**Setup**

- Candidate must have either (1) a Gmail account or (2) a local python environment.

    1. Gmail
        - Ensure that Google Colaboratory is installed in your Google Account.
            - Within Google Drive, click “+ New” → “More” → Google Colaboratory
            - If you haven’t enabled Google Colaboratory, click “Connect more apps” and do so.
        - Open and run the colaboratory notebook in the Google Drive shared with you.

    2. Local
        - Create a new Python file in your preferred text editor.
        - In that file, add the following code.
        - When your interview is completed, upload the file to the Google Drive shared with you.

```python
import pandas as pd
import numpy as np
import random

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Define node names (10 node operators)
nodes = [f'node_{i}' for i in range(1, 11)]

# Create node-specific probabilities for each action.
# Adjust the ranges as needed.
node_probabilities = {}
for node in nodes:
    node_probabilities[node] = {
        'start_end': np.random.uniform(0.9, 1.0),
        'get_data': np.random.uniform(0.9, 1.0),
        'communicate': np.random.uniform(0.9, 1.0),
        'make_report': np.random.uniform(0.9, 1.0),
        'transact': np.random.uniform(0.95, 1.0)
    }

# Simulation parameters
```

```
total_rounds = 10000
rounds_per_epoch = 6

results = []

# Simulate each round
for global_round in range(1, total_rounds + 1):
    # Determine epoch and round number within the epoch
    epoch = (global_round - 1) // rounds_per_epoch + 1
    round_in_epoch = (global_round - 1) % rounds_per_epoch + 1

    # Randomly select a leader for this round
    leader = random.choice(nodes)

    # Simulate the leader's Start/End action BEFORE attempting any transactions.
    # This result will determine if any transaction attempts should occur.
    leader_start_end = 1 if np.random.rand() <
node_probabilities[leader]['start_end'] else 0

    # If the leader's Start/End fails, then we do NOT attempt any transaction.
    if leader_start_end == 1:
        # Since any node can transmit, we consider all nodes.
        candidates = nodes.copy()
        random.shuffle(candidates)
        transact_results = {}
        # Attempt candidates sequentially.
        for candidate in candidates:
            # Simulate the candidate's transaction attempt.
            if np.random.rand() < node_probabilities[candidate]['transact']:
                transact_results[candidate] = 1  # success
                break  # Once one candidate succeeds, stop further attempts.
            else:
                transact_results[candidate] = 0  # attempted but failed
        # For nodes not attempted, they will be left out (and get NA later).
    else:
        # Leader failed Start/End, so no transaction attempt is made.
        transact_results = {}

    # Now simulate actions for each node in this round.
    for node in nodes:
        # For Start/End: only the leader performs this action.
        # Non-leaders receive NA (np.nan).
        if node == leader:
            start_end = leader_start_end
        else:
```

```
            start_end = np.nan

        # All nodes perform these actions using their node-specific probabilities.
        get_data    = 1 if np.random.rand() < node_probabilities[node]['get_data']
else 0
        communicate = 1 if np.random.rand() <
node_probabilities[node]['communicate'] else 0
        make_report = 1 if np.random.rand() <
node_probabilities[node]['make_report'] else 0

        # For Transact:
        # - If a node was attempted, record its attempt result (0 or 1).
        # - If not attempted (or if leader_start_end failed), record NA.
        transact = transact_results[node] if node in transact_results else np.nan

        # Save the record for this node in this round.
        results.append({
            'epoch': epoch,
            'round_in_epoch': round_in_epoch,
            'global_round': global_round,
            'leader': leader,
            'node': node,
            'start_end': start_end,
            'get_data': get_data,
            'communicate': communicate,
            'make_report': make_report,
            'transact': transact
        })

# Create a DataFrame from the results
df = pd.DataFrame(results)
rounds = df.loc[:, ['epoch', 'round_in_epoch', 'global_round',
'leader']].drop_duplicates()
start_end = df.loc[:, ['epoch', 'round_in_epoch', 'global_round', 'node',
'start_end']].drop_duplicates()
get_data = df.loc[:, ['epoch', 'round_in_epoch', 'global_round', 'node',
'get_data']].drop_duplicates()
communicate = df.loc[:, ['epoch', 'round_in_epoch', 'global_round', 'node',
'communicate']].drop_duplicates()
make_report = df.loc[:, ['epoch', 'round_in_epoch', 'global_round', 'node',
'make_report']].drop_duplicates()
transact = df.loc[:, ['epoch', 'round_in_epoch', 'global_round', 'node',
'transact']].drop_duplicates()
```

**Interview**

Your task is to determine the performance of nodes in a network. This network runs a protocol. This protocol iterates in sequence through “epochs” and “rounds.” In each epoch, there are 6 rounds. For example, consider the following sequence of (epoch, round): (8000,4), (8000,5), (8000,6), (8001,1), (8001,2), etc. A leader is selected to administer all rounds in each epoch. In each round, the leader is expected to successfully start and end the protocol, and each node in the network is expected to procure data, communicate with its peers, make reports to the leader, and occasionally do blockchain transactions. Whether each node successfully did each of these tasks in each round can be embodied via boolean. For example, consider the following panel data and notes.

| SAMPLE DATA | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Epoch** | **Round** | **Protocol** | **Leader** | **Transactor** | **Node** | **Start/End** | **Get Data** | **Communicate** | **Make Report** | **Transact** |
| 8000 | 6 | X | A | 0 | A | 1 | 1 | 1 | 1 | 0 |
| 8000 | 6 | X | A | 0 | B | 0 | 1 | 1 | 1 | 0 |
| 8000 | 6 | X | A | 0 | C | 0 | 1 | 1 | 0 | 0 |
| 8000 | 6 | X | A | 0 | D | 0 | 1 | 0 | 1 | 0 |
| 8000 | 6 | X | A | 0 | E | 0 | 0 | 1 | 1 | 0 |
| 8000 | 6 | X | A | 1 | F | 0 | 1 | 1 | 1 | 1 |
| 8001 | 1 | X | B | 0 | A | 0 | 1 | 0 | 1 | 0 |
| 8001 | 1 | X | B | 0 | B | 1 | 1 | 1 | 1 | 0 |
| 8001 | 1 | X | B | 1 | C | 0 | 0 | 1 | 1 | 0 |
| 8001 | 1 | X | B | 0 | D | 0 | 1 | 1 | 0 | 0 |
| 8001 | 1 | X | B | 1 | E | 0 | 1 | 1 | 1 | 1 |
| 8001 | 1 | X | B | 0 | F | 0 | 1 | 1 | 1 | 0 |

- First Round (8000,6)
  - Only the leader (Node A) was asked to Start/End the round, and did so successfully.
  - Only one node (Node F) was asked to transact, and did so successfully.
  - Node C procured data and communicated with its peers, but didn’t report.
  - Node E communicated with its peers but contributed an empty report.
- Second Round (8001,1)
  - Only the leader (Node B) was asked to Start/End the round, and did so successfully.
  - Two nodes (Node C and E) were asked to transact, but only one did so.
  - Node D procured data and communicated with its peers, but didn’t report.
  - Node C communicated with its peers but contributed an empty report.
- Transact will only populate if Start/End is successful

- Assume that Get Data, Communicate, and Make Report are independent
- Rounds are successful when Transact is successful

Propose a method by which to convert these data into performance scores for each node. Discuss the approach with your interviewer. Be wary of false assumptions and statistical pitfalls. Consider potential attack vectors and logical means of creating additional data. Discuss the strengths and weaknesses of your approach. The data directory will contain 6 dataframes with the relevant data:

- rounds
- communicate
- get_data
- make_report
- start_end
- transact