澳門大學

UNIVERSIDADE DE MACAU

UNIVERSITY OF MACAU

ISOM3025-002
Business Modeling and Simulation

Report

Group 30: University Study Room Scenario Simulation

By:
Toh Yong Li DC229938
Ho Lai Chon BC001661
Luo Junlin BB906647

1. Motivations & problem description

Study rooms are a common student studying resource in universities. However, the number of study rooms to open from time to time may be difficult to determine. Opening too many study rooms will result in oversupply, affecting the expenses of the university. Opening too few study rooms will result in unsatisfied demand that can lead to a decrease in the universities' reputation. The university needs to choose a policy to be used for opening the study rooms in order to avoid the waste of resources. The objective is to decrease study room costs while trying to satisfy the demands as much as possible.

The system simulates and predicts demand for the study rooms. It uses past data to simulate the study rooms' past usage patterns for result simulation. The time variable is the main factor affecting the demand for the study rooms. The university can then use the model to simulate different policy implementations and have more information about the study room demand and how to choose the right policy.

These problems may motivate the universities to use this system as it analyses the demand level throughout the year, simulates various scenarios and helps to make the best decision regarding opening student study rooms. Then, the universities can greatly reduce the costs in order to increase profit.

2. Generating data

Due to the lack of real-life data, our simulation can be achieved by generating pseudo data of time-series demand of study rooms according to a pattern that resembles real-life situations.

To generate pseudo data, we first specify the start and end dates, then generate the range of dates. We created a function to generate data as shown in Figure 2.1.

```python
# data generation
def generate_data(date: date) -> int:
    week = date.isocalendar()[1]
    weekday = date.weekday()
    lam = max(average + week_factor[week-1] + day_factor[weekday], 0.5)
    demand = np.random.poisson(lam=lam,size=1)[0]
    return demand
```

Figure 2.1: Data Generation function

In the function, we generate a daily random Poisson distribution with the value of lambda depending on the week and day factor. The week factor is a list of integers specifying the demand level for each week of the year. The day factor is a list of integers specifying the demand level for each day of the week. Since every day and week has a different level of demand, we sum them up with the average level of demand to determine the value for lambda, while making sure the value of lambda does not fall below 0.

3. Input data analysis & modelling

3.1. Input data analysis

Analysing the pseudo data, we can see that there is an increase in demand level from February to June and from September to December as compared to the month of June to August and the month from December to February as shown in Figure 3.1 below.

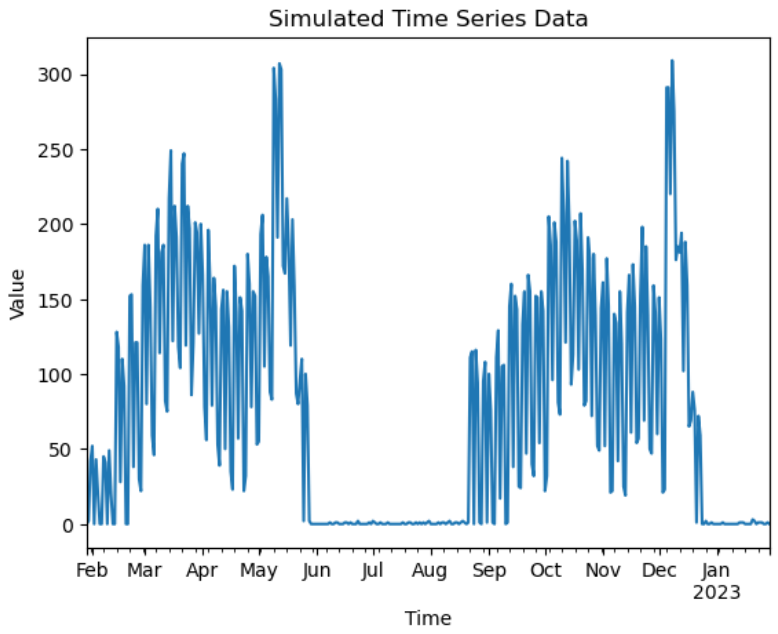


Figure 3.1.1 Simulated Time Series Data

These data are reasonable as a University Semester 1 starts from August to December and Semester 2 starts from January to May. Also, the mid-term test is usually in March and October, while the end-term exam is in May and December. Hence, an obvious increase in demand level during these periods should be observed.

3.2. Modelling

Decision making variables

Decision making of the study room scenario boils down to determining how many study rooms to open for each day. In this model, we set two decision variables in motion that make up the policy to open the number of study rooms.

method to determine the expectation of demand: Our model determines how many study rooms to open based on an preemptive estimation of the demand for future periods. The method to determine expectation of future demands will affect the university's estimation of the demand, thus affecting the number of study rooms the university decides to open.

buffer: According to whether the university wants to cover more demand or reduce more study room cost, a buffer is used to add a linear change to the number of opened study rooms on top of the expectation of demand.

For each day:
Number of opened study room = expectation of demand + buffer

Cost variables

In our model, there will be 2 main factors affecting the profit of the university this year. A shortage of study rooms opened will cause dissatisfaction of the students, giving rise to students' complaints, which causes loss of reputation for the university, leading to ***tuition loss*** (a decrease in tuition fee profit due to a decreasing the number of freshmen) in the future year. An oversupplied number of study rooms could possibly possess a large sum of ***study room costs*** payable by the University.

Thus, we define 2 variables to constitute **tuition loss:**
tuition loss per student loss*: the total tuition loss after discounting the time value of money for this year for each student loss next year.*
impact of unsatisfied demand (α): *how many student losses next year would a single unsatisfied study room demand would result in.*
total tuition loss = tuition loss per student loss * α * unsatisfied demand

For study room cost, we define a single variable:

daily study room cost: how much would opening a study room for a day cost.
total study room cost = daily study room cost * number of opened study rooms for the day

Cost function

total loss = total tuition loss + daily study room cost
The goal of the model is to minimise the total loss per above.

