Exercise 1

Group ID: TAU03E

Import libraries

In [26]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Task 1.

Answer contains two real-world examples where mathematical modeling can be used.

1. Find the Shortest Path to the Nearest Market

Sometimes time is of the essence, and knowing the shortest route to the nearest market becomes important. The situation could be urgent, like needing cleaning supplies quickly, or it’s close to 9 PM. This problem can be modeled using graphs. The nearby paths/routes can form a network, where the paths are represented as edges. Each path is connected to others just as they are in the real neighborhood, and weights are assigned to each path based on their lengths.

The shortest path can be found within the graph using algorithms like Dijkstra's algorithm. With Dijkstra, we can find the shortest routes in the graph to the nearby market, or any other destination. These algorithms and methods are already in use in real-world applications such as map services.

2. Household Energy Consumption problem.

The idea is to mathematically model consumption rates. Household may have multiple devices which consume plenty of energy. Mathematically the consumption can be modeled. For example, the model can contain the biggest electricity consumers and gainers such as solar panels. The simple way is to combine all of the wanted consumers and gainers are combined into the model with coefficients of their relevancy.

For more advanced cases models can hold real-time information about energy prices, weather conditions (which can affect prices) and time-dependent price changes such as day or night. The model can be used to optimize and/or predict energy consumption. Furthermore, the model can be combined with a smart home system to automate energy consumption optimization.

Task 2.

Let's use the SIR model, which is a widely used model to describe the spread of an epidemic. In the model $S(t)$ stands for the count of susceptibles, $I(t)$ the count of infected individuals and $R(t)$ the count of recovered individuals at a day t.

We are especially interested in the pace of spread for the epidemic, so we can look at the rate of change in the random variables of the model.

The count of susceptibles changes at a rate

$$\frac{dS}{dt} = -\beta IS$$

where β stands for the average contacts per day for an individual multiplied by the probability for a susceptible to contract the disease from an infected individual. This we can define as the average amount of people x in the range of 1 meter in the population multiplied by the probability 0.20 if there are more than 5 people in range, else 0. We get

$$\beta = \begin{cases} 0.20x, & \textbf{if } x > 6 \\ 0, & \textbf{if } x \leq 5 \end{cases}$$

The rate of change in infected individuals is defined as

$$\frac{dI}{dt} = \beta IS - \lambda I$$

where λ stands for the count of days that an individual stays infected. We can define this by the probability of recovery $\lambda = 0.08$.

Lastly, we have the rate of change in recoveries, which is defined as

$$\frac{dR}{dt} = \lambda I$$

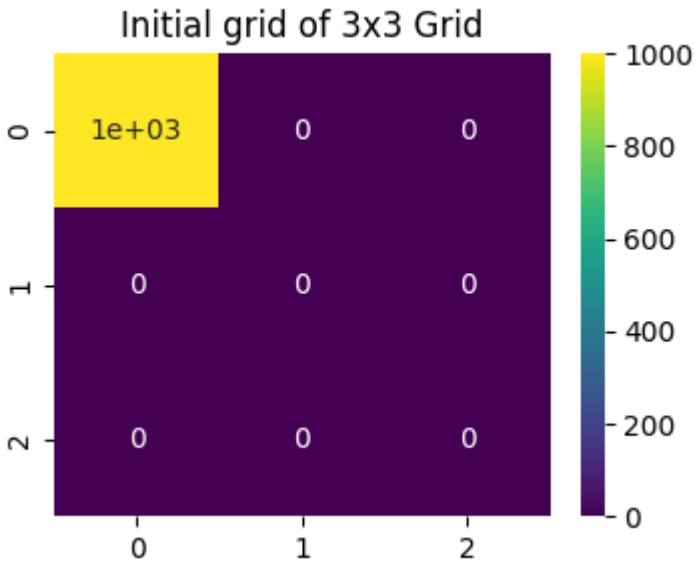
At first, we know that the count of infected individuals $I = 500$ and susceptibles $S = N - 500$, where N is the total amount of people in the monitored area. Knowing these, we can further estimate the spread of the disease.

Task 3.

b)

Initially, there is 1000 butterflies in a corner square

In [27]:
```
grid = np.zeros((3,3))
grid[0,0] = 1000

plt.figure(figsize=(4, 3))
sns.heatmap(grid, annot=True, cmap="viridis", cbar=True, vmin=0, vmax=100
plt.title('Initial grid of 3x3 Grid')
plt.show()
```



Collect the coordinates of adjacent squares for each square (let's assume that the butterflies can cross diagonally)

In [28]:
```
adjacent_squares = {
    (0,0): [(0,1), (1,0), (1,1)],
    (0,1): [(0,0), (0,2), (1,0), (1,1), (1,2)],
    (0,2): [(0,1), (1,1), (1,2)],
    (1,0): [(0,0), (0,1), (1,1), (2,0), (2,1)],
    (1,1): [(0,0), (0,1), (0,2), (1,0), (1,2), (2,0), (2,1), (2,2)],
    (1,2): [(0,1), (0,2), (1,1), (2,1), (2,2)],
    (2,0): [(1,0), (1,1), (2,1)],
    (2,1): [(1,0), (1,1), (1,2), (2,0), (2,2)],
    (2,2): [(1,1), (1,2), (2,1)],
}
```

There is 0.2 probability that the butterflies will cross the boundry to an adjancet square within an hour

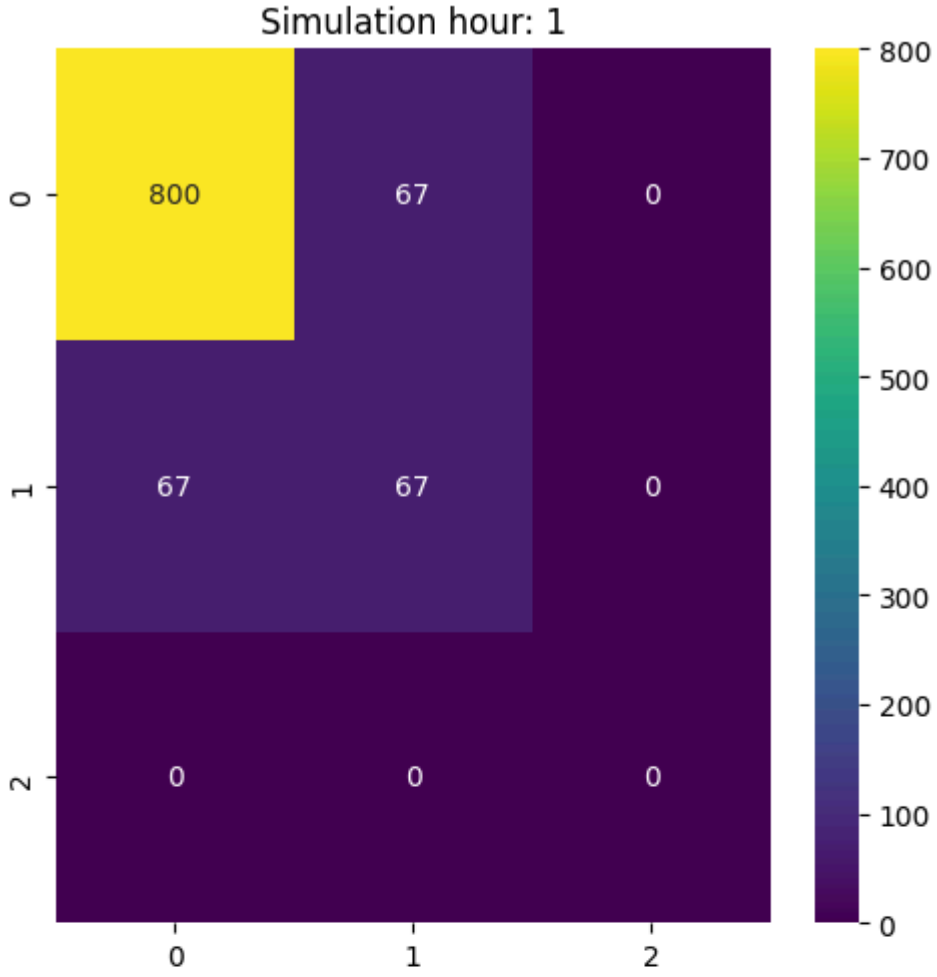
In [29]:
```
p = 0.2
```

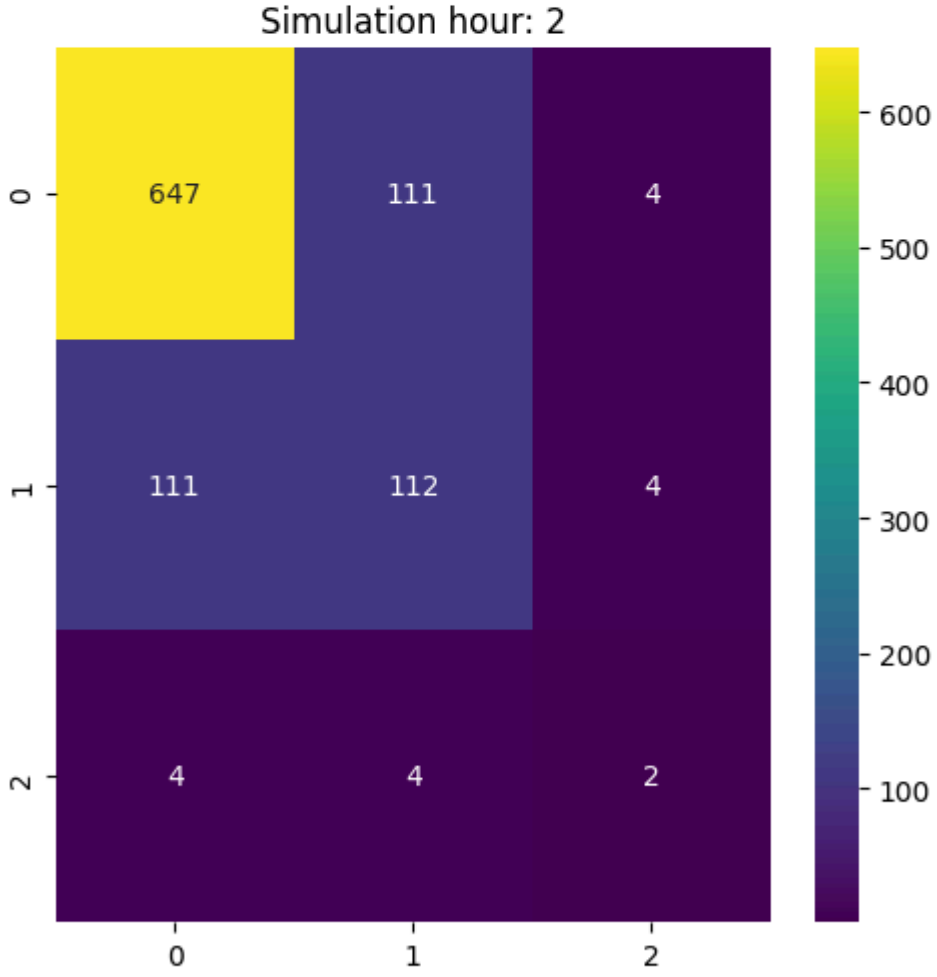
Simulate the grid over hours

In [30]:
```
hours = 100
grid_by_hour = {}
for hour in range(hours):
    spread = dict.fromkeys(adjacent_squares.keys(), 0)

    for square, adj_squares in adjacent_squares.items():
        n_butterflies = grid[square]
        for adj_square in adj_squares: spread[adj_square] += (n_butterfli

    grid *= (1-p)
    for square, added_butterflies in spread.items(): grid[square] += adde

    grid_by_hour[hour+1] = grid.copy()
```

