Micron NUS-ISE Business Analytics Case Competition 2025

ArrivalEvent

Contents

Question 1

Part a)

In designing the wafer loading profile for Micron's production strategy in 2026-2027, given that no explicit optimization objective function was given, we set our objective to **minimize the total number of wafers** required per quarter while ensuring that production meets the Total Addressable Market (TAM) demand.This is achieved by leveraging the most efficient nodes available in each quarter. Through building a linear optimization model, which integrates the constraints provided in question 1(a), the model aims to minimize production costs and optimize resource utilization through the above stated objective function.

```
# Define the optimization model
model = pulp.LpProblem("Wafer_Loading_Minimization", pulp.LpMinimize)

# Decision variables: Wafer starts per week for each node in each quarter
x = pulp.LpVariable.dicts("wafer_start", [(n, q) for n in gb_per_wafer for q in quarters], lowBound=0, cat='Integer')

# Objective function: Minimize the total number of wafers across all quarters
model += pulp.lpSum(x[n, q] for n in gb_per_wafer for q in quarters)
```

Since different nodes offer varying storage capacities per wafer and yield rates with the progression of time, our approach focuses on:

1. Maximizing the use of high-capacity nodes (Node 3 > Node 2 > Node 1) as their yield improves to reduce wafer consumption
2. Minimizing reliance on low-yield nodes in the early quarters to avoid wastage.

Result Analysis

Q1'26 - Q2'26:

- Node 1 remains the primary source due to its high 98% yield and stable output. However, its low GB per wafer (100K GB) means its efficiency is limited.
- Node 2 receives moderate increases in wafer allocation as its yield improves significantly from 60% to 82%.
- Node 3 remains minimally allocated due to its low 20% yield in Q1'26, making it inefficient in the short term.

Q3'26 - Q4'26:

- Node 2 is prioritized as its yield reaches 95-98%, making it nearly as reliable as Node 1 but with a much higher GB per wafer (150K GB).
- Node 3 allocation grows steadily, as its yield improves from 35% to 50%, making it increasingly viable.
- Node 1 continues to be reduced, as it has the lowest efficiency.

Q1'27 - Q2'27:

- Node 3 becomes the dominant node, as its yield rises from 65% to 85%, making it far more efficient than Nodes 1 and 2.

- Node 2 usage remains high due to its stable 98% yield.
- Node 1 is phased out to minimize costs, given its low GB per wafer.

Q3’27 - Q4’27:

- Node 3 is now the dominant supplier, with its yield reaching 95-98%, allowing a reduction in overall wafer count while meeting TAM requirements.
- Node 2 is still used moderately, but Node 1 is completely phased out.

Part b)

(i) Determining Tool Requirements

- We took the weekly wafer production numbers for each node (from Part A) and converted them into quarterly totals (by multiplying by 13 weeks).
- For each workstation, we then calculated the total processing time needed. For example, if a node is processed at a workstation, its contribution to processing time is its weekly wafer count × the number of weeks × the “minute load” (i.e., the number of minutes a wafer requires at that workstation).
- Since not every node goes through every workstation (e.g., Node 3 only uses workstations A, B, C, F, G, J), we only sum the contributions from the nodes that actually use that workstation.
- Next, we compared the total processing time needed to the available processing time per tool at that workstation. The available time per tool is calculated as 13 x 10080 minutes/week x the workstation’s utilization rate. Dividing the total processing time by the available time gives us the tool requirement.

```
# Loop over each quarter:
for q_idx, q in enumerate(quarters):
    # Revenue Calculation
    total_GB_output = 0.0
    for node in gb_per_wafer:
        # Get the weekly wafer start from part 1a solution (x dictionary)
        wafer_weekly = pulp.value(x[(node, q)])
        wafers_quarter = wafer_weekly * weeks_per_quarter
        node_output = wafers_quarter * gb_per_wafer[node] * yield_rates[node][q_idx]
        total_GB_output += node_output
    revenue = total_GB_output * contribution_margin
    revenue_q[q] = revenue

    # Tool Requirement and CAPEX Calculation
    CAPEX_q = 0.0
    for w in workstations:
        total_minutes = 0.0
        # Sum processing minutes only for nodes that actually use workstation w.
        for node in gb_per_wafer:
            if w in minute_load.get(node, {}):
                wafer_weekly = pulp.value(x[(node, q)])
                total_minutes += wafer_weekly * minute_load[node][w] * weeks_per_quarter

        available_time = weeks_per_quarter * minutes_per_week * utilization[w]
        req_tools = math.ceil(total_minutes / available_time) if available_time > 0 else 0
        tool_required[(w, q)] = req_tools
```