4. Simulation and programming

Our simulation process simulates each day's statistics throughout the year. For each day, we use the same function that was used to generate the pseudo-past data to simulate real-time demand, use the policy adopted to determine the number of study rooms to open, then we calculate all relevant statistics for each day and make a one-year summary.

4.1. Determine the expectation of today's demand

Initially, we use a simple method to determine the expectation of today's demand by calculating the mean value of all the historical data as shown in Figure 4.1.1 below.

```
def simple(today: date, d: DataFrame) -> int:
    # always return the mean value of the past year data

    return round(data['demand'].mean())
```

Figure 4.1.1: The Simple Method

Then, we tried to improve upon the simple method by introducing 4 different forecast methods that forecast today's study room demand from the generated past year's pseudo demands.

Forecast method 1: Using yesterday's demand data as the forecast value for today's demand expectation.

```python
def forecast1(today: date, data: DataFrame) -> int:
    """
        The number of study room opened today is based on yesterday data.
    """
    yesterday: date = today - timedelta(days=1)
    yesterday_pd = pd.to_datetime(yesterday)
    expectation: int = data.loc[yesterday_pd, "demand"]

    return expectation
```

Figure 4.1.2: Forecast Method 1

Forecast method 2: Using last week's demands' mean as the forecast value for this week's data.

```python
def forecast2(today: date, data: DataFrame) -> int:
    """
        The number of study room opened this week is based on lask week data mean.
    """
    # all the dates data that is included in calculating the expectation for today
    accountables: List(int) = []

    weekday: int = today.weekday()
    date_back: date = today - timedelta(days=weekday+1)
    for i in range(7):
        date = date_back - timedelta(days=i)
        date_pd = pd.to_datetime(date)
        accountables.append(data.loc[date_pd, "demand"])

    expectation = round(np.mean(accountables))
    return expectation
```

Figure 4.1.3 Forecast Method 2

Forecast Method 3: Using last month's demands' mean as the forecast value for this month's data.

```python
def forecast3(today: date, data: DataFrame) -> int:
    """
        The number of study room opened this month is based on last month data mean.
    """
    last_month: int = find_previous_month(today.month)
    year: int = today.year-1 if last_month == 12 else today.year
    first_day = date(year, last_month, 1)

    accountables: List(int) = []
    d = first_day
    this_month = d.month
    while d.month == this_month:
        date_pd = pd.to_datetime(d)
        accountables.append(data.loc[date_pd, "demand"])
        d = d + timedelta(days=1)
    expectation: int = round(np.mean(accountables))

    return expectation
```

Figure 4.1.4 Forecast Method 3

Forecast Method 4: Using the generalised linear model - Poisson regression model as the forecasting model to forecast today's demand according to the past year's demand. We use a Python package to fit the model and use it to forecast the demand.

log(demand) = β0 + β1 * month

where:

- log(demand) is the logarithm of the demand count for each day
- month is a categorical variable representing the month of the year, with 12 levels (one for each month)
- β0 is the intercept term
- β1 is the coefficient for the month variable, representing the difference in demand between months.

Figure 4.1.5 Poisson Regression Model

```
# Fit the GLM poisson forecast model

GLM_model = smf.glm(formula='demand ~ C(month)', data=data,
                    family=sm.families.Poisson(link=sm.families.links.log())).fit()
```

44] Python

```
def forecast4(today: date, data: DataFrame) -> int:
    """

        Determine number to open based on Poisson Regression Model with trend components and month variables.
    """
    new_data = {"month": today.month}

    predicted_demand = round(GLM_model.predict(new_data)[0])

    return predicted_demand
```

Figure 4.1.6 Forecast Method 4

The first 3 forecast models are naive models that use a certain range of past data as the forecast value for current data. After we observed a very poor performance with the third forecast model (the model that uses last month's data mean as the forecast value for this month's data), we suspect that it's due to a large month factor that influences the demand. Therefore we developed a Poisson regression model with a month variable as the new forecast model, in the hope that it would improve the performance of the forecast.

4.2. Simulate the loss based on the expectation of today's demand

After we have the ability to generate the expectation of today's demand, we can begin the simulation process. First, we initialise the parameters:

```python
# decision variables
buffer: int = 0                          # number to open more than demand to ensure that demand is always met
forecast_method: int = 1                 # which forecast method to adopt

# tuition variables
tuition_per_student: float = 160000          # how much school would gain from one student increase next
α: float = 0.01                            # decrease in next year student per demand not satified

# Room cost variables
daily_room_cost: float = 200               # how much opening one study room for a day would cost

# Begin the simulation process for a year beginning from the second semester of 2022/2023
start_date = date(2023, 1, 30)
end_date = date(2024, 1, 29)
```

Figure 4.2.1 Parameters initialization

```python
# return a forecast demand for the date
def forecast(date, data) -> int:
    if forecast_method == 1:
        return forecast1(date, data)
    elif forecast_method == 2:
        return forecast2(date, data)
    elif forecast_method == 3:
        return forecast3(date, data)
    elif forecast_method == 4:
        return forecast4(date, data)
    elif forecast_method == 0:
        return simple(date, data)
    else:
        raise ValueError("Invalid forecast method")
```

Figure 4.2.2 Forecast Function

From the start date of 2023/1/30, we loop through the whole year until the end date of 2024/1/29, for each day we generate the following statistics:

The number of study rooms opened = *expectation of demand + buffer*

Real demand for study rooms of the day*: using the generate_data function to generate the past year's data and simulate real-time data.*

Shortage = *max(demand - opened, 0)*

Oversupply = *max(opened - demand, 0)*

Total Tuition loss = *shortage * the impact of unsatisfied demand (α) * tuition loss per student.*