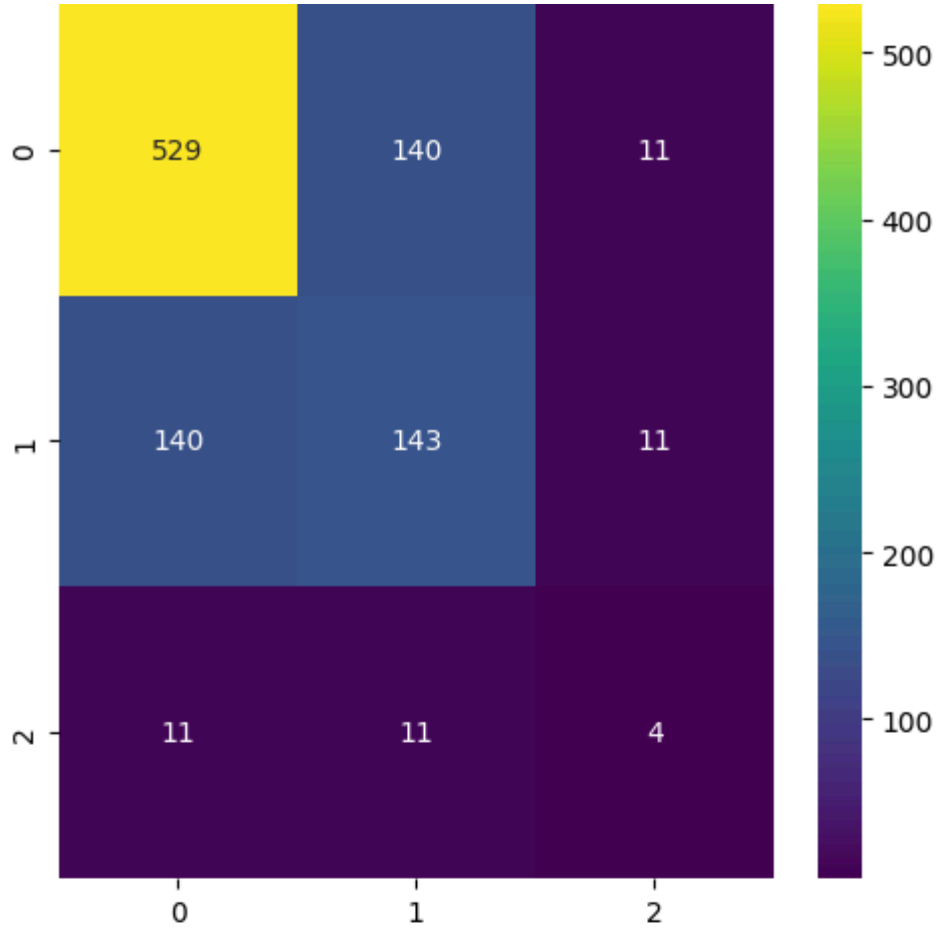
Visualize grid evolution

In [36]:
```
hours = [1,2,3, 10, 25, 50, 100]
fig, axs = plt.subplots(len(hours), 1, figsize=(5, len(hours)*5))

for idx, hour in enumerate(hours):
