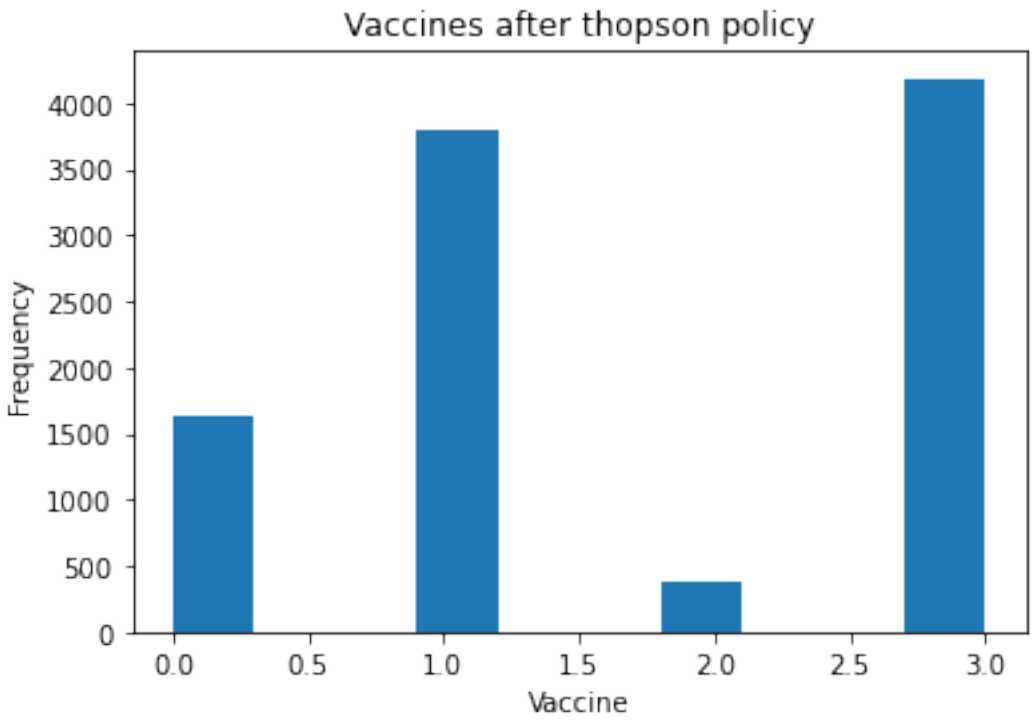
[23]:
```
xs = np.linspace(0, 1, 10000)
plt.plot(xs, beta.pdf(xs, alphas[0], betas[0]))
plt.plot(xs, beta.pdf(xs, alphas[1], betas[1]))
plt.plot(xs, beta.pdf(xs, alphas[2], betas[2]))
plt.plot(xs, beta.pdf(xs, alphas[3], betas[3]))
plt.legend(['0', '1', '2', '3'])

#
```

[23]:
```
<matplotlib.legend.Legend at 0x7f91780fde80>
```



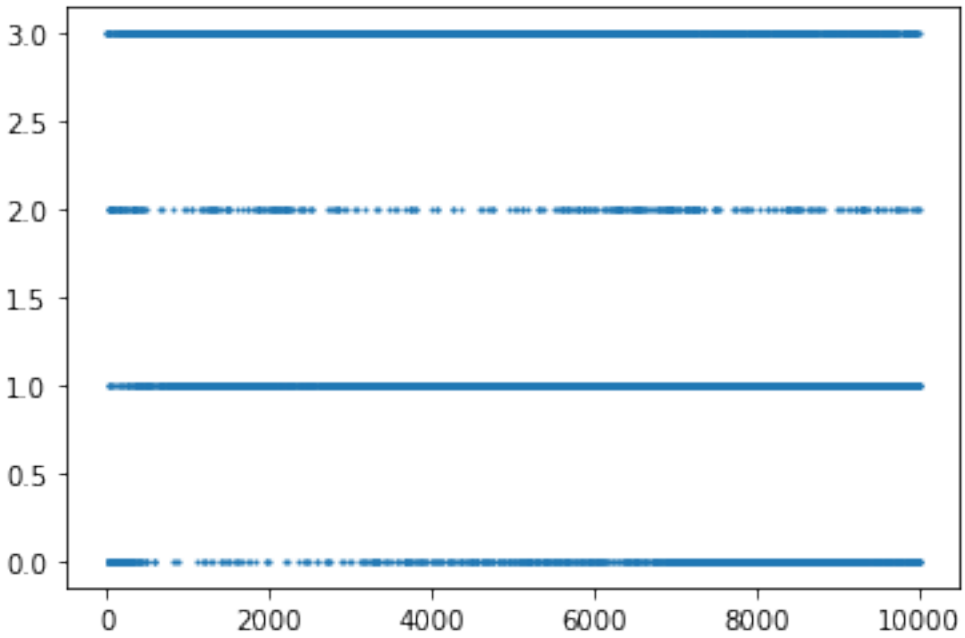
[24]:
```
def plot_thompson_pol(actions):
    #plt.hist(action_arr)
    plt.hist(actions)#, label=['0','1','2','3'])
    plt.xlabel('Vaccine')
    plt.ylabel('Frequency')
    plt.title('Vaccines after thopson policy')
    #plt.legend(loc='upper right')
    plt.savefig('figures/vaccines_thompson_policy.png')
    plt.show()
```

