Permutation

In [ ]:

```
# permutations using library function
from itertools import permutations

# Get all permutations of [1, 2, 3]
perm = permutations([1, 2, 3])

# Print the obtained permutations
for i in list(perm):
    print (i)
```

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

In [ ]:

```
# permutations of given length
from itertools import permutations

# Get all permutations of length 2
perm = permutations([1, 2, 3], 2)

# Print the obtained permutations
for i in list(perm):
    print (i)
```

```
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

Combination

In [ ]:

```
# combinations of given length
from itertools import combinations

# Get all combinations of [1, 2, 3]
# and length 2
comb = combinations([1, 2, 3],2)

# Print the obtained combinations
for i in list(comb):
    print (i)
```

```
(1, 2)
(1, 3)
(2, 3)
```

In [ ]:

```
# Print all combinations with an element-to-itself combination
from itertools import combinations_with_replacement

# Get all combinations of [1, 2, 3] and length 2
comb = combinations_with_replacement([1, 2, 3], 2)
```

```
# Print the obtained combinations
for i in list(comb):
    print (i)
```

```
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)
```

Probability

In [ ]:

```
import numpy as np
# probability of flipping a heads
ph=0.5
#Number of coins flips to simulate
num_flips=25

#simulate coin flips
def flip_coin(N,p=0.5):
  prob=[p,(1-p)]
  return np.random.choice(['H','T'],size=N,p=prob)

#accumulate flips
flips=flip_coin(num_flips,ph)

#count heads
num_heads=np.sum(flips=='H')

#Display
print("flips: "," ".join(flips))
print(f"Number of Heads: {num_heads}")
print(f'P(H)={num_heads/num_flips} (Number of Heads/Total Flips)')
```

```
flips:  T T H H T T H H H T H H T H T H T H T T T T T T H
Number of Heads: 11
P(H)=0.44 (Number of Heads/Total Flips)
```

Bayes

In [ ]:

```
import numpy as np
import scipy.stats

def compute_posterior(prior, sensitivity, specificity):
    likelihood = sensitivity  # p(test|disease present)
    marginal_likelihood = sensitivity * prior + (1 - specificity) * (1 - prior)
    posterior = (likelihood * prior) / marginal_likelihood
    return(posterior)

sensitivity = 0.90
specificity = 0.99
prior_values = 0.074
posterior_values = compute_posterior(prior_values, sensitivity, specificity)
print(prior_values,posterior_values)
```

```
0.074 0.8779330345373055
```

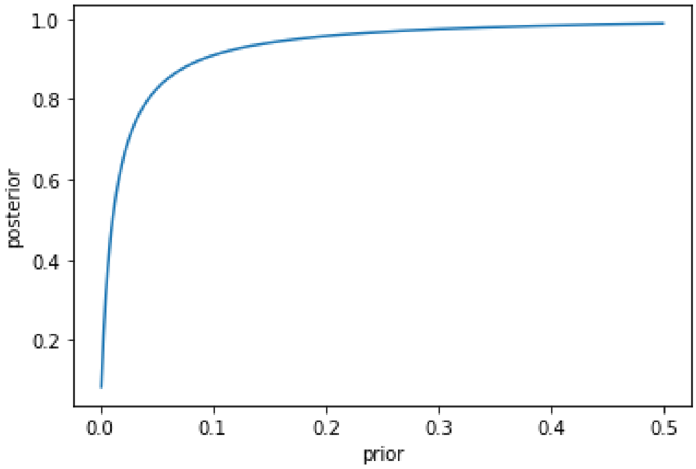
In [ ]:

```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt


def compute_posterior(prior, sensitivity, specificity):
```

```python
    likelihood = sensitivity  # p(test|disease present)
    marginal_likelihood = sensitivity * prior + (1 - specificity) * (1 - prior)
    posterior = (likelihood * prior) / marginal_likelihood
    return(posterior)

sensitivity = 0.90
specificity = 0.99
prior_values = np.arange(0.001, 0.5, 0.001)
posterior_values = compute_posterior(prior_values, sensitivity, specificity)
plt.plot(prior_values, posterior_values)
plt.xlabel('prior')
_ = plt.ylabel('posterior')
```



In [ ]:

```python
def bayes_theorem(p_b, p_g_given_b, p_g_given_not_b):
   # calculate P(not B)
   not_b = 1 - p_b
   # calculate P(G)
   p_g = p_g_given_b * p_b + p_g_given_not_b * not_b
   # calculate P(B|G)
   p_b_given_g = (p_g_given_b * p_b) / p_g
   return p_b_given_g
#P(B)
p_b = 1/4
# P(G|B)
p_g_given_b = 1
# P(G|notB)
p_g_given_not_b = 1/3
# calculate P(B|G)
result = bayes_theorem(p_b, p_g_given_b, p_g_given_not_b)
# print result
print('P(B|G) = %.2f%%' % (result * 100))
```

```
P(B|G) = 50.00%
```