Study room cost = *study room cost per day * the number of study rooms opened*

```
def simulate():
    global data
    np.random.seed(12)
    current_date = start_date
    simulation = []
    total_cost = 0
    simu_data = data.copy()
    while current_date <= end_date:
        real_time_demand = generate_data(current_date)
        predicted = forecast(current_date, simu_data)
        simu_data.loc[pd.to_datetime(current_date), "demand"] = real_time_demand

        number_to_open = predicted + buffer
        shortage = max(real_time_demand-number_to_open, 0)
        over_supply = max(number_to_open-real_time_demand, 0)

        # Loss due to unsatisfied students
        tuition_loss = (shortage * α * tuition_per_student)

        # Study room cost
        study_room_cost = daily_room_cost * number_to_open

        cost = tuition_loss + study_room_cost
        total_cost += cost
```

```
        simulation.append({"date": pd.to_datetime(current_date), "demand": real_time_demand, "predict": predicted,
                           "opened": number_to_open, "cost": cost, "tuition loss": tuition_loss,
                           "study room cost": study_room_cost, "unsatisfied": shortage,
                           "tuition loss": tuition_loss, "study room cost": study_room_cost,
                           "oversupply": over_supply})
        current_date = current_date + timedelta(days=1)

    simulation = DataFrame(simulation)
    simulation.set_index('date', inplace=True)
    mean_daily_cost = total_cost / ((end_date-start_date).days + 1)
    print(f"mean cost per day is {mean_daily_cost:.2f}")
    return simulation
```

Figure 4.2.3 Simulation Function

Finally, we appended all the above statistics for each day into a data frame called “simulation”. Now the “simulation” data frame holds statistics for the whole year, ready for later analysis.

5. Output analysis

5.1. Performance of different forecast methods

We first try out the different forecast models and see their influence on the shortage and supply as well as overall cost.

To compare the result, 4 graphs are used to show the time series of demand and open number and cost that occurred each day, as well as the distribution of shortage and oversupply throughout the year.

```python
# See how different forecast method would affect the cost
forecast_method = 0
simulation = simulate()
total_tui_loss = simulation['tuition loss'].sum()
total_study_room_cost = simulation['study room cost'].sum()
sizes = [total_tui_loss, total_study_room_cost]
labels = ['tuition loss', 'study room cost']
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(8, 12))
simulation[["demand", "opened"]].plot(ax=axes[0, 0])
simulation[["cost", "tuition loss", "study room cost"]].plot(ax=axes[0, 1])
simulation.loc['2023-02-02':'2023-03-02', ["demand", "opened"]].plot(ax=axes[2, 0])
simulation["unsatisfied"].hist(ax=axes[1, 1], bins=20)
simulation["oversupply"].hist(ax=axes[2, 1], bins=20)
axes[1, 1].set_xlabel('shortage')
axes[2, 1].set_xlabel('oversupply')
axes[1, 0].pie(sizes, labels=labels, autopct='%1.1f%%')
plt.show()
print("mean shortage:", round(simulation["unsatisfied"].mean(), 2))
print("mean oversupply:",round(simulation["oversupply"].mean(), 2))
```

✓ 2.1s Python

Figure 5.1.1 Code to compare the result

Simple method:

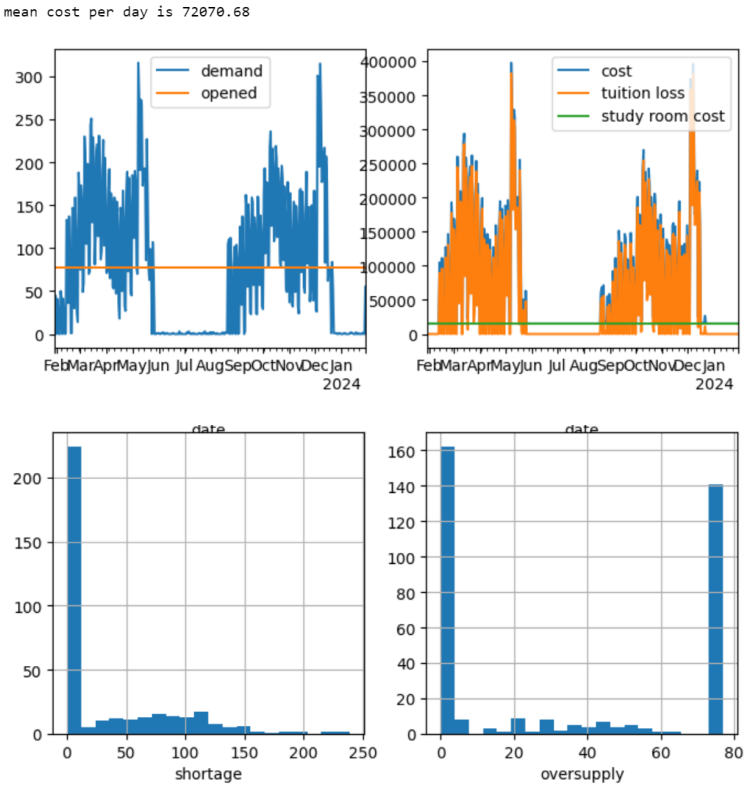
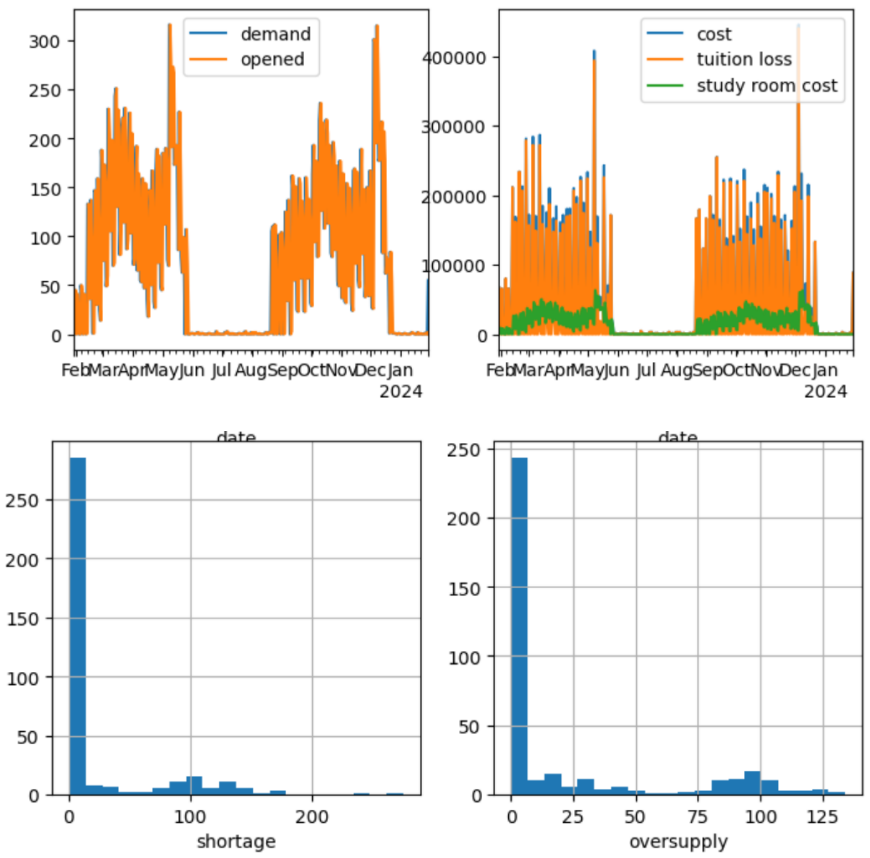