[25]:
```
plot_thompson_pol(action_arr)
```



[26]:
```
plt.scatter(range(n_population), action_arr, s=1)
```

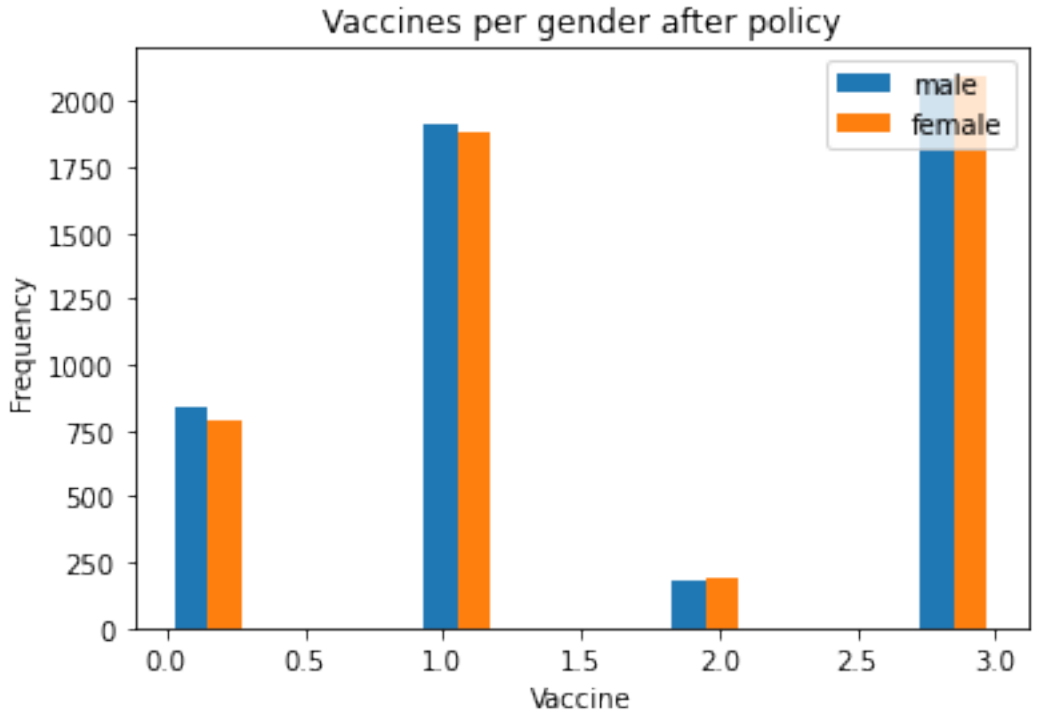
[26]:
```
<matplotlib.collections.PathCollection at 0x7f917852a9d0>
```



[27]:
```
print(action_arr.shape)

def plot_fair_pol(actions, data):
    mm = np.where(data[:,11]==1) #male
    ff = np.where(data[:,11]==0) #female

    m_a = np.take(actions, mm)
    m_a = m_a.flatten()

    f_a = np.take(actions, ff)
    f_a = f_a.flatten()

    plt.hist([m_a, f_a],label=['male','female'])
    plt.xlabel('Vaccine')
    plt.ylabel('Frequency')
    plt.title('Vaccines per gender after policy')
    plt.legend(loc='upper right')
    plt.savefig('figures/fair_policy_plot.png')
    plt.show()
```

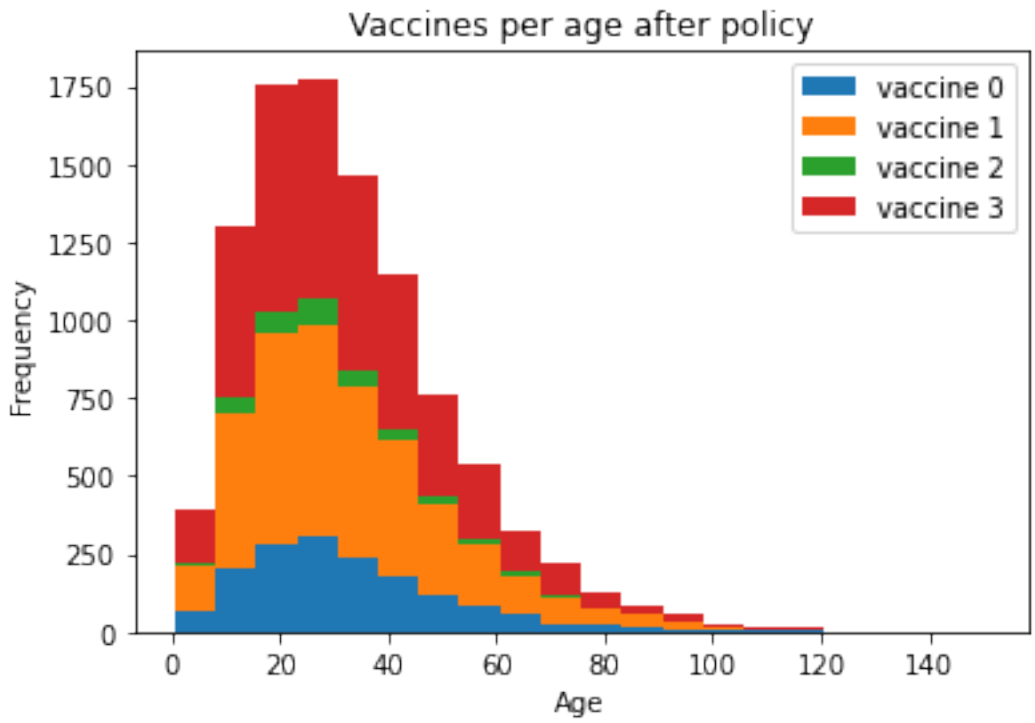
```
(10000,)
```

[28]:
```
plot_fair_pol(action_arr, X)
```



[29]:
```
def plot_vacc_age(data, actions):
    vacc_0 = np.where(actions==0) #no vaccine
    vacc_1 = np.where(actions==1) #vaccine 1
    vacc_2 = np.where(actions==2) #vaccine 2
    vacc_3 = np.where(actions==3) #vaccine 3

    v0_vacc = np.take(data[:,10], vacc_0)
    v0_vacc = v0_vacc.flatten()

    v1_vacc = np.take(data[:,10], vacc_1)
    v1_vacc = v1_vacc.flatten()

    v2_vacc = np.take(data[:,10], vacc_2)
    v2_vacc = v2_vacc.flatten()

    v3_vacc = np.take(data[:,10], vacc_3)
    v3_vacc = v3_vacc.flatten()


    #f_a = np.take(actions, ff)
    #f_a = f_a.flatten()

    plt.hist([v0_vacc, v1_vacc, v2_vacc, v3_vacc],label=['vaccine 0','vaccine␣
↪1','vaccine 2','vaccine 3'], bins=20, stacked=True)
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.title('Vaccines per age after policy')
    plt.legend(loc='upper right')
    plt.savefig('figures/fair_age_plot.png')
    plt.show()
```

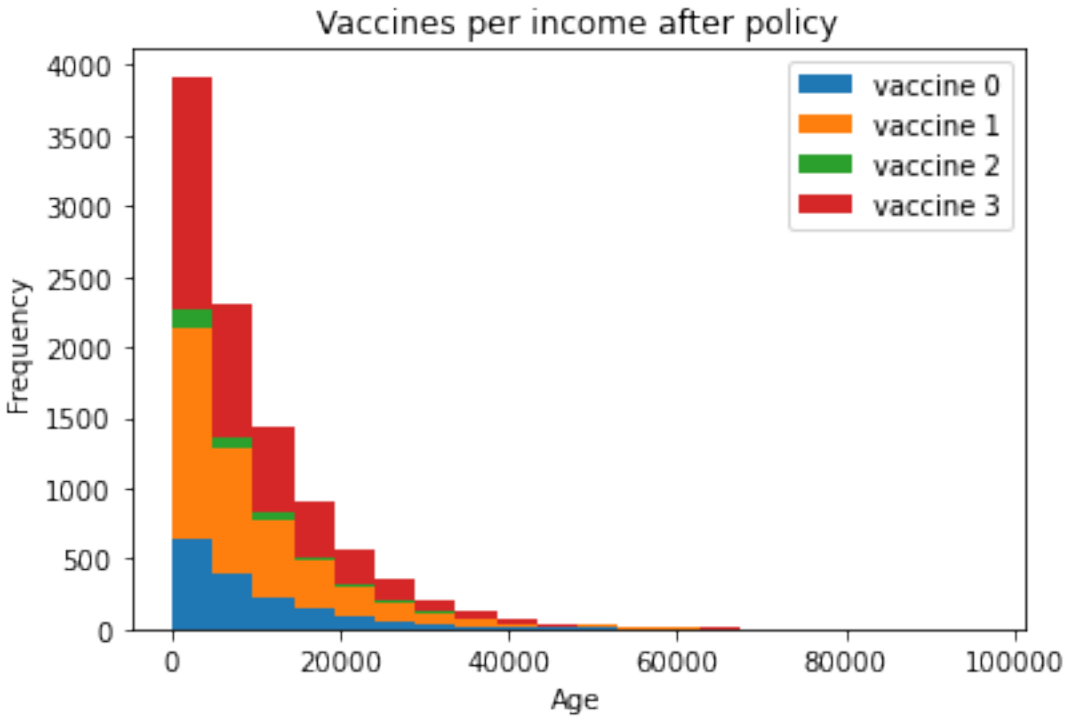
[30]:
```
plot_vacc_age(X, action_arr)
```



[31]:
```
def plot_vacc_inc(data, actions):
    vacc_0 = np.where(actions==0) #no vaccine
    vacc_1 = np.where(actions==1) #vaccine 1
    vacc_2 = np.where(actions==2) #vaccine 2
    vacc_3 = np.where(actions==3) #vaccine 3

    v0_vacc = np.take(data[:,12], vacc_0)
    v0_vacc = v0_vacc.flatten()

    v1_vacc = np.take(data[:,12], vacc_1)
    v1_vacc = v1_vacc.flatten()

    v2_vacc = np.take(data[:,12], vacc_2)
    v2_vacc = v2_vacc.flatten()

    v3_vacc = np.take(data[:,12], vacc_3)
    v3_vacc = v3_vacc.flatten()


    #f_a = np.take(actions, ff)
    #f_a = f_a.flatten()
```

```
    plt.hist([v0_vacc, v1_vacc, v2_vacc, v3_vacc],label=['vaccine 0','vaccine␣
↪1','vaccine 2','vaccine 3'], bins=20, stacked=True)
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.title('Vaccines per income after policy')
    plt.legend(loc='upper right')
    plt.savefig('figures/fairnes_salary_plot.png')
    plt.show()
```

[32]:
```
plot_vacc_inc(X, action_arr)
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

1 Thompson with laplace

[33]:
```
n_genes = 128
n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
n_treatments = 4
n_population = 10_000
n_symptoms = 10

# symptom names for easy reference
from covid.auxilliary import symptom_names

np.random.seed(1)

population = Population(n_genes, n_vaccines, n_treatments)
X = population.generate(n_population)
n_features = X.shape[1]
```

[34]:
```
action_space = np.array([-1,0,1,2])
n_actions = action_space.shape[0]

action_arr_lap = np.zeros(n_population)
symptom_arr_lap = np.zeros((n_population, n_symptoms+1))
```

[35]:
```
policy_l = Naive(n_actions, action_space)
model_l = ThompsonSampling(n_actions)
policy_l.set_model(model)
```

```
Initialising policy with  4 actions
A = { [-1  0  1  2] }
```

[ ]:
```
```

[36]:
```
print("With a for loop")
# The simplest way to work is to go through every individual in the population
for t in range(n_population):
    a_t = policy_l.get_action()

    # Then you can obtain results for everybody
    y_t = population.vaccinate([t], a_t.reshape((1, 1)))

    new_col = y_t[:,[5,7,8]].sum(axis=1)

    new_col0 = new_col > 0
    new_col01 = new_col0.astype(int)

    new_col01 = np.reshape(new_col01,(1,-1))
    y_t_new = np.hstack((y_t,new_col01))
```

```
    # Feed the results back in your policy. This allows you to fit the
    # statistical model you have.

    policy_l.observe_with_laplace(a_t, y_t_new[:,10])
    #policy.observe(a_t, y_t_new[:,1])

    action_arr_lap[t] = a_t
    symptom_arr_lap[t] = y_t_new
```

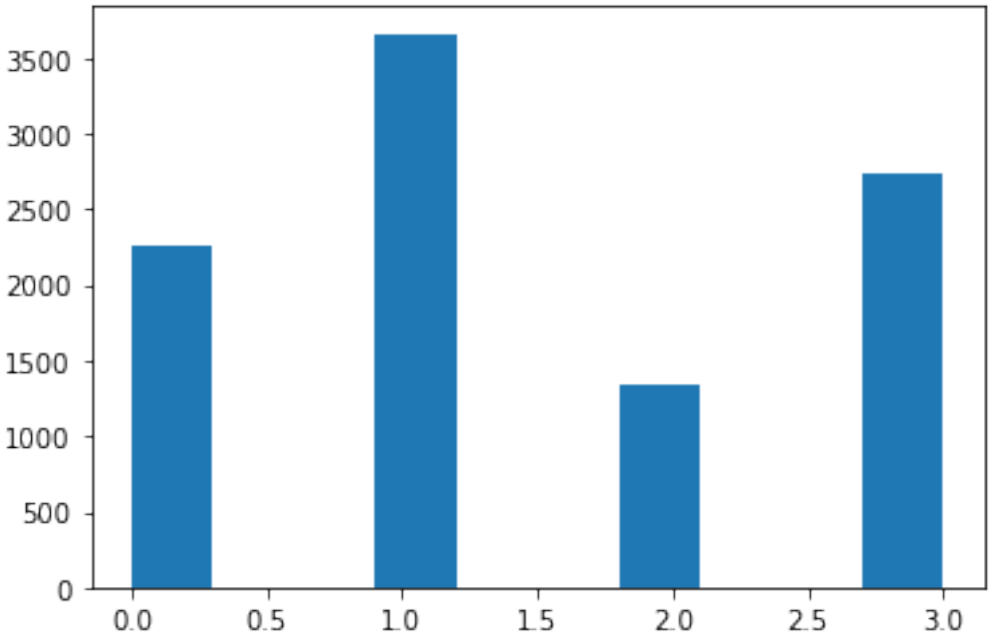
With a for loop

[37]:
```
alphas_l, betas_l = policy_l.model.get_params()
for i, j in zip(alphas_l, betas_l):
    print(i,j)
print(alphas_l[0]/(alphas_l[0] + betas_l[0]))
print(alphas_l[1]/(alphas_l[1] + betas_l[1]))
print(alphas_l[2]/(alphas_l[2] + betas_l[2]))
```

```
142.49802833539425 3757.5019716646075
261.14442632401193 7199.8555736759945
64.81713641433676 1660.1828635856637
243.64451720493534 6678.355482795076
0.03653795598343441
0.035001263412948895
0.03757515154454304
```

[38]:
```
plt.hist(action_arr_lap)
```

[38]:
```
(array([2263.,    0.,    0., 3659.,    0.,    0., 1343.,    0.,    0.,
        2735.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <BarContainer object of 10 artists>)
```



[ ]:

2 Utility

[39]:
```
utility_laplace = -sum(alphas_l)
```

[40]:
```
utility_laplace
```

[40]: -712.1041082786783

[41]:
```
#utility_normal = -sum(alphas)
```

[42]:
```
utility_normal
```

[42]: -348.0

[43]:
```
alphas
```

[43]: array([142.49802834, 261.14442632, 64.81713641, 243.6445172])

3 Experiment design

3.1 historical

[44]:
```
n_genes = 128
n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
n_treatments = 4
n_population = 10_000
n_symptoms = 10

# symptom names for easy reference
from covid.auxilliary import symptom_names

np.random.seed(1)

population = Population(n_genes, n_vaccines, n_treatments)
X = population.generate(n_population)
n_features = X.shape[1]
```

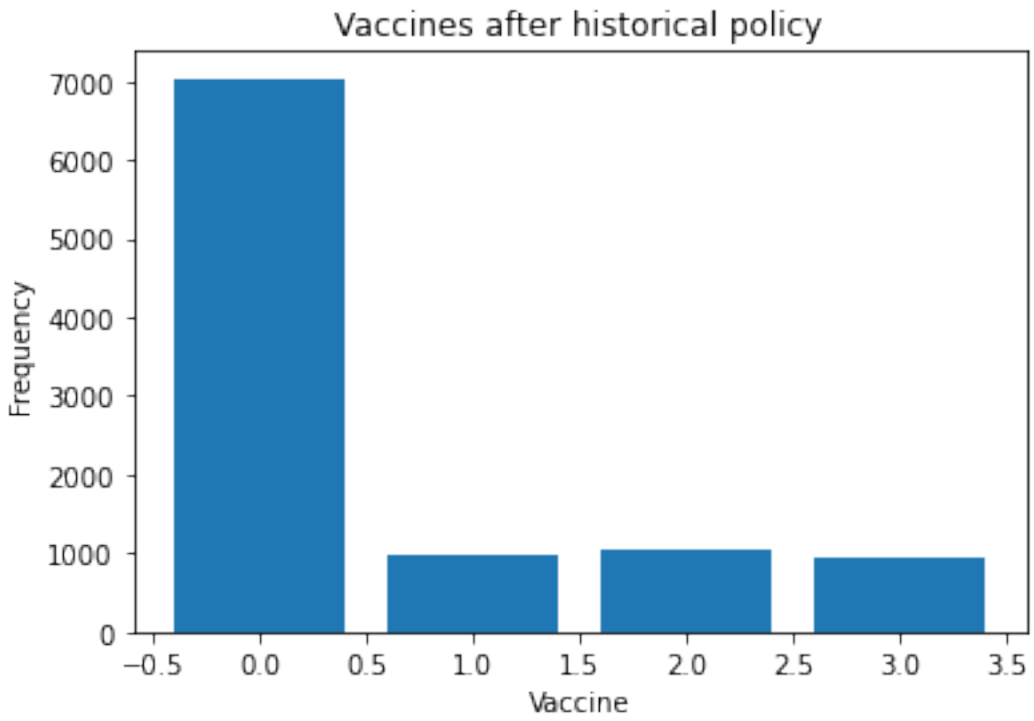
[45]:
```
new_col = X[:,[5,7,8]].sum(axis=1)
new_col0 = new_col > 0
new_col01 = new_col0.astype(int)
new_col01 = np.reshape(new_col01,(n_population,-1))
X_new = np.hstack((X,new_col01))

hist_pol_utility = -X_new[:,150].sum(axis=0)
```

[46]:
```
hist_pol_utility
```

[46]:
```
-243.0
```

[47]:
```
def plot_hist_policy(data):

    v1 = data[:,147].sum(axis=0)
    v2 = data[:,148].sum(axis=0)
    v3 = data[:,149].sum(axis=0)
    v0 = (data[:,[147,148,149]].sum(axis=1)==0).sum()
    plt.bar([0,1,2,3], [v0,v1,v2,v3])
    plt.xlabel('Vaccine')
    plt.ylabel('Frequency')
    plt.title('Vaccines after historical policy')
    plt.savefig('figures/vaccices_historical_policy.png')
    plt.show()
```

[48]:
```
plot_hist_policy(X)
```



[49]:
```
from covid.policy import RandomPolicy
```

[50]:
```
"""
vaccine_policy = RandomPolicy(n_vaccines, action_space) # make sure to add -1␣
 ↪for 'no vaccine'

Y_rand_pol = np.zeros((n_population,n_symptoms))
A_rand_pol = np.zeros(n_population)


print("With a for loop")
# The simplest way to work is to go through every individual in the population
for t in range(n_population):
    #a_t = vaccine_policy.get_action(X[t])
    a_t = vaccine_policy.get_action(X[t])
    print(a_t)
    # Then you can obtain results for everybody
    y_t = population.vaccinate([t], a_t)
    #y_t = population.vaccinate([t], a_t.reshape((1, 1)))

    # Feed the results back in your policy. This allows you to fit the
    # statistical model you have.
    #vaccine_policy.observe(X[t], a_t, y_t)
```

```
    vaccine_policy.observe(X[t], a_t, y_t)

    #print(a_t.shape)
    Y_rand_pol[t] = y_t
    #A_rand_pol[t] = a_t
"""
```

[50]:
```
'\nvaccine_policy = RandomPolicy(n_vaccines, action_space) # make sure to add -1
for \'no vaccine\'\n\nY_rand_pol =
np.zeros((n_population,n_symptoms))\nA_rand_pol =
np.zeros(n_population)\n\n\nprint("With a for loop")\n# The simplest way to work
is to go through every individual in the population\nfor t in
range(n_population):\n    #a_t = vaccine_policy.get_action(X[t])\n    a_t =
vaccine_policy.get_action(X[t])\n    print(a_t)\n    # Then you can obtain
results for everybody\n    y_t = population.vaccinate([t], a_t)\n    #y_t =
population.vaccinate([t], a_t.reshape((1, 1)))\n    \n    # Feed the results
back in your policy. This allows you to fit the\n    # statistical model you
have.\n    #vaccine_policy.observe(X[t], a_t, y_t)\n
vaccine_policy.observe(X[t], a_t, y_t)\n    \n    #print(a_t.shape)\n
Y_rand_pol[t] = y_t\n    #A_rand_pol[t] = a_t\n'
```

3.2 random

[51]:
```
vaccine_policy = RandomPolicy(n_vaccines, action_space)



print("Vaccinate'em all")
# Here you can get an action for everybody in the population
A = vaccine_policy.get_action(X)
A = np.array([int(i) for i in A])
A = A.astype(int)
A = A.reshape((n_population, 1))+1
print(A)
# Then you can obtain results for everybody
Y = population.vaccinate(list(range(n_population)), A)
# Feed the results back in your policy.
vaccine_policy.observe(X, A, Y)
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
[[0]
 [2]
 [1]
 …
```

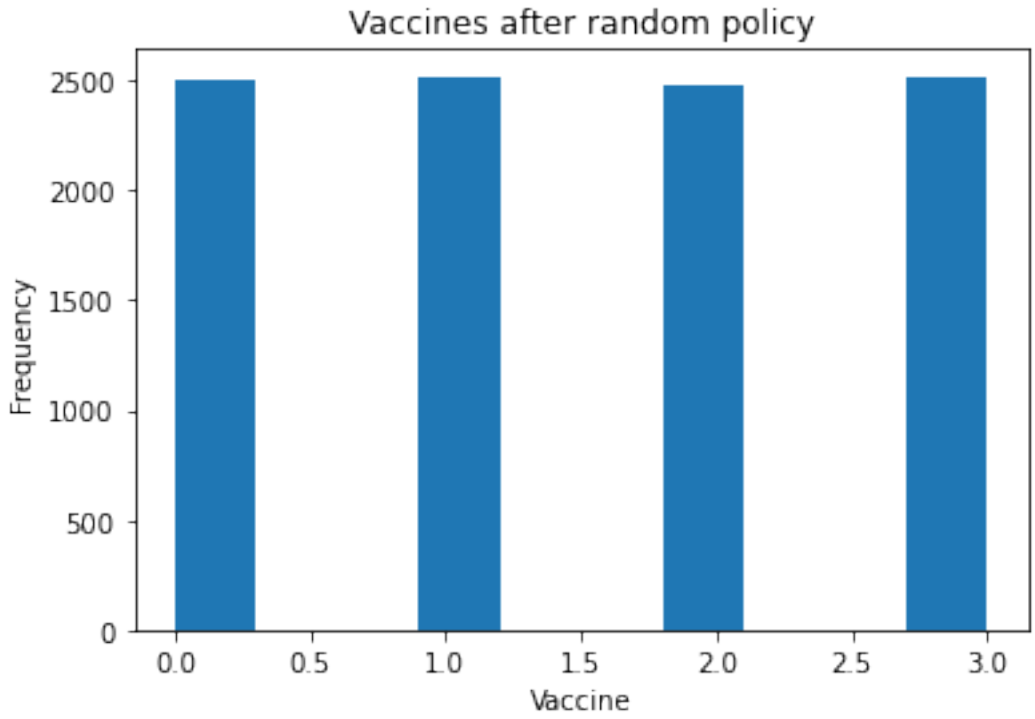
```
 [1]
 [2]
 [3]]
```

[52]:
```
print(A)
```

```
[[0]
 [2]
 [1]
 …
 [1]
 [2]
 [3]]
```

[53]:
```
def plot_random_pol(actions):

    plt.hist(actions)# label=['0','1','2','3'])
    plt.xlabel('Vaccine')
    plt.ylabel('Frequency')
    plt.title('Vaccines after random policy')
    #plt.legend(loc='upper right')
    plt.savefig('figures/vaccines_random_policy.png')
    plt.show()
```

[54]:
```
plot_random_pol(A)
```



[55]:
```
print(Y)
```

```
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

[56]:
```
new_col = Y[:,[5,7,8]].sum(axis=1)
new_col0 = new_col > 0
new_col01 = new_col0.astype(int)
new_col01 = np.reshape(new_col01,(n_population,-1))
Y_new = np.hstack((Y,new_col01))
rand_pol_utility = -Y_new[:,10].sum(axis=0)
```

[57]:
```
rand_pol_utility
```

[57]:
```
-344.0
```

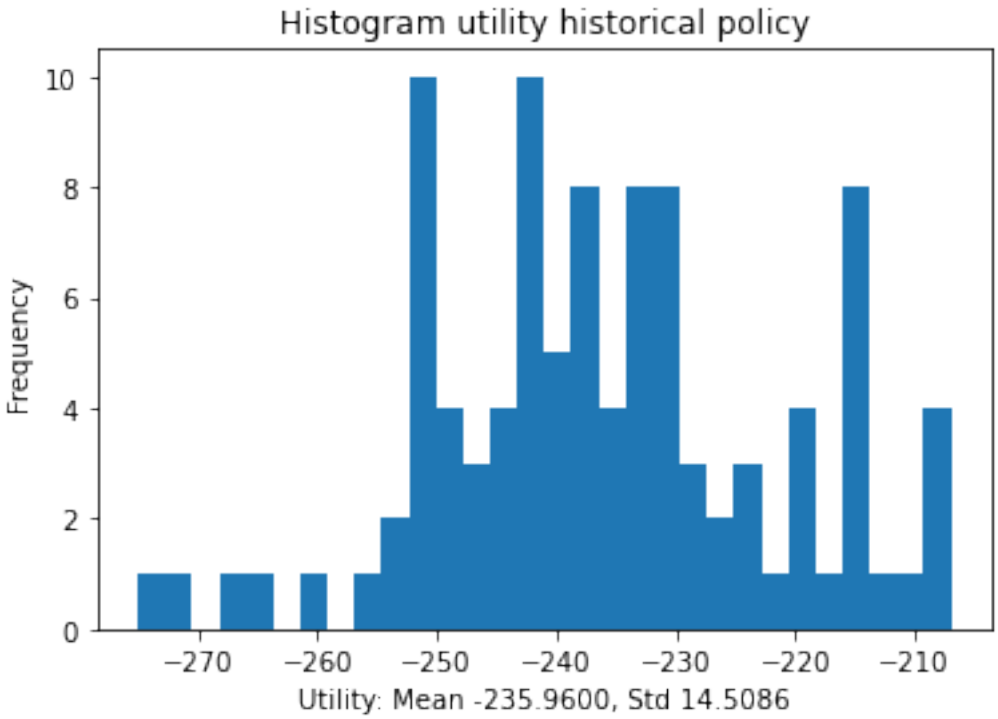
4 Bootstrap historical and simulated

4.1

[58]:
```
utility_simalated = np.zeros(100)

for i in range(100):
    n_genes = 128
    n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
    n_treatments = 4
    n_population = 10_000
    n_symptoms = 10

    # symptom names for easy reference
    from covid.auxilliary import symptom_names

    np.random.seed(i)

    population = Population(n_genes, n_vaccines, n_treatments)
    X = population.generate(n_population)
    n_features = X.shape[1]

    new_col = X[:,[5,7,8]].sum(axis=1)
```

```
    new_col0 = new_col > 0
    new_col01 = new_col0.astype(int)
    new_col01 = np.reshape(new_col01,(n_population,-1))
    X_new = np.hstack((X,new_col01))

    hist_pol_utility = -X_new[:,150].sum(axis=0)

    utility_simalated[i] = hist_pol_utility
```

[71]:
```
plt.hist(utility_simalated, bins=30)
mean = np.mean(utility_simalated)
std = np.std(utility_simalated)
plt.ylabel('Frequency')
xlabel_str = f"Utility: Mean {mean:.4f}, Std {std:.4f}"
plt.xlabel(xlabel_str)
plt.title('Histogram utility historical policy')
plt.savefig('figures/histogram_utility_historical_policy.png')
```



[ ]:
```
```

[63]:
```
utility_random = np.zeros(100)
```

```
for i in range(100):
    #n_genes = 128
    #n_vaccines = 3 # DO NOT CHANGE, breaks the simulator.
    #n_treatments = 4
    #n_population = 10_000
    #n_symptoms = 10

    # symptom names for easy reference
    #from covid.auxilliary import symptom_names

    np.random.seed(i)

    population = Population(n_genes, n_vaccines, n_treatments)
    X = population.generate(n_population)
    n_features = X.shape[1]

    vaccine_policy = RandomPolicy(n_vaccines, action_space)



    print("Vaccinate'em all")
    # Here you can get an action for everybody in the population
    A = vaccine_policy.get_action(X)
    A = np.array([int(i) for i in A])
    A = A.astype(int)
    A = A.reshape((n_population, 1))+1
    #print(A)
    # Then you can obtain results for everybody
    Y = population.vaccinate(list(range(n_population)), A)
    # Feed the results back in your policy.
    vaccine_policy.observe(X, A, Y)

    new_col = Y[:,[5,7,8]].sum(axis=1)
    new_col0 = new_col > 0
    new_col01 = new_col0.astype(int)
    new_col01 = np.reshape(new_col01,(n_population,-1))
    Y_new = np.hstack((Y,new_col01))
    rand_pol_utility = -Y_new[:,10].sum(axis=0)

    utility_random[i] = rand_pol_utility
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

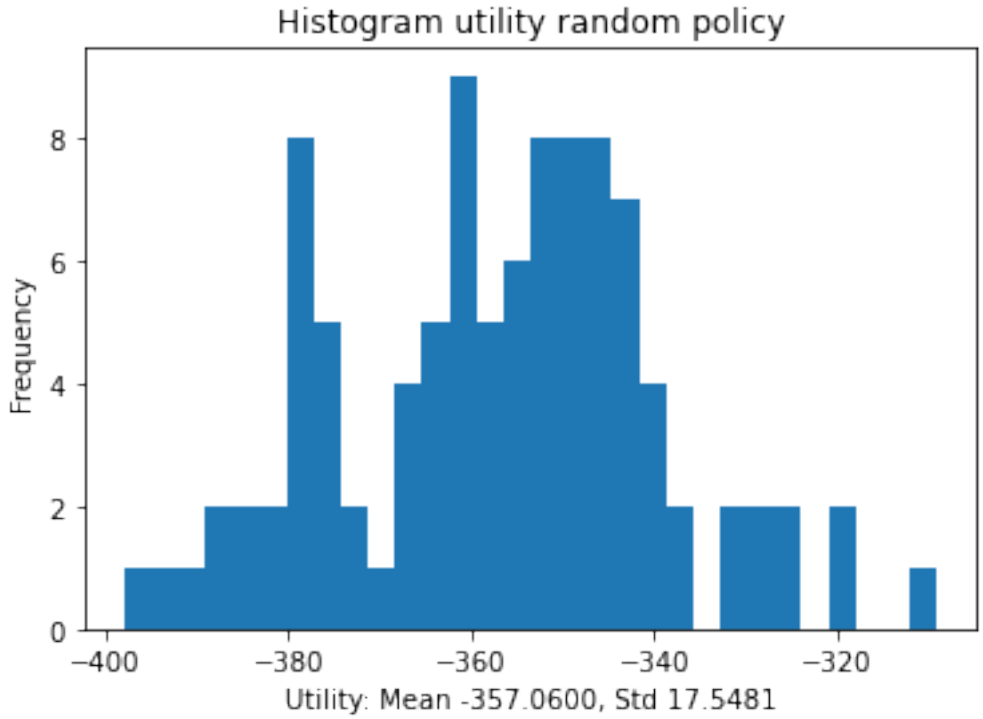
```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

```
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
Initialising policy with  3 actions
A = { [-1  0  1  2] }
Vaccinate'em all
```

[64]:
```
print("=======================================")
print("=======================================")
print("=======================================")
```

```
=======================================
=======================================
=======================================
```

[70]:
```
plt.hist(utility_random,bins=30)
mean = np.mean(utility_random)
std = np.std(utility_random)
plt.ylabel('Frequency')
xlabel_str = f"Utility: Mean {mean:.4f}, Std {std:.4f}"
plt.xlabel(xlabel_str)
plt.title('Histogram utility random policy')
plt.savefig('figures/histogram_utility_random_policy.png')
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: