**5 prompts in this document**

1. PROMT 1 – Dictionary to store mean length of stays for each patient type
2. PROMT 2 – If statement to create a specific length of stay if patient is transferred to ESD
3. PROMT 3 – Recalculating ESD Percentage
4. Prompt 4 – Code for bed capacity scenarios
5. Prompt 5 – Code for bed capacity scenarios Time to bed trouble shoot

**Issy vague comments on model for Destinee**

**PROMT 1 – Dictionary to store mean length of stays for each patient type**

## Main command

Provide only the Python code to create a dictionary that stores the mean and standard deviation of length of stay parameters for different patient types in an acute stroke unit. Create a length of stay distributions dictionary using Lognormal distributions for each patient type. Show how to retrieve the length of stay distribution for a specific patient type . Do not include any other code outside this, using SimPy 4 and Python 3.10.

## General model logic

The simulation uses lognormal distributions to model length of stay for different patient types in an acute stroke unit. Each patient type has a specific mean and standard deviation for their length of stay:

Stroke: 7.4 days, SD = 8.6

TIA: 1.8 days, SD = 2.3

ComplexNeuro: 4.0 days, SD = 5.0

Other: 3.8 days, SD = 5.2

The general structure of the simulation is distribution classes, model parameters, AcuteStrokeUnit class (holds arrival generators) , and AcutePatient class (stroke unit process).

## Simulation methodology

A Patient class where each patient object has a Lognormal LOS distribution. Set a random seed for reproducibility. Lognormal distribution classes are previously defined within the code don’t do that again.

**OUTPUT**

#Model parameters

# acute stroke unit length of stay (lognormal distribution) for each patient type

STAY\_PARAMS = {

"Stroke": (7.4, 8.6), # Mean = 7.4, SD = 8.6

"TIA": (1.8, 2.3), # Mean = 1.8, SD = 2.3

"ComplexNeuro": (4.0, 5.0), # Mean = 4.0, SD = 5.0

"Other": (3.8, 5.2) # Mean = 3.8, SD = 5.2”

#class scenario

# Length of stay distribution (lognormal) for each patient type using STAY\_PARAMS

self.length\_of\_stay\_dist = {

patient\_type: Lognormal(mean, std, random\_seed=SEEDS[4])

for patient\_type, (mean, std) in STAY\_PARAMS.items()

}

#class acute patient

# Get length of stay distribution from Scenario

self.length\_of\_stay\_dist = args.length\_of\_stay\_dist[patient\_type]

happy with this output, works and when running the model, the mean length of stays for each patient group align with the parameters set.

**PROMT 2 – If statement to create a specific length of stay if patient is transferred to ESD**

## Main command

Provide only the Python code for the if-statement that checks if a Stroke patient is transferred to ESD. If transferred, it should use a Lognormal(4.6, 4.8) distribution with a set random seed for reproducibility, it should use the standard Stroke LOS distribution. Do not include any other code outside this if-statement, using SimPy 4 and Python 3.10.

## General model logic

The simulation uses lognormal distributions to model length of stay for different patient types in an acute stroke unit. Each patient type has a specific mean and standard deviation for their length of stay:

Stroke: 7.4 days, SD = 8.6

TIA: 1.8 days, SD = 2.3

ComplexNeuro: 4.0 days, SD = 5.0

Other: 3.8 days, SD = 5.2

There is a Bernoulli-distributed probability of ESD transfer. If a Stroke patient is transferred to ESD, their mean LOS changes to 4.6 days, SD = 4.8.

The general structure of the simulation is distribution classes, model parameters, AcuteStrokeUnit class (holds arrival generators) , and AcutePatient class (stroke unit process).

## Simulation methodology

A Patient class where each patient object has a Lognormal LOS distribution and a Bernoulli-distributed ESD transfer probability. A check to determine whether a stroke patient is transferred to ESD and if so, use the updated LOS distribution (4.6, 4.8). Set random seed for reproducibility.

## Further prompt

Lognormal distribution classes are previously defined and set the random seed within length of stay bracket and using the previously defined length of stay distribution if not applicable.

**OUTPUT**

# Check if the patient is a stroke patient and is transferred to ESD

if self.patient\_type == "Stroke" and self.esd\_transfer.sample():

# If stroke patient is transferred to ESD, use new LOS distribution (4.6, 4.8)

length\_of\_stay = Lognormal(4.6, 4.8, random\_seed=SEEDS[6]).sample()

self.time\_to\_esd = self.env.now # Time of transfer to ESD

trace(f'Patient {self.identifier} (Stroke) transferred to ESD at {self.env.now:.3f}')

else:

# Otherwise, use standard stroke LOS

length\_of\_stay = self.length\_of\_stay\_dist.sample()

This is good and works however, “**This will sample the same length of stay for all ESD patients. The reason is that it creates a new Lognormal distribution each time it is called and passes in the same seed.  Then it calls sample.  You should create your Lognormal distribution elsewhere and pass it to the process.  In class, for example, we developed a Scenario class where we setup and seeded all distributions.”.**

**BOLD IS EMAIL FROM TOM**

This is what is should look like/looks like now in the code:

#Model parameters

# length of stay parameters (lognormal)

STAY\_PARAMS = "Stroke\_ESD": (4.6, 4.8) # Mean = 4.6, SD = 4.8

#class scenario

# Initialize Length of Stay distributions for all patient types

self.length\_of\_stay\_dist = {

patient\_type: Lognormal(mean, std, random\_seed=SEEDS[4])

for patient\_type, (mean, std) in STAY\_PARAMS.items()

}

#class acute patient

# ESD Transfer logic

if self.is\_esd:

self.time\_to\_esd = self.env.now - bed\_time

length\_of\_stay = self.args.length\_of\_stay\_dist["Stroke\_ESD"].sample()

else:

length\_of\_stay = self.length\_of\_stay\_dist.sample() ()

**PROMT 3 – Recalculating ESD Percentage**

## Main command

Recalculate the ESD (Early Supported Discharge) percentage for a stroke patient flow model. The model now only considers two pathways: ESD and Other. Use the given percentages to determine the new probability for ESD. Use Python 3.10. Do not include any other code outside this calculation.

## General model logic

The model uses a Bernoulli distribution to determine patient pathways in a stroke unit. The original percentages for patient destinations were:

Stroke Rehab: 24%

ESD: 13%

Other: 63%

Recalculate these percentages considering only ESD and Other pathways. It’s to calculate this:

# transfer (bernoulli) PROB\_ESD = 0.13

## Simulation methodology

Provide the resulting probability as PROB\_ESD, which will be used in the model's Bernoulli distribution for patient pathway assignment.

**OUTPUT**

Given the original distribution:

Rehab: 24%  
ESD: 13%  
Other: 63%

Now that we're only considering ESD and Other pathways:

Calculate the new total (ESD + Other):  
13% + 63% = 76%

Calculate the new ESD probability:  
New PROB\_ESD = 13% / 76% ≈ 0.1711

Therefore, the recalculated ESD probability is:

PROB\_ESD = 0.1711

This means that after removing the Rehab pathway, the probability of a patient being assigned to the ESD pathway increases from 13% to approximately 17.11%.

This prompt is good, no need for further prompts, llm seem to work well to basic things.

**Prompt 4 – Code for bed capacity scenarios**

## Main Command

Provide only the Python code to run different scenarios of an Acute Stroke Unit with different bed capacity including the warm-up period, calculate bed utilization, results visualisation. Use SimPy 4 and Python 3.10. Include functions for running the simulation, calculating bed utilization, creating scenarios, and plotting results. Do not include any other code outside this.

## General Model Logic

The warm-up period is previously defined at 600 days, 6 replications and the run period for the model is 1445. The original scenario uses a bed capacity of 10, create 3 more scenarios increasing the number of beds by 10 each time.

The general structure of the simulation is distribution classes, model parameters, AcuteStrokeUnit class (holds arrival generators) , and AcutePatient class (stroke unit process).

The code should include:

Run parameters

Bed utilization calculation

Simulation run function

Scenario generation for different bed counts

Results visualization

## Simulation Methodology

Matplotlib should be used to make the bar chart.

# Further prompts

Add warm-up and replications into the model.

Store the results from the scenarios into a dictionary and then produce a bar chart using those results.

Ensure bed utilisation does not exceed 100%

**OUTPUT**

# Constants for warm-up and results collection  
AUDIT\_INTERVAL = 1  # Minutes per audit interval  
WARM\_UP\_INTERVALS = 600  # Number of intervals for warm-up  
WARM\_UP = WARM\_UP\_INTERVALS \* AUDIT\_INTERVAL  # Warm-up period in minutes  
RC\_PERIOD = 1440  # Results collection period (1 day in minutes)

def calculate\_bed\_utilization(model, start\_time):  
    """Calculate bed utilization after the warm-up period, capped at 100%."""  
    total\_bed\_time = sum(min(p.length\_of\_stay, model.env.now - start\_time)   
                         for p in model.patients   
                         if p.length\_of\_stay > 0 and (model.env.now - p.time\_to\_bed) >= start\_time)  
    total\_time = (model.env.now - start\_time) \* model.args.unit\_beds\_capacity  
    utilization = (total\_bed\_time / total\_time) \* 100 if total\_time > 0 else 0  
    return min(utilization, 100)  # Cap at 100%

def run\_simulation(scenario):  
    """Runs a single simulation with a given scenario and returns bed utilization."""  
    env = simpy.Environment()  
    model = AcuteStrokeUnit(env, scenario)

    # Start the arrival processes  
    model.start\_patient\_arrivals()

    # Run the simulation for warm-up + results collection period  
    env.run(until=WARM\_UP + RC\_PERIOD)

    # Calculate bed utilization only for the results collection period after warm-up  
    bed\_utilization = calculate\_bed\_utilization(model, WARM\_UP)  
    return bed\_utilization

# Create scenarios  
def get\_scenarios():  
    """Returns a dictionary of scenarios with different bed counts."""  
    scenarios = {}  
    bed\_counts = [10, 20, 30, 40]  # Example bed counts  
    for bed\_count in bed\_counts:  
        scenario = Scenario()  
        scenario.unit\_beds = bed\_count  
        scenario.unit\_beds\_capacity = bed\_count  
        scenarios[f'beds\_{bed\_count}'] = scenario  
    return scenarios

scenarios = get\_scenarios()

# Store results  
results = {}

# Run simulations for each scenario and store results  
for scenario\_name, scenario in scenarios.items():  
    bed\_utilization = run\_simulation(scenario)  
    results[scenario\_name] = bed\_utilization  
    print(f"Scenario: {scenario\_name}, Bed Utilization: {bed\_utilization:.2f}%")

# Create bar chart  
plt.figure(figsize=(10, 6))  
plt.bar(results.keys(), results.values(), color='skyblue')  
plt.title('Bed Utilization by Scenario (After Warm-Up)')  
plt.xlabel('Scenarios')  
plt.ylabel('Bed Utilization (%)')  
plt.ylim(0, 100)  # Set y-axis limit from 0 to 100%

# Add value labels on top of each bar  
for i, (scenario, utilization) in enumerate(results.items()):  
    plt.text(i, utilization, f'{utilization:.2f}%', ha='center', va='bottom')

# Rotate x-axis labels for better readability  
plt.xticks(rotation=45, ha='right')

# Adjust layout and display the chart  
plt.tight\_layout()  
plt.show()

This prompt was successfully however, took multiple further prompts to receive a outputs that ran without errors and with all the critical information.

**Prompt 5 – Code for bed capacity scenarios time to bed trouble shoot**

## Main command

Provide only the Python code to create troubleshooting functions for an acute stroke unit simulation model using SimPy 4 and Python 3.10. Focus on debugging metrics like bed utilization and patient flow. Include clear instructions on where to add each piece of code in the existing model structure and how to call them during or after the simulation run. Do not include any other code outside these troubleshooting functions.

## General Model Logic

The general structure of the simulation includes distribution classes, model parameters, an AcuteStrokeUnitclass (which holds arrival generators), and an AcutePatient class (which manages patient processes). I am encountering issues with my simulation, particularly with metrics like bed utilization and patient flow. I would like you to generate troubleshooting codes

Please include clear instructions on where each piece of code should be added in my existing model and how to call them during or after the simulation run

## Simulation Methodology

Model parameters (like mean inter-arrival times, length of stay parameters) are previously defined alongside distribution classes .

Outputs such as wait\_for\_bed are already stored as variables.

**OUTPUT**

1. Check bed capacity:

Ensure that N\_BEDS in your model parameters is set to a reasonable number (e.g., 10 or more).

Verify that self.unit\_beds in AcuteStrokeUnit is correctly initialized with this capacity.

1. Examine inter arrival rates:

scenario = Scenario()

iat\_mapping = {

"Stroke": "MEAN\_IAT\_STROKE",

"TIA": "MEAN\_IAT\_TIA",

"ComplexNeuro": "MEAN\_IAT\_CN",

"Other": "MEAN\_IAT\_OTHER"

}

for patient\_type in ["Stroke", "TIA", "ComplexNeuro", "Other"]:

samples = [scenario.arrival\_dist[patient\_type].sample() for \_ in range(10000)]

mean\_sampled = np.mean(samples)

print(f"Expected Mean IAT for {patient\_type}: {globals()[iat\_mapping[patient\_type]]}")

print(f"Sampled Mean IAT for {patient\_type}: {mean\_sampled}")

1. Verify time calculations:

In the AcutePatient class, check the calculation of self.time\_to\_bed. It should be:  
self.time\_to\_bed = bed\_time - arrival\_time

1. Inspect queue management:

Ensure patients are correctly entering and leaving the queue for beds.

Check if patients are being removed from the queue when they get a bed.

1. Review simulation run time:

Confirm that the simulation runs for the intended duration and that the warm-up period is appropriate.

1. Check for resource leaks:

Ensure that beds are properly released after patient discharge.

1. Debug with print statements:

Add print statements to track patient movement, especially around bed assignment and release. Use tracing.

1. Validate random number generation:

Ensure that the random number generators for inter-arrival times and length of stay are functioning correctly.

# Check LOS distributions

for patient\_type, (mean, std) in STAY\_PARAMS.items():

samples = [scenario.length\_of\_stay\_dist[patient\_type].sample() for \_ in range(10000)]

mean\_sampled = np.mean(samples)

std\_sampled = np.std(samples)

print(f"{patient\_type} LOS - Expected Mean: {mean}, Sampled Mean: {mean\_sampled}")

print(f"{patient\_type} LOS - Expected Std: {std}, Sampled Std: {std\_sampled}")

## Further prompts

I’ve been though all the suggestions and they all match indicating these to not be the issue, attached are the outputs for 2) and 8).

“Expected Mean IAT for Stroke: 1.2

Sampled Mean IAT for Stroke: 1.2024384116422209

Expected Mean IAT for TIA: 9.3

Sampled Mean IAT for TIA: 9.339204603897512

Expected Mean IAT for ComplexNeuro: 3.6

Sampled Mean IAT for ComplexNeuro: 3.5535958120930102

Expected Mean IAT for Other: 3.2

Sampled Mean IAT for Other: 3.2649366936884276”

“Stroke LOS - Expected Mean: 7.4, Sampled Mean: 7.355859627935247

Stroke LOS - Expected Std: 8.6, Sampled Std: 8.303862267020325

Stroke\_ESD LOS - Expected Mean: 4.6, Sampled Mean: 4.60475785191077

Stroke\_ESD LOS - Expected Std: 4.8, Sampled Std: 4.657452536504659

TIA LOS - Expected Mean: 1.8, Sampled Mean: 1.7786686649498653

TIA LOS - Expected Std: 2.3, Sampled Std: 2.2324721560890324

ComplexNeuro LOS - Expected Mean: 4.0, Sampled Mean: 3.9833220381356305

ComplexNeuro LOS - Expected Std: 5.0, Sampled Std: 4.816294450733544

Other LOS - Expected Mean: 3.8, Sampled Mean: 3.739324505462664

Other LOS - Expected Std: 5.2, Sampled Std: 4.996354412956359”

Next was suggested tracking individual patients entering the system with the results as followed:

Total patients processed:

Stroke: 2371

TIA: 325

ComplexNeuro: 858

Other: 930

With a bed utilisation of 99% the LLM concluded that the only ways to reduce the bottleneck of patients was the increase the beds, assigning priority for some patient groups, both which would change the basis of the model, indicating the LLM was not able to find the root of the high very large wait\_for\_bed times highlighting the limitations of the LLM. Lots of further error prompts were used to make the code the LLM provided compatible with the simulation.