Figure 5.1.2 Result using Simple method

Forecast Method 1:

```
mean cost per day is 50362.74
```

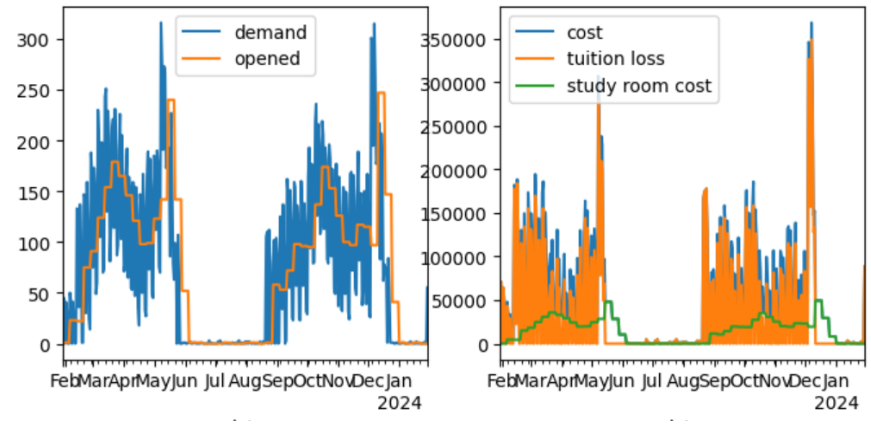


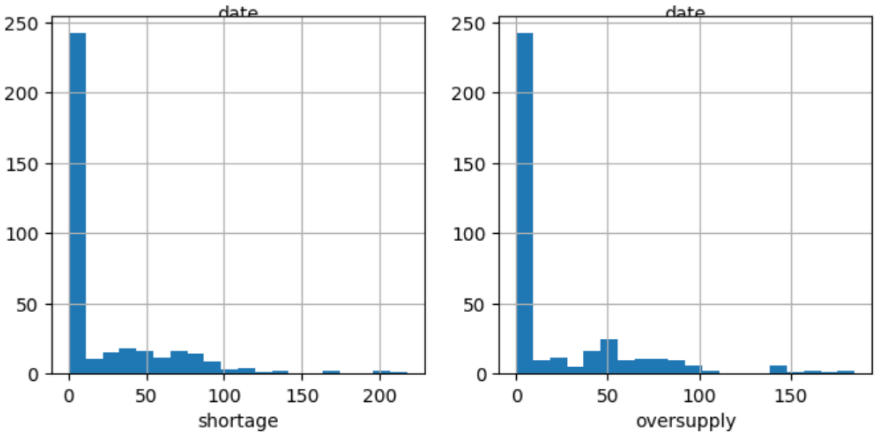

```
mean shortage: 21.82
mean oversupply: 21.67
```

Figure 5.1.3 Result using Forecast Method 1

Forecast Method 2:

```
mean cost per day is 50329.86
```



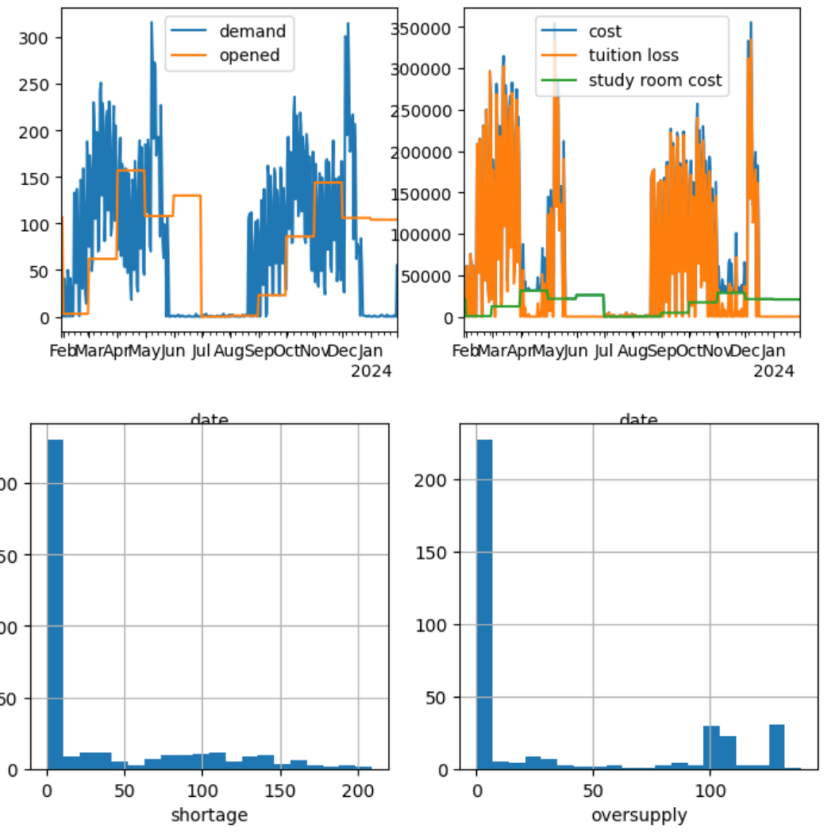



```
mean shortage: 21.8
mean oversupply: 21.66
```

Figure 5.1.4 Result using Forecast Method 2

Forecast Method 3:

```
mean cost per day is 70533.15
```




```
mean shortage: 34.43
mean oversupply: 34.27
```

Figure 5.1.5 Result using Forecast Method 3

Forecast Method 4:

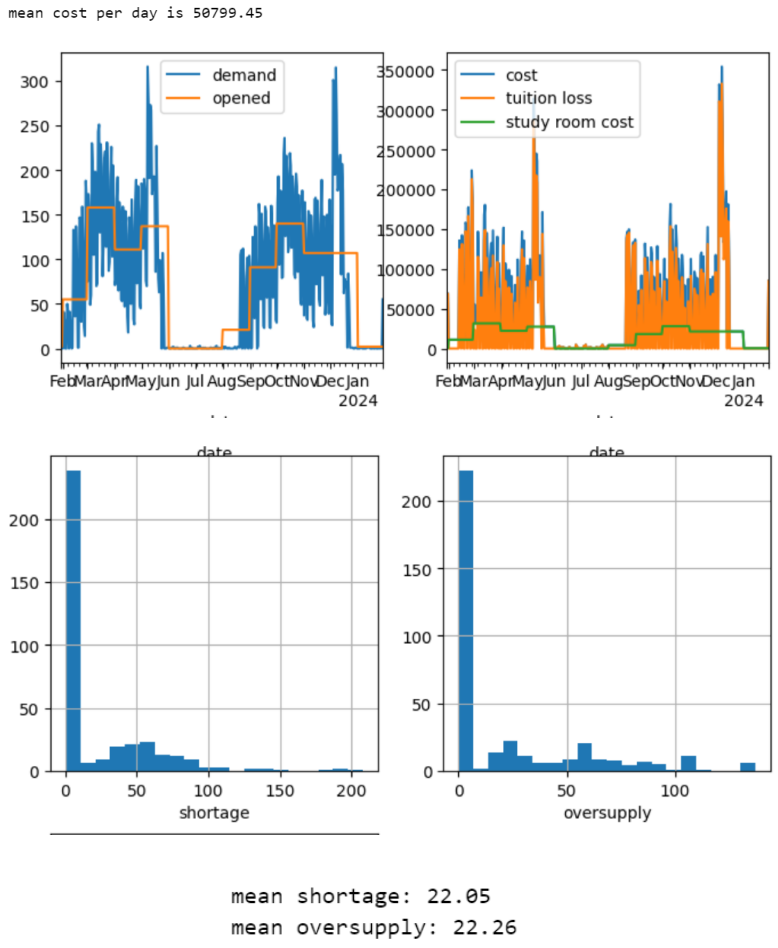


Figure 5.1.7 Result using Forecast Method 4

All forecast models except model 3 have achieved substantially better results than the simple method, with the first forecast model having a better result by a small margin, that is, the model that reacts fastest to the effect of time.

5.2. Explore the effect of buffers

From the observation of the cost time series, we can observe that in almost all situations the tuition loss cost component has a dominant effect over the study room cost. For a clearer dive into the cost structure, we generate pie charts for different situations as illustrated below.

Simple method:

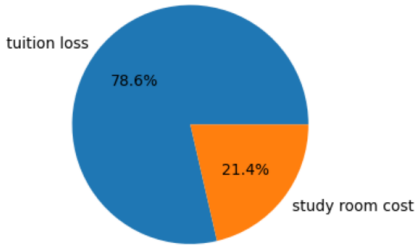


Figure 5.2.1 Cost structure of the Simple method

Forecast method 1:

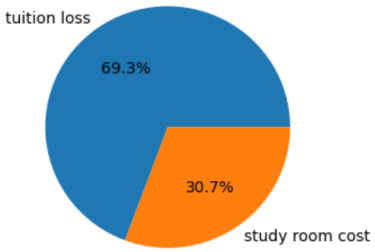


Figure 5.2.2 Cost structure of Forecast method 1

Forecast method 2:

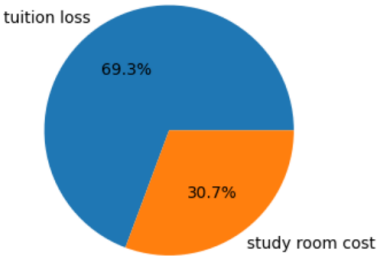


Figure 5.2.3 Cost structure of Forecast method 2

Forecast method 4:

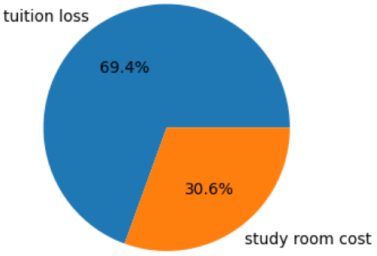


Figure 5.2.4 Cost structure of Forecast method 4

Maybe an even better result can be achieved by reducing shortage (unsatisfied demand) as much as possible at the cost of increasing oversupply, so that the tuition

loss is minimised. So we try to increase the buffer to increase the overall opened study rooms and cover more demand.

After we set the buffer to 20, the yields the below result:

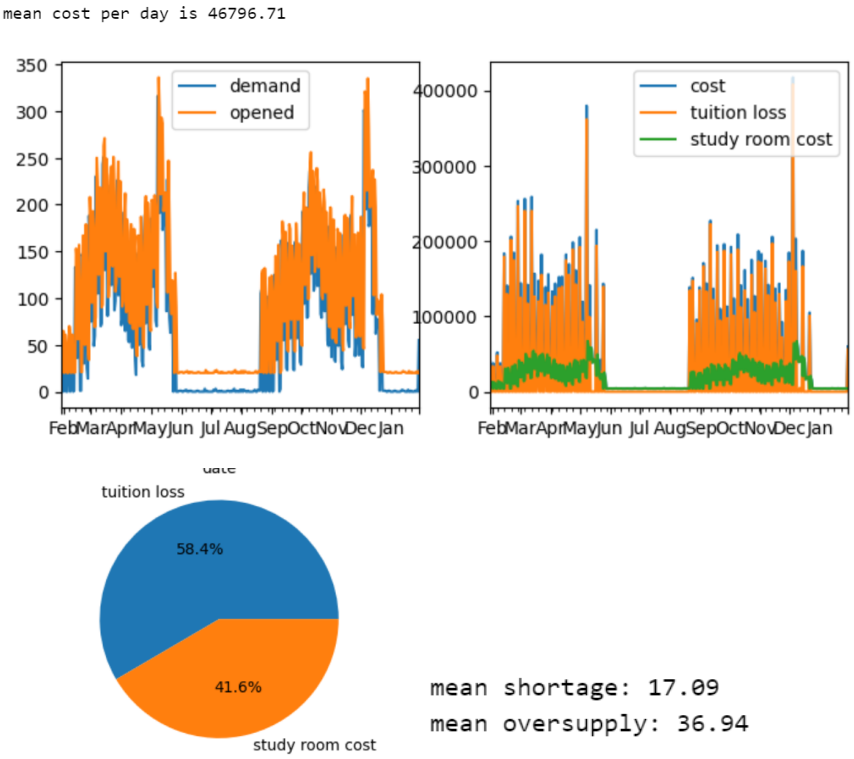
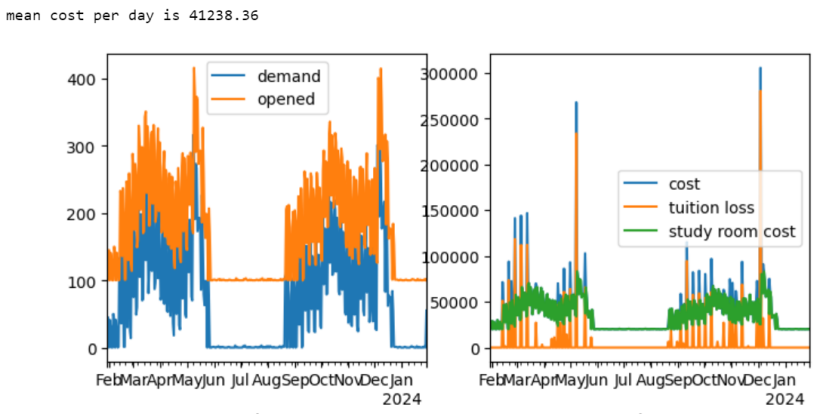


Figure 4.2.5 Result of setting the buffer to 20

We can see the apparent change in the cost proportion of tuition loss and study room cost, and how the mean shortage has decreased at the cost of increasing oversupply. And the daily mean cost has decreased by a significant amount of 4k. We experimented with several even larger buffers in hopes of further decreasing the total cost.

Buffer = 100:



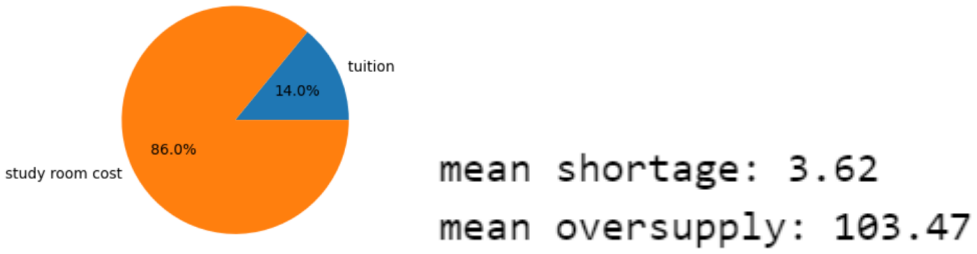


Figure 4.2.6 Result of setting the buffer to 100

Buffer = 200:

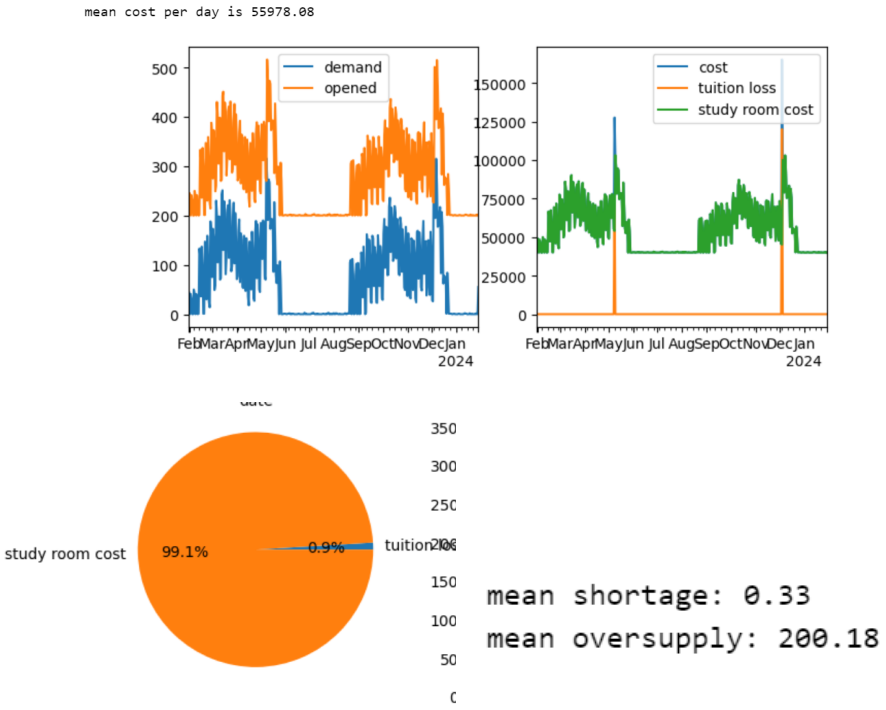


Figure 4.2.7 Result of setting the buffer to 200

According to the simulation result, the cost has further reduced to 41238 after we increased the buffer to 100, but has rebounded to 55978 after we increased the buffer to 200, which suggests that at that point the decrease in the shortage doesn't justify the increase in the oversupply anymore. So the optimal buffer should lie somewhere between 100 and 200.

5.3. Discussion of different cost structure

For the previous settings, we set the cost settings as listed below:

Tuition loss per unsatisfied demand: *(student loss per unsatisfied demand) * (tuition loss per student) = 0.01 * 160000 = 1600*

Daily study room cost: *200 per room*

But due to different university natures, the cost structure may vary largely. For example, a government-funded university in Mainland China that emphasises more on the conditions of study rooms may have the below cost settings:

Tuition loss per unsatisfied demand: *(student loss per unsatisfied demand) * (tuition loss per student) = 0.01 * 32000 = 320*

Daily study room cost: *250 per room*

```
# Explore how different tuition and daily cost per study room affect the process
tuition_per_student = 32000
daily_room_cost = 250
buffer = 0
```

```
mean cost per day is 26292.60
```

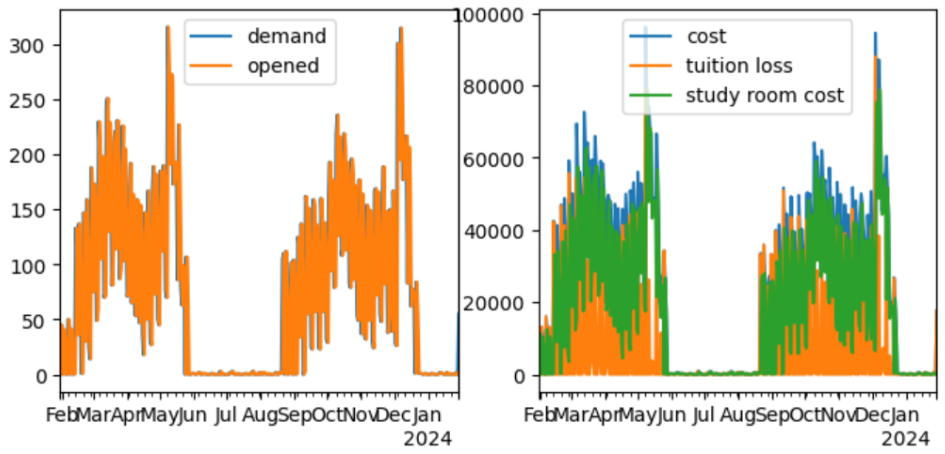


Figure 5.3.1 overview of a different cost setting

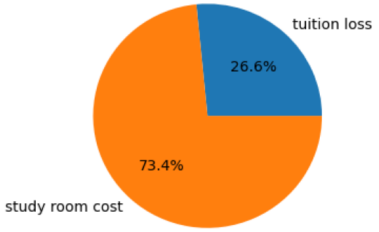


Figure 5.3.2 Cost structure of a different university nature

The mean loss (total loss) caused by the number of opened study rooms has changed significantly due to a high drop in tuition fee and an increase in the study

room cost. The cost structure has reversed to a state where study room cost has the dominant effect here.

Under this circumstance, if we try to increase the buffer, see what will happen:

Buffer = 5:

