Cascade Dynamics on Random Networks

1. Introduction:

This project explores the dynamics of threshold-based cascades on random networks, with the aim of replicating results from Duncan J. Watts' seminal paper on global cascades. The implemented model simulates how behaviors, innovations, or information spread in a social network when agents adopt if the fraction of adopted neighbors exceeds a threshold φ. The focus is on reproducing cascade size distributions (Figure 3) and cascade windows as a function of threshold φ and mean degree z (Figure 4).

2. Model Description:

The model consists of agents represented as nodes in a network. Each agent has two states: adopted or not adopted. Adoption follows a threshold rule: an agent adopts when the fraction of its neighbors that have already adopted is greater than or equal to φ. The network structure is generated using Erdős–Rényi random graphs with mean degree z. The cascade unfolds starting from a small set of initial adopters, propagating iteratively until no further adoptions occur.

3. Code Structure:

The implementation is organized into several key components:

- Agent class: represents individual nodes with an ID and adoption status.
- Cascade Model class: manages the propagation process using a queue-based breadth-first approach.
- Network utilities: functions to generate random networks and compute cumulative distributions.
- Experiment utilities: functions to run multiple trials of cascades to estimate distributions.
- Plotting routines: generate cascade size distributions and cascade window plots with Matplotlib.
- Markdown explanations: provide background, objectives, and context within the notebook

4. Strengths of the Implementation:

- Conceptually correct separation of responsibilities between agents, cascade logic, and experiments.
- Efficient use of a queue to propagate cascades instead of repeatedly sweeping the network.
- Numerical safeguards (handling degree-zero nodes to avoid division by zero).
- Use of LaTeX in plots for mathematical clarity.
- Clear documentation within the notebook using Markdown text.

5. Issues Identified:

- Several code cells in the notebook are truncated, containing literal '...' characters. This prevents complete execution and must be fixed.
- Randomness in network generation and seeding of adopters is not controlled by fixed seeds, leading to irreproducibility across runs.
- Performance issues: the implementation regenerates networks and queries neighbors inefficiently for large-scale experiments (e.g., N=1000, runs=500).
- Lack of type hints and docstrings reduces maintainability and clarity.
- Plotting functions are not encapsulated, leading to duplicated code and reduced readability.
- Validation and testing are missing, so correctness for edge cases is not systematically ensured.

6. Recommendations:

- Fix truncated code cells by re-saving the notebook and ensuring no content is cut off.
- Set random seeds for NumPy, Python's random, and NetworkX to ensure reproducibility of experiments.
- Optimize performance by precomputing adjacency lists and degrees and separating cascade logic into a lightweight function.
- Refactor plotting into reusable functions plot cascade distribution and plot cascade window.
- Add concise docstrings and type annotations for clarity and IDE support.
- Include unit tests for simple cases: $\varphi=0$ (full cascade), $\varphi=1$ (only initial adopters), small line graphs, etc.
- Improve plot presentation by standardizing color schemes, legend placement, and axis scaling.

7. Validation and Testing:

To validate the model, it is recommended to perform tests on special cases:
- For $\varphi=0$, all nodes should eventually adopt (except isolated ones).
- For $\varphi=1$, only the initially seeded adopters should adopt.
- On structured graphs (e.g., a line or star), the cascade behavior should match analytical expectations.

Regression tests with fixed seeds can ensure stability of results over time.

8. Conclusion:

The project successfully implements threshold cascades on random networks, reproducing key qualitative results from Watts' model. While the conceptual design is strong, issues of code truncation, reproducibility, and efficiency need to be addressed. By incorporating the suggested improvements, the notebook can be transformed into a robust and reusable tool for research and teaching, with improved clarity, speed, and reliability.