Note:

- **This step determines the total tool requirement for each workstation in every quarter**
- **Ensures each workstation has sufficient tools to process the required wafers**

(ii) Calculating CAPEX

- For each quarter, we then determined how many additional tools must be purchased (on top of the initial tools provided). In subsequent quarters, extra tools are purchased only if the tool requirement exceeds the cumulative (already available) capacity from previous quarters.
- The cost of extra tools is computed by multiplying the number of additional tools by the CAPEX per tool for that workstation. Summing these across all workstations gives the total CAPEX for the quarter.

```
if q_idx == 0:
    # For Q1, compare against the free initial tool count
    additional_tools = max(0, req_tools - initial_tools[w])
    previous_req[w] = initial_tools[w] + additional_tools  # Should equal req_tools if req_tools > initial
else:
    # For subsequent quarters, purchase additional tools only if the requirement increases
    additional_tools = max(0, req_tools - previous_req[w])
    previous_req[w] = max(previous_req[w], req_tools)
```

Note:

- **This step ensures that the minimum necessary tools are purchased each quarter**
- **Determines how much CAPEX is incurred due to new tool purchases**

(iii) Net Profit Calculation

- Revenue is calculated from the production: for each node, the quarterly output in GB is obtained by multiplying the number of wafers by the GB per wafer and the yield; then the total is multiplied by a fixed contribution margin.
- Net profit for each quarter is then the revenue minus the CAPEX incurred (the extra tool purchases).

(iv) Evaluating the Loading Profile

- With the tool requirement and CAPEX figures in hand, we can see the financial outcomes per quarter. For instance, in Q1'26 no extra tools are purchased (hence CAPEX is $0) and profit equals revenue. In later quarters, if extra tools must be bought, the CAPEX reduces net profit, even turning profit negative in some quarters.
- This analysis allows us to judge if the current loading profile from Part A (which minimizes wafer production) is truly optimal for profit. If the production mix forces

the purchase of many expensive extra tools, then it might be better to adjust the loading profile. For example, by shifting production toward nodes that have higher yield or require less processing time, we could reduce CAPEX and further improve net profit.

```
# CAPEX: Sum over quarters and workstations: (CAPEX cost per extra tool * extra tools purchased)
total_CAPEX = pulp.lpSum(CAPEX_per_tool[w] * Y[(w, q)] for w in workstations for q in quarters)

model += total_revenue - total_CAPEX, "Total_Profit"
```

Note:

- This step evaluates if the current loading plan from Part A leads to optimal profitability
- Helps determine whether a different loading strategy could reduce CAPEX and improve net profit

  Thus, we modify the model from part 1(a), changing our objective function from minimizing the number of wafers to maximizing profit.

Part c)

Our chosen loading profile strikes a balance between meeting the forecasted TAM and minimizing CAPEX from extra tool purchases. It adheres to production constraints by ensuring quarterly outputs stay within ±2 billion GB of forecasted demand while limiting abrupt changes (±2,500 wafers/week) between quarters. A key advantage is that the integrated model capitalizes on existing tool capacity: it utilizes the free machines available in Q1 and only purchases additional tools when production demands exceed current capacity. This minimizes unnecessary CAPEX, ensuring that extra expenditures are incurred only when essential for meeting market demand. Overall, the integrated approach results in a loading profile that not only meets forecasted demand but also optimizes net profit.