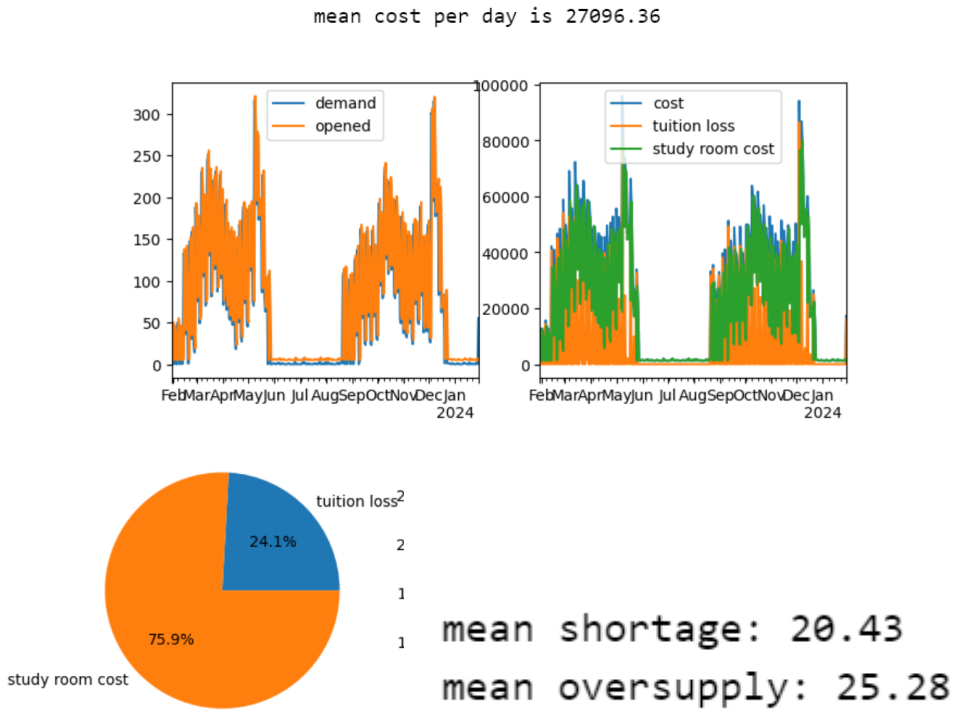


Figure 5.3.3 simulation result when buffer = 5

The total loss has even increased for a slight increase in the buffer. In this cost structure, the oversupply weighs more heavily in the total loss than shortage. Increasing oversupply to decrease shortage is not a wise decision here.

So, how about we decrease the buffer?

Buffer = -20:

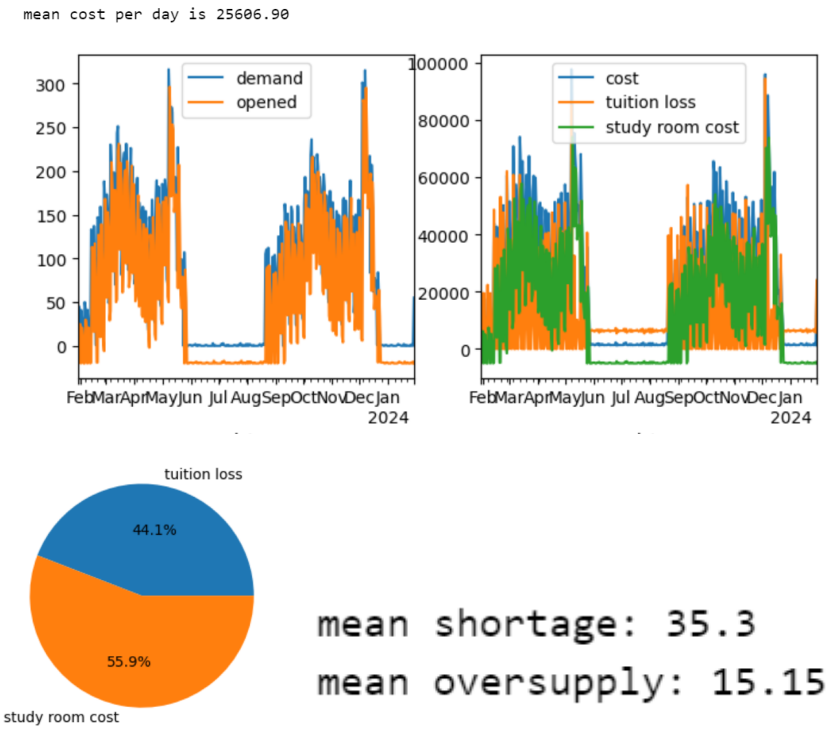


Figure 5.3.4 simulation result when buffer = -20

This way we are trying to decrease oversupply at the cost of increasing shortage. The total loss has indeed decreased. Can we achieve a better result by further decreasing the buffer?

Buffer = -40:

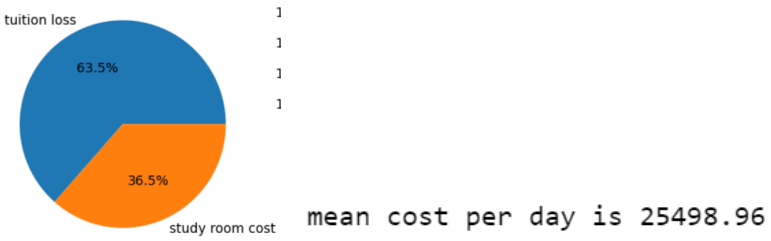


Figure 5.3.5 simulation result when buffer = -40

Buffer = -60:

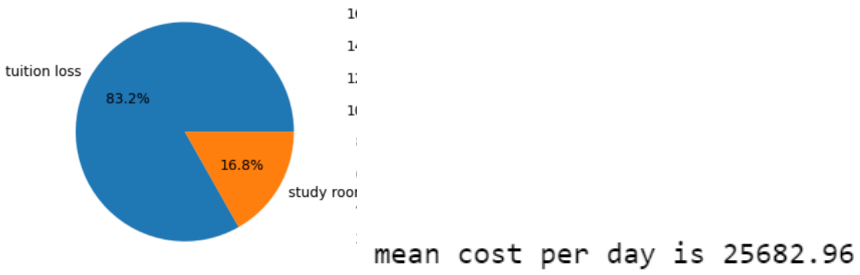


Figure 5.3.5 simulation result when buffer = 60

From the above results we can see the total loss starts to increase when we set the buffer to 60. So the optimal value of the buffer is somewhere between 40 and 60.

Discussions

Universities used the model to design the best policy in order to increase profit. The system finds out the demand on a specific date with the simulation and generates the model. The demand changes may affect the number of study rooms to be open. The cost can then be controlled more easily.

There are many variables. For example, variables like buffer, forecast method, daily room cost, and tuition per student. They may have an impact on the simulation and cause many changes in the result.

Our target is to simulate results and find out the best policy for the university in order to have many profits. By using different variables and generating different results from them, then the university can have the data they want.