    sns.heatmap(grid_by_hour[hour], annot=True, cmap="viridis", cbar=True
    axs[idx].set_title(f"Simulation hour: {hour}")

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

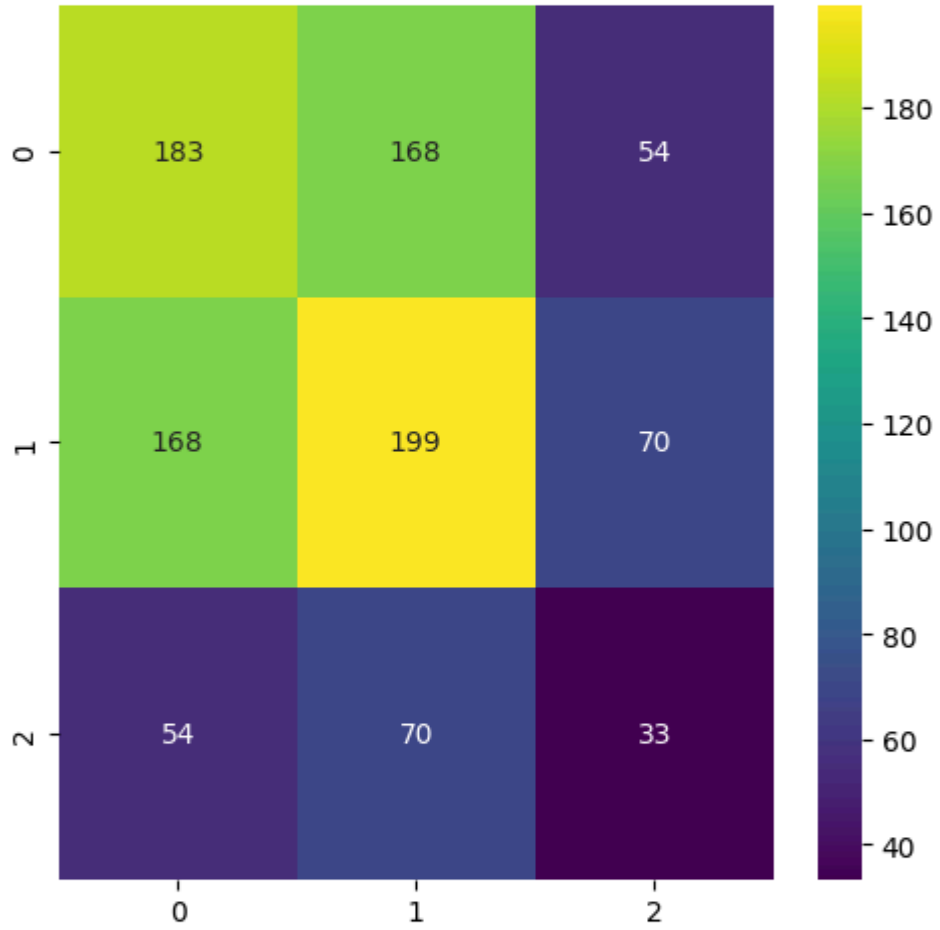




