```
import pandas as pd
import numpy as np
from scipy.special import comb


df = pd.read_csv('/content/2020_ten_bent_coins.csv').transpose()


df
```

	0	1	2	3	4	5	6	7	8	9	...	490	491	492	493	494	495	496	497	498	499
1	1	0	1	0	1	1	0	1	1	0	...	0	0	1	0	1	0	0	1	0	1
2	1	0	0	0	0	1	0	0	1	0	...	1	1	1	0	0	0	0	1	1	0
3	1	0	1	0	0	1	0	0	1	0	...	0	1	0	0	0	1	0	0	0	1
4	1	0	1	0	1	1	0	1	0	0	...	1	0	1	0	1	0	0	1	0	0
5	1	0	0	0	0	1	1	0	0	0	...	0	1	0	0	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	1	0	0	0	0	1	1	0	0	1	...	1	1	1	0	0	0	0	0	0	0
97	1	1	1	0	1	1	0	0	0	0	...	1	1	1	0	1	1	0	1	0	0
98	1	0	1	0	1	0	1	0	1	0	...	1	1	1	0	1	0	0	0	0	1
99	1	0	0	1	1	1	0	0	0	0	...	1	1	1	0	0	1	0	1	1	0
100	1	0	0	0	1	1	1	0	0	0	...	1	1	0	1	1	0	0	1	1	0

100 rows × 500 columns

```
# O being tail and 1 being head
# counting number of heads and tails
np.random.seed(0)
heads = df.sum().to_numpy() #numpy array
tails = 100 - heads
selected_coin = np.random.randint(0,10,size=(500,)) #creating an array of 500 values with each one having value ranging from 1 to 10
_, count_selected_coin = np.unique(selected_coin,return_counts = True) # count of which coin has been selected how many times
MLE_vector = np.zeros(10) #maximum likelihood estimation
```

selected_coin

```
array([5, 0, 3, 3, 7, 9, 3, 5, 2, 4, 7, 6, 8, 8, 1, 6, 7, 7, 8, 1, 5, 9,
       8, 9, 4, 3, 0, 3, 5, 0, 2, 3, 8, 1, 3, 3, 3, 7, 0, 1, 9, 9, 0, 4,
       7, 3, 2, 7, 2, 0, 0, 4, 5, 5, 6, 8, 4, 1, 4, 9, 8, 1, 1, 7, 9, 9,
       3, 6, 7, 2, 0, 3, 5, 9, 4, 4, 6, 4, 4, 3, 4, 4, 8, 4, 3, 7, 5, 5,
       0, 1, 5, 9, 3, 0, 5, 0, 1, 2, 4, 2, 0, 3, 2, 0, 7, 5, 9, 0, 2, 7,
       2, 9, 2, 3, 3, 2, 3, 4, 1, 2, 9, 1, 4, 6, 8, 2, 3, 0, 0, 6, 0, 6,
       3, 3, 8, 8, 8, 2, 3, 2, 0, 8, 8, 3, 8, 2, 8, 4, 3, 0, 4, 3, 6, 9,
       8, 0, 8, 5, 9, 0, 9, 6, 5, 3, 1, 8, 0, 4, 9, 6, 5, 7, 8, 8, 9, 2,
       8, 6, 6, 9, 1, 6, 8, 8, 3, 2, 3, 6, 3, 6, 5, 7, 0, 8, 4, 6, 5, 8,
       2, 3, 9, 7, 5, 3, 4, 5, 3, 3, 7, 9, 9, 9, 7, 3, 2, 3, 9, 7, 7, 5,
       1, 2, 2, 8, 1, 5, 8, 4, 0, 2, 5, 5, 0, 8, 1, 1, 0, 3, 8, 8, 4, 4,
       0, 9, 3, 7, 3, 2, 1, 1, 2, 1, 4, 2, 5, 5, 5, 2, 5, 7, 7, 6, 1, 6,
       7, 2, 3, 1, 9, 5, 9, 9, 2, 0, 9, 1, 9, 0, 6, 0, 4, 8, 4, 3, 3, 8,
       8, 7, 0, 3, 8, 7, 7, 1, 8, 4, 7, 0, 4, 9, 0, 6, 4, 2, 4, 6, 3, 3,
       7, 8, 5, 0, 8, 5, 4, 7, 4, 1, 3, 3, 9, 2, 5, 2, 3, 5, 7, 2, 7, 1,
       6, 5, 0, 0, 3, 1, 9, 9, 6, 6, 7, 8, 8, 7, 0, 8, 6, 8, 9, 8, 3, 6,
       1, 7, 4, 9, 2, 0, 8, 2, 7, 8, 4, 4, 1, 7, 6, 9, 4, 1, 5, 9, 7, 1,
       3, 5, 7, 3, 6, 6, 7, 9, 1, 9, 6, 0, 3, 8, 4, 1, 4, 5, 0, 3, 1, 4,
       4, 4, 0, 0, 8, 4, 6, 9, 3, 3, 2, 1, 2, 1, 3, 4, 1, 1, 0, 7, 8, 4,
       3, 5, 6, 3, 2, 9, 8, 1, 4, 0, 8, 3, 9, 5, 5, 1, 7, 8, 6, 4, 7, 3,
       5, 3, 6, 4, 7, 3, 0, 5, 9, 3, 7, 5, 5, 8, 0, 8, 3, 6, 9, 3, 2, 7,
       0, 3, 0, 3, 6, 1, 9, 2, 9, 4, 9, 1, 3, 2, 4, 9, 7, 4, 9, 4, 1, 2,
       7, 2, 3, 9, 7, 6, 6, 2, 3, 6, 0, 8, 0, 7, 6, 5])
```

heads

```
array([96, 26, 40, 21, 42, 89, 44, 31, 49, 10, 89, 42, 60, 65, 24, 83, 62,
       75, 79, 38,  1, 41, 53, 69, 14, 65,  0, 45, 88, 81, 48,  9, 77, 59,
        2, 68,  1, 12,  9, 89, 28, 82, 32, 77, 36,  0,  5, 35,  0, 72, 77,
        7,  0, 62, 80, 16, 90, 38, 78, 31, 38, 94, 15, 13, 11,  3, 92, 51,
       57, 10, 66,  7, 71, 37, 43, 45, 17, 29,  0, 77,  6, 57, 92, 88, 79,
        7, 25, 62, 86, 62, 54, 53, 52, 19, 10, 66, 23,  1, 40, 49, 17, 13,
       17, 18, 50, 87, 32, 49, 49, 66,  0,  0, 72, 88,  9,  4, 40, 61, 11,
       71, 77, 11, 49, 75, 92, 75, 16, 50, 21,  1, 65, 36,  9, 56, 72, 86,
        2, 77, 10, 64, 33, 95, 21, 70, 66,  2, 48, 59, 12,  2, 93,  2, 48,
       53, 38, 34, 25,  2, 10, 63,  5, 40, 30, 16, 90,  7, 71, 89,  9, 47,
       71, 65, 78, 72,  0, 27,  1, 52, 50, 17, 30, 56, 25,  0, 84, 14, 41,
       93, 41, 36, 34, 72, 68, 75, 17,  0, 72, 44, 59, 89, 78, 88, 70,  7,
       67, 30, 61, 35, 60,  0, 11,  1, 45,  8, 82, 48, 13, 56, 86, 23, 24,
       51, 38, 42, 68,  0, 22, 79, 69, 67, 79, 67, 18,  2, 40, 46, 72, 92,
       85, 46, 56,  2, 27, 84, 30, 59, 38,  3, 29, 78,  0, 84, 70, 36, 34,
       44, 51, 68, 39, 77, 58, 90, 87, 10, 75, 76, 25, 47, 89, 13, 78, 71,
       40, 77, 18, 27, 89, 53, 52, 75, 36, 48, 53, 41,  1, 49,  1, 49,  0,
       60,  9,  0, 64, 90, 74, 71, 63, 81, 47, 33, 48,  8, 12, 52, 50, 44,
       15, 19,  1, 88, 19, 82, 51, 94, 61, 58, 86, 48, 12, 20, 45, 45, 62,
```

```
 1, 57, 74, 43, 91, 15, 25, 43, 88, 91, 90, 60, 24, 41, 82,  1, 47,
56, 66, 85, 80, 27, 77, 70, 68, 10, 92,  1, 45, 52, 30,  9, 10, 89,
94, 10, 84,  7, 58, 22, 45, 18, 55, 54, 57, 50, 67, 91,  2, 54,  1,
38, 26, 25, 58, 50, 80, 15, 70, 69, 28, 36, 37,  0, 90, 29, 57, 20,
92, 88,  0, 36, 10, 88, 78, 44, 74, 83,  1, 86, 57,  0, 66, 38, 40,
34, 40, 61,  3, 20, 23,  6, 16, 71, 78, 43, 37, 34, 78, 16, 33, 36,
96, 52, 89, 67, 80, 69, 36, 72, 84,  3, 80, 30, 81, 29,  7, 23,  1,
25, 11,  9, 38, 45, 77, 33, 49,  9, 56, 16, 21, 36, 79, 44, 31, 14,
89,  1, 17, 90, 25, 17, 43, 48, 16, 29, 54, 35, 69, 19, 72, 17, 92,
 7,  3,  1,  9, 17, 78, 32, 88,  0, 91, 78, 89, 95, 27, 65, 89, 64,
 7, 57, 25,  1, 52, 27, 46])
```

```
MLE_vector.shape
```

```
(10,)
```

```
for i,j in zip(heads, selected_coin):
  MLE_vector[j] += i


#The MLE vector is then divided by the product of the count of the selected coin and the total number
#of tosses (100) to obtain the MLE estimates of the unknown bias values.
MLE_vector = MLE_vector/(count_selected_coin*100)


#A function compute_likelihood is defined to calculate the
#likelihood of a given observation (number of heads) given the number of tosses and the estimated bias value.
def compute_likelihood(obs, n, pheads):

    likelihood = comb(n, obs, exact=True)*(pheads**obs)*(1.0-pheads)**(n-obs)

    return likelihood
```

```
#The MLE estimates are updated using the Expectation-Maximization (EM) algorithm. In each iteration (or epoch) of the EM algorithm,
#the expected values of the number of heads and tails for each coin are calculated based on the current MLE estimates.
#The MLE estimates are then updated based on these expected values.
np.random.seed(0)
p_heads = np.zeros((100,10))
p_heads[0]=np.random.random((1,10))
print(p_heads[0])
#The loop continues until the improvement in the MLE estimates between two consecutive iterations is less than a threshold eps, which is set
eps = 0.01
improvement = float('inf') #positive infinity
epoch = 0
while improvement>eps:
  expectation = np.zeros((10,500,2))

  for i in range(500):
    e_head = heads[i]
    e_tail = tails[i]

    likelihood = np.zeros(10)

    for j in range(10):
      likelihood[j]= compute_likelihood(e_head,100,p_heads[epoch][j])

    weights = likelihood/np.sum(likelihood)
    for j in range(10):
      expectation[j][i] = weights[j]*np.array([e_head,e_tail])

  theta = np.zeros(10)
  for i in range(10):
    theta[i] = np.sum(expectation[i],axis =0)[0]/np.sum(expectation[i])

  p_heads[epoch+1] = theta
  print(f'Epoch ->{epoch}\n Theta ->{theta}')

  improvement = max(abs(p_heads[epoch+1]-p_heads[epoch]))
  epoch+=1
```

```
   [0.5488135  0.71518937 0.60276338 0.54488318 0.4236548  0.64589411
    0.43758721 0.891773   0.96366276 0.38344152]
   Epoch ->0
    Theta ->[0.54030055 0.74700458 0.60774813 0.53612429 0.39390495 0.66432208
    0.42455597 0.87913902 0.9408577  0.16211897]
   Epoch ->1
    Theta ->[0.5302535  0.75955475 0.61332103 0.5258201  0.35874312 0.67708556
    0.40729141 0.87384035 0.92352302 0.10546534]
```

```
Epoch ->2
 Theta ->[0.52018909 0.76576297 0.61839452 0.5155784  0.31426446 0.68593045
 0.39912291 0.87020696 0.9136364  0.08328092]
Epoch ->3
 Theta ->[0.51163503 0.76889438 0.62203863 0.50672204 0.27727676 0.69178149
 0.39765551 0.86830699 0.90930678 0.07144459]
Epoch ->4
 Theta ->[0.50591337 0.77056051 0.62378851 0.50050643 0.24532125 0.69564221
 0.39109125 0.86756872 0.90765306 0.0584879 ]
Epoch ->5
 Theta ->[0.50177514 0.77150992 0.62403936 0.49571276 0.22302672 0.69828232
 0.38031544 0.86742743 0.90706774 0.04969973]
Epoch ->6
 Theta ->[0.49827412 0.77206431 0.62329137 0.49144947 0.20702191 0.70011169
 0.36859236 0.86757113 0.90687069 0.044538  ]
Epoch ->7
 Theta ->[0.49486354 0.77237337 0.62188139 0.48714269 0.19357881 0.70134625
 0.35755323 0.8678396  0.90680197 0.04040518]
Epoch ->8
 Theta ->[0.49139945 0.77251562 0.61997426 0.48257882 0.18073897 0.70210547
 0.34738862 0.86814663 0.90676844 0.03582938]
Epoch ->9
 Theta ->[0.48790436 0.772535   0.61765985 0.47769309 0.16779264 0.70246642
 0.33751193 0.86844301 0.90673968 0.03014372]
Epoch ->10
 Theta ->[0.48437903 0.77245793 0.6150036  0.4723681  0.15546467 0.70248497
 0.32750971 0.86869972 0.90670787 0.02413786]
Epoch ->11
 Theta ->[0.48082938 0.77230195 0.61204314 0.4664367  0.14469318 0.7022024
 0.31791435 0.86889965 0.90667292 0.01925729]
Epoch ->12
 Theta ->[0.47744018 0.77207913 0.60879011 0.45986275 0.13608127 0.70164984
 0.30932917 0.86903319 0.90663702 0.01504909]


#The MLE estimates are stored in the theta variable, which is the final output of the code.
for i, j in enumerate(theta): # to get the index as well as value
    print(f"{i+1} : {j:.3f}")

 1 : 0.477
 2 : 0.772
 3 : 0.609
 4 : 0.460
 5 : 0.136
 6 : 0.702
 7 : 0.309
 8 : 0.869
```

```
     9 : 0.907
     10 : 0.015


theta

     array([0.47744018, 0.77207913, 0.60879011, 0.45986275, 0.13608127,
            0.70164984, 0.30932917, 0.86903319, 0.90663702, 0.01504909])
```