Question 2

Part a)

(i) Population mean and standard deviation

Given that the RPT samples are representative of the actual RPT distribution, we are able to obtain the population mean and standard deviation by using the pandas module in Python.

```
# Compute Population Mean and Standard Deviation
rpt_stats = df.describe().loc[["mean", "std"]]
```

(ii) Expected RPT

Expected value by definition equals to mean. Additionally, the law of large numbers ensures that the sample mean of a large sample size converges to the true mean distribution. Therefore, the expected RPT follows that of the mean in (ai).

(iii) Expected Tool Requirement

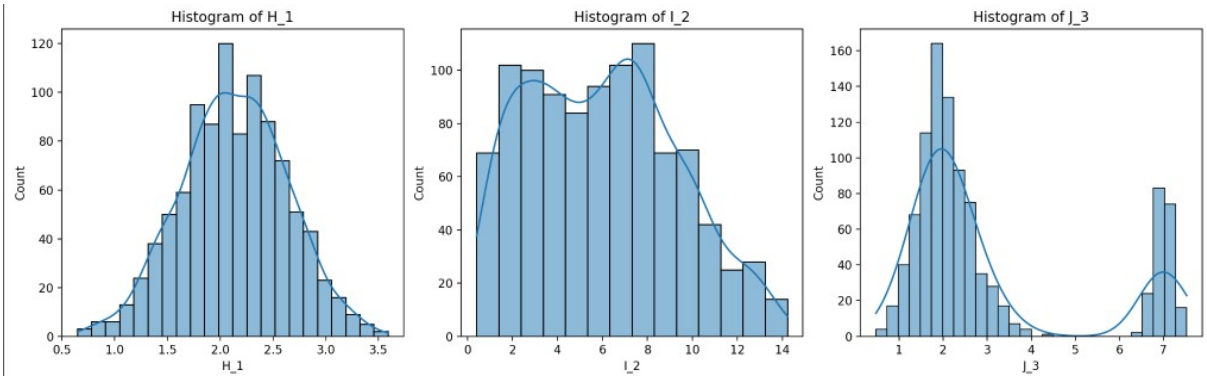
Since Workstations H, I and J only produce one node each, their individual minute loads are equal to the RPT of that node. Plugging the expected RPT from (ii) into the formula, we have the respective expected tool requirements.

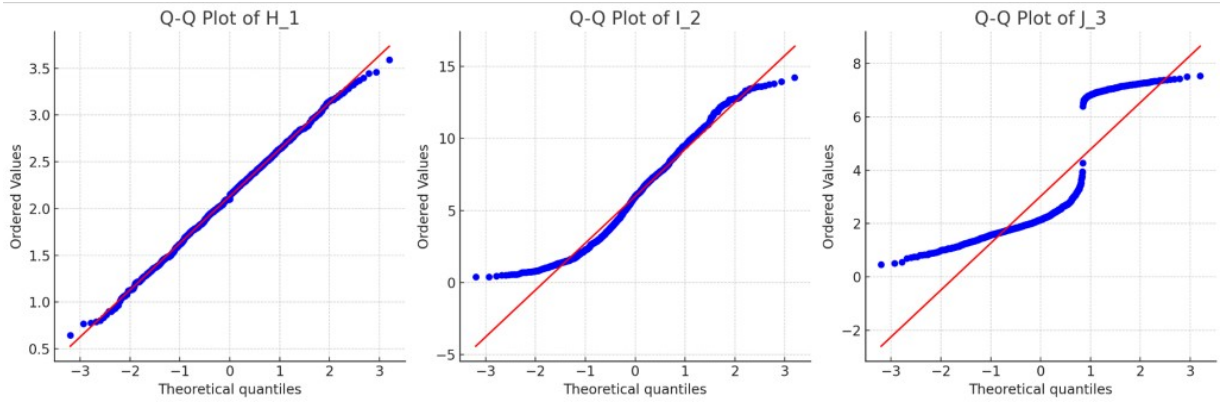
```
# Compute Expected Tool Requirement using Expected RPT
loading = {"H": 12000, "I": 5000, "J": 1000}  # Wafer loading for Q1'26
utilization = {"H": 0.85, "I": 0.75, "J": 0.60}  # Utilization rate per workstation
total_time_per_week = 7 * 24 * 60  # Total available tool time per week

tool_requirements = {
    workstation: (loading[workstation] * expected_rpt_values[workstation]) / (total_time_per_week * utilization[workstation])
    for workstation in loading
}
```

(iv) Total Processing Time

In order to calculate the total processing time, we first do a normality test (e.g. Shapiro–Wilk test) or use graphical methods (e.g. histogram, QQ plot) to determine the distribution:






```
Shapiro-Wilk Test Results:
H: p-value = 0.6334 (Not significant, data is likely normal)
I: p-value = 4.51e-13 (Significant, data is not normal)
J: p-value = 1.53e-37 (Significant, data is not normal)
```

If the data is normally distributed (e.g. H), we do a Monte Carlo simulation with 10000 iterations to estimate the distribution of the random variable (each wafer's production time).

If it is not (e.g. I, J), we use Bootstrap Estimation, taking 10000 random samples to estimate the distribution.

```python
# Hybrid Simulation Method
column_mapping = {"H": "H_1", "I": "I_2", "J": "J_3"}
num_simulations = 10000

simulated_times = {}
for workstation in loading:
    if workstation == "H":  # Monte Carlo Simulation for Normal Distribution
        simulated_times[workstation] = np.sum(np.random.normal(
            expected_rpt_values[workstation],
            rpt_stats.loc["Population Std Dev", column_mapping[workstation]],
            loading[workstation]
        ))
    else:  # Bootstrap Sampling for Non-Normal Distributions
        sampled_rpt = np.random.choice(df[column_mapping[workstation]], size=loading[workstation], replace=True)
        simulated_times[workstation] = np.sum(sampled_rpt)
```

This hybrid approach ensures that we are using dynamic methods to estimate total processing time. Different populations can use this methodology as we do not have a fixed assumption on the data's distribution, and use different methods based on initial analysis.

(v) Probability of Exceeding Time

Total available tool time is a constant, with the shown formula. For total processing time, H is normally distributed, while I and J are approximately normal due to the Central Limit Theorem. Thus, we conduct an upper tail test:

```
# Mean and SD of normal and aprroximately normal distributions
# (calculated from (iv))
H_mean, H_std = 25579.49, 55
I_mean, I_std = 29590.95, 235
J_mean, J_std = 3024.91, 65

# Total available tool time per week:
# mins in week * tool count * utilization
H_tool_time = 7 * 24 * 60 * 3 * 0.85
I_tool_time = 7 * 24 * 60 * 4 * 0.75
J_tool_time = 7 * 24 * 60 * 1 * 0.6

# Compute upper-tail probability (P(X > available tool time))
H_prob = 1 - norm.cdf(H_tool_time, loc=H_mean, scale=H_std)
I_prob = 1 - norm.cdf(I_tool_time, loc=I_mean, scale=I_std)
J_prob = 1 - norm.cdf(J_tool_time, loc=J_mean, scale=J_std)
```

(vi) Probability of at least one failure in a quarter

P(at least one failure in quarter) = $1 - (1 - P_{\text{Exceedance in a week}})^{13}$
Node 1 (H): P(failure in a quarter)=$1 - (1 - 0.0115)^{13}$ (13.96%)
Node 2 (I): P(failure in a quarter) = $1 - (1 - 0.00285)^{13}$ (3.64%)
Node 3 (J): (0%)
Overall 17.09% of 1 week failing over 1 quarter

(vii) Probability of at least one quarter failing

P(at least one failure)=$1 - (1 - 0.1709)^{8}$ (77.68%)
Overall 77.68% of 1 week failing over 8 quarters.

Part b)

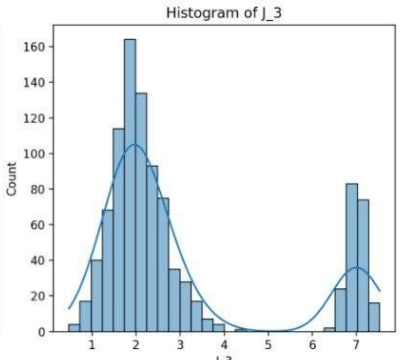
The semiconductor industry is highly unpredictable due to several factors, including fluctuations in temperature, variations in raw material quality, machine precision, and supply chain disruptions. These factors introduce non-uniform production rates and extreme variability, making it difficult to rely on a single-point estimate like the median. Temperature variations, for example, can cause micro-level defects, impacting the reliability of semiconductor yields, while

equipment calibration inconsistencies can lead to performance deviations across production batches. Given these unpredictable factors, a robust statistical approach is necessary to capture variability rather than ignore it.

Using bootstrapping and Monte Carlo simulation provides a more comprehensive understanding of risk and performance trends. Bootstrapping allows for resampling of data with replacement, generating multiple simulated datasets to estimate the true distribution of RPT (reliability performance time) under real-world uncertainty. Unlike the median, which only provides a static value at the 50th percentile, bootstrapping constructs confidence intervals, showing a range of probable outcomes rather than a single deterministic number.

Monte Carlo simulation further enhances reliability prediction by running thousands of randomized scenarios, incorporating actual production variability into predictive models. Instead of assuming a fixed RPT value, Monte Carlo allows probabilistic forecasting, simulating how different operational conditions (e.g., fluctuating machine precision, varying cooling times) affect output. This method is particularly valuable in semiconductor manufacturing, where small process deviations can lead to exponential reliability shifts.

Compared to using the median, which disregards outliers and systemic risks, bootstrapping and Monte Carlo simulation offer a dynamic and adaptive model for decision-making. These methods allow manufacturers to improve risk mitigation and production planning in an industry where precision and variability management are critical.



Histogram shows the presence of outliers and how they can cause fluctuation, such as in the case of J. Solely using the median will not be able to account for them.

Part c)

Yes, profitability will be negatively impacted due to:

- Missed wafer output: A 77.68% chance of at least one quarter failing means Micron will likely fall short of expected production, reducing GB output and revenue
- Potential penalties and lost sales: If Micron has commitments to customers, failing to meet output targets can lead to contractual penalties or lost market share

- Increased CAPEX or OPEX: To mitigate risk, additional tools may be required, increasing capital expenditure (CAPEX). Alternatively, extending production hours (e.g., increasing utilization) raises operational expenses (OPEX)

A 77.68% failure probability is too high, given the capital-intensive nature of semiconductor manufacturing. A more reasonable risk threshold would be below 5-10%

To lower the risk below the acceptable threshold (5-10%), Micron should modify the tool purchase strategy, loading distribution, or utilisation rates:

1. Increase Tool Count in Workstations H and I:
    - Node 1 (H) has 13.96% failure per quarter, meaning it is under-capacity.
    - Node 2 (I) has 3.64% failure per quarter, which is lower but still contributes to risk.
    - Node 3 (J) has 0% failure, so no changes needed.
    - Adding 1-2 extra tools to Workstation H and possibly I can lower the probability of exceeding tool time.

2. Modify Weekly Loading Strategy
    - Instead of keeping a fixed weekly loading rate, Micron could adjust loading dynamically based on RPT distributions.
    - This requires real-time monitoring of RPT trends and adjusting weekly wafer starts accordingly.

<u>Conclusion</u>

Micron should use a variable RPT, possibly following our Monte Carlo/ Bootstrapping analysis to allow for **dynamic RPT distributions**. This would modify operation strategies to allow for more tools for workstations H and I. Hence, the **total instances of processing time > total available tool time** is lower and thus the **rate of failure decreases**.