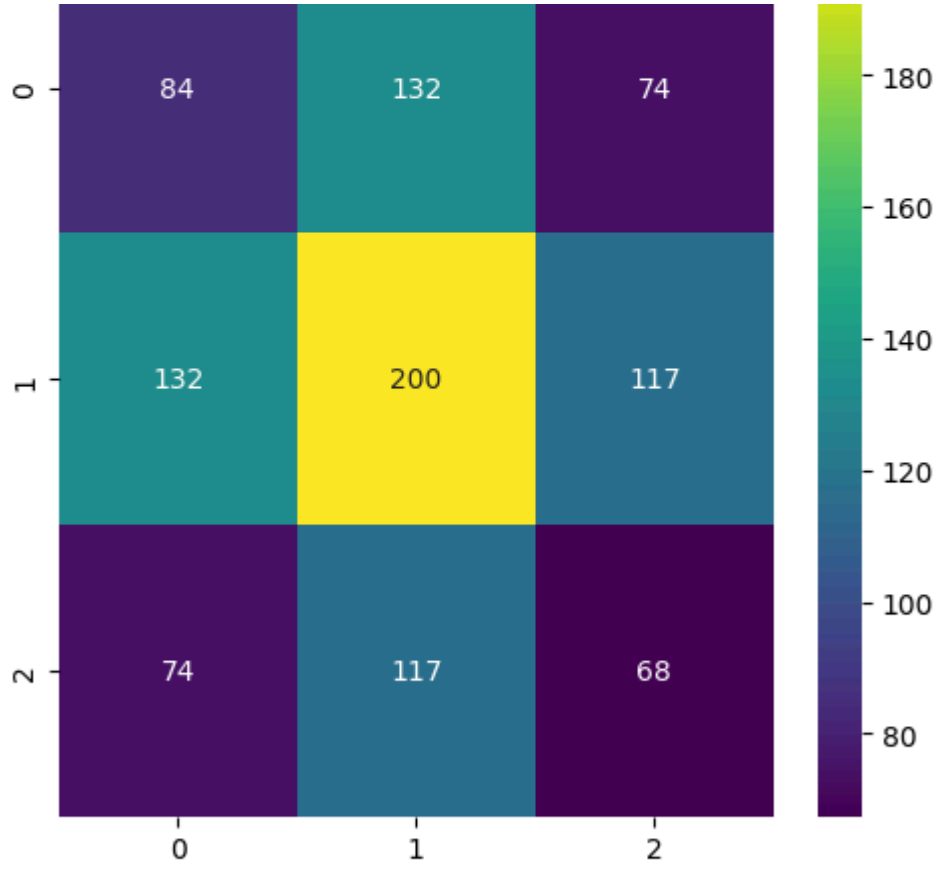
Simulation hour: 3



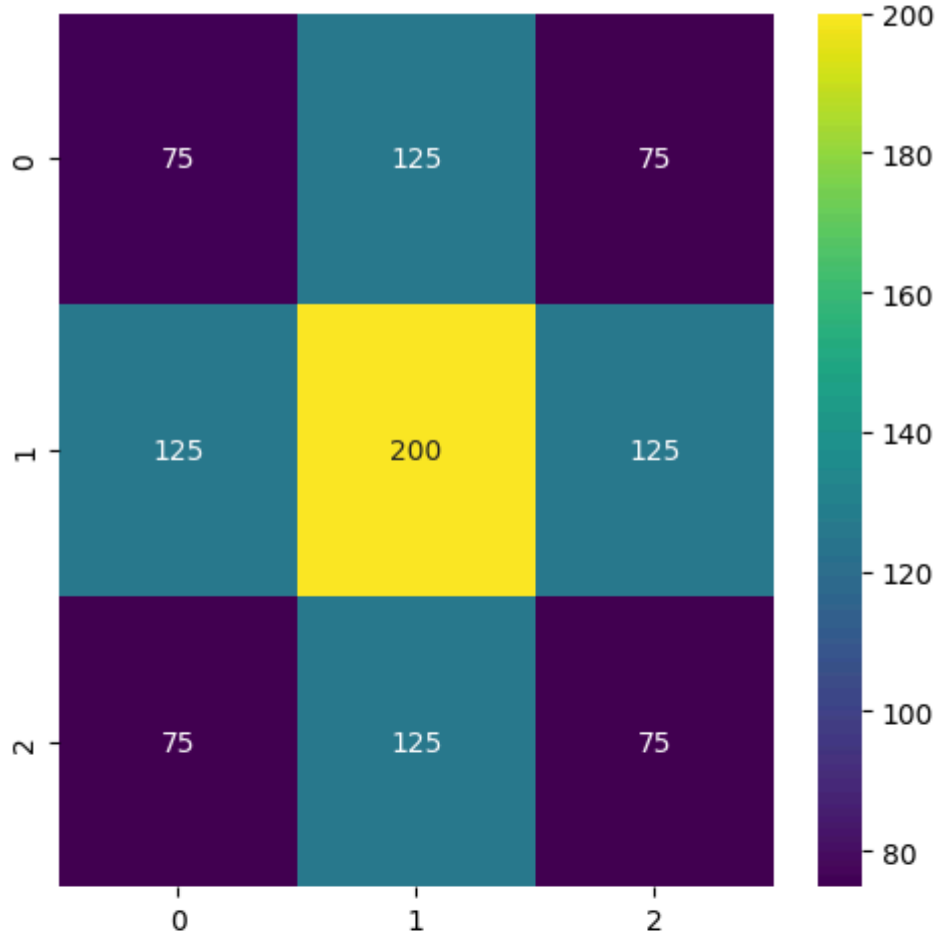
Simulation hour: 10



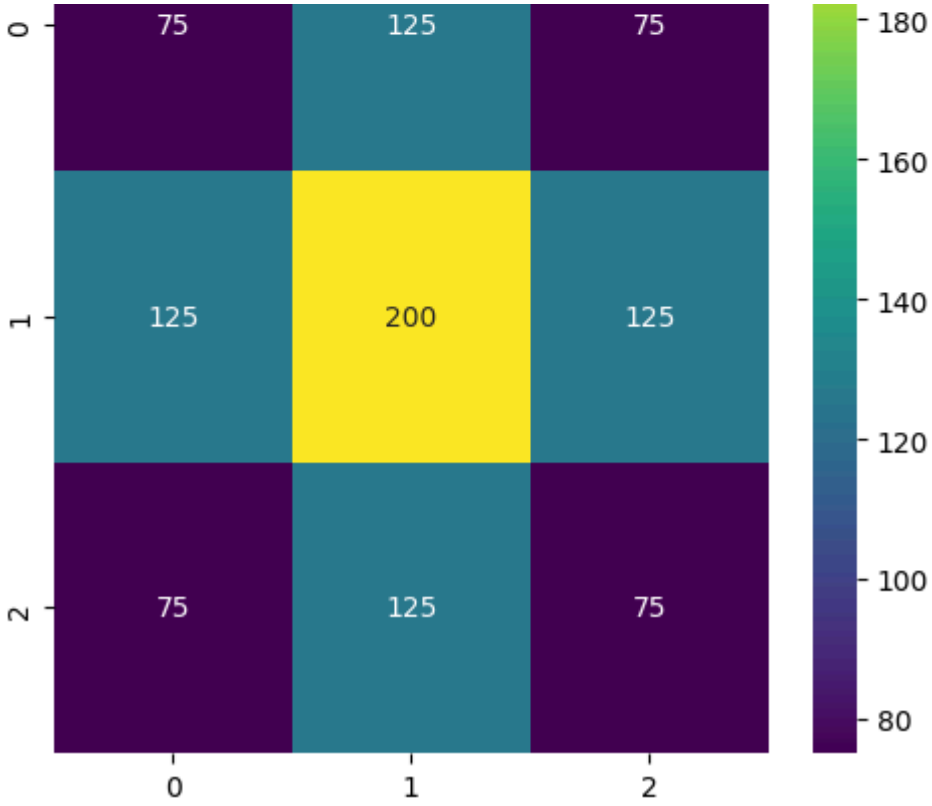
Simulation hour: 25



Simulation hour: 50



Simulation hour: 100



In [ ]: