Economic Competition Simulation and Analysis adversell

Parsa Haghighatgoo
EC Homework Project
Shiraz University

November 7, 2025

Abstract

This report presents an analysis of a simulated duopoly market where two sellers compete by adjusting prices and advertisement levels. Using Python and data visualization tools such as Matplotlib and Seaborn, we model, compute, and interpret the equilibrium outcomes of this competitive environment. The study explores how pricing and advertising decisions influence profits and market dynamics.

Contents

1 Introduction and Theoretical Background

This project investigates the behavior of two competing firms (Seller A and Seller B) in a duopoly market. Each seller chooses a price and an advertisement level to maximize its profit, considering the other seller's strategy. The model is designed to explore:

- The interdependence between price and advertising decisions.
- The concept of Nash equilibrium in a non-cooperative setting.
- How competition affects profit and equilibrium outcomes.

1.1 Concepts Involved

1. **Game Theory:** Each seller acts strategically based on the competitor's actions.
2. **Equilibrium Analysis:** Finding price and ad combinations where neither seller can improve profit unilaterally.
3. **Profit Function:** Profit is computed based on demand, cost, and advertising efficiency.
4. **Visualization:** Using plots to represent equilibrium surfaces and comparative dynamics.

2 Methodology and Code Explanation

In this section, we describe the data generation, model formulation, and computational approach used in the project. Each subsection corresponds to a major part of the Python implementation, with explanations of logic and purpose.

2.1 Data Loading and Preprocessing

Python code:

```
# A) Load & combine both sheets into one DataFrame
import pandas as pd
import numpy as np

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

file_path = "dataset/online_retail_II.xlsx"  # adjust if needed

# Read all sheets at once (returns a dict: {sheet_name: df})
sheets = pd.read_excel(file_path, sheet_name=None)

# Add a column to remember which sheet each row came from, then stack
df_list = []
for name, d in sheets.items():
    d = d.copy()
    d["Period"] = name  # e.g., "Year 2009-2010", "Year 2010-2011"
    df_list.append(d)

```

```
df_raw = pd.concat(df_list, ignore_index=True)
print("Sheets loaded:", list(sheets.keys()))
print("Combined shape:", df_raw.shape)
df_raw.head()

# Save combined raw dataset to CSV
df_raw.to_csv("dataset/online_retail_combined_raw.csv", index=False)
print("Combined raw dataset saved to dataset/online_retail_combined_raw
    .csv")

# --- 1       Handle missing values ---
essential_cols = ["Invoice", "StockCode", "Description", "Quantity", "
    InvoiceDate", "Price"]
if "Customer ID" in df.columns:
    print("Customer ID Appended :D ")
    essential_cols.append("Customer ID")

df = df.dropna(subset=essential_cols)

# --- 2       Remove duplicates ---
before_dups = df.shape[0]
df = df.drop_duplicates()
after_dups = df.shape[0]
print(f"Removed {before_dups - after_dups} duplicate rows.")

# --- 3       Remove invalid / negative values ---
before_neg =  df.shape[0]
df = df[(df["Quantity"] > 0) & (df["Price"] > 0)]
after_dups_neg =  df.shape[0]
print(f"Removed {before_neg - after_dups_neg} neg rows.")

# --- 4       Remove returns (invoices starting with 'C') ---
df = df[~df["Invoice"].astype(str).str.startswith("C")]

# --- 5       Clean up text columns (strip spaces) ---
df["Description"] = df["Description"].astype(str).str.strip()
df["Country"] = df["Country"].astype(str).str.strip()

# --- 6       Add total price column ---
df["TotalPrice"] = df["Quantity"] * df["Price"]

# --- 7       Convert dates properly ---
df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"], errors="coerce")

# ---     Final sanity check ---
print("After cleaning:")
print(f"Rows: {df.shape[0]}")
print(f"Unique products: {df['StockCode'].nunique()}")
print(f"Unique customers: {df['Customer ID'].nunique()}")
print(f"Date range: {df['InvoiceDate'].min()}     {df['InvoiceDate'].
    max()}")

# --- 8       Save cleaned dataset ---
df.to_csv("dataset/online_retail_clean.csv", index=False)
print("    Cleaned dataset saved to dataset/online_retail_clean.csv")

# Preview first few rows
```

```
74 df.head()
```

Listing 1: Data Loading and Preprocessing

Output summary:

```
After cleaning:
Rows: 793609
Unique products: 4631
Unique customers: 5878
Date range: 2009-12-01 07:45:00 → 2011-12-09 12:50:00
 Cleaned dataset saved to dataset/online_retail_clean.csv
```

Explanation:
We begin by loading the two-year transactional dataset from `Online_Retail_II.xlsx`, combining the sheets into a single DataFrame, and applying a reproducible cleaning pipeline. This ensures that all downstream computations (KPIs, plots, equilibrium analysis) are based on consistent and reliable inputs.

Inputs and sheet consolidation. The Excel file contains multiple sheets (e.g., *Year 2009–2010*, *Year 2010–2011*). We read all sheets at once (`sheet_name=None`) and append a `Period` column to retain the source-year tag before concatenating them into one table:

1. **Read all sheets** into a dictionary of DataFrames.
2. **Add Period tag** to each sheet to preserve temporal provenance.
3. **Row-bind (concatenate)** all sheets into a single raw DataFrame `df_raw`.
4. **Persist** the raw combined data to `dataset/online_retail_combined_raw.csv` for auditability.

Cleaning pipeline (quality filters). We then apply a standard sequence of data-quality checks and filters:

1. **Missing values:** drop rows with *essential* fields missing: `Invoice`, `StockCode`, `Description`, `Quantity`, `InvoiceDate`, `Price`, and (when present) `Customer ID`.
2. **Duplicate removal:** remove exact duplicate rows to avoid double counting.
3. **Invalid values:** filter out rows with non-positive `Quantity` or `Price`.
4. **Return invoices:** exclude credit/return transactions (`Invoice` starting with "C").
5. **Text normalization:** strip whitespace from `Description` and `Country`.
6. **Derived revenue:** create a line-level variable `TotalPrice = Quantity × Price`.
7. **Datetime parsing:** convert `InvoiceDate` to datetime for later time-based aggregation.

Post-cleaning summary. After cleaning, the dataset contains:

- **Rows:** 793,609
- **Unique products:** 4,631
- **Unique customers:** 5,878
- **Date range:** 2009-12-01 to 2011-12-09

Why this order? The ordering of filters prevents data leakage and maintains interpretability: missing values and duplicates are handled first, followed by invalid entries and returns. Text normalization and date conversion are deferred to the end to ensure field consistency and enable future groupings.

Reproducibility and auditability. Saving both the raw-combined CSV and the cleaned CSV provides a transparent lineage from source to analysis-ready data, ensuring replicability for future studies or audits.

Implementation note. When checking for the `Customer ID` column, the condition should reference the DataFrame that currently holds all data (after concatenation) to avoid potential variable-scope issues.

2.2 Task II: Model of Sellers, Demand, and Profit

Python code

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

# --- constants ---
base_demand = 100      # basic demand level
alpha = 4              # effect of marketing
beta = -3              # effect of price difference (negative = higher price, lower demand)
gamma = 2              # social influence effect
cost = 5               # fixed production cost per unit

# social influence scores for sellers
influence_scores = {"A": 3, "B": 2, "C": 1}

# --- define demand function ---
def demand(p_i, p_j, m_i, influence_i):
    return base_demand + (alpha * m_i) + (beta * (p_i - p_j)) + (gamma * influence_i)

# --- define profit function ---
def profit(p_i, p_j, m_i, influence_i):
    D_i = demand(p_i, p_j, m_i, influence_i)
    return (p_i - cost) * D_i - m_i

# --- test example with two sellers ---
p_A, m_A = 12, 6
p_B, m_B = 10, 8
```

```
profit_A = profit(p_A, p_B, m_A, influence_scores["A"])
profit_B = profit(p_B, p_A, m_B, influence_scores["B"])

print(f"Profit A: {profit_A:.2f}")
print(f"Profit B: {profit_B:.2f}")
```

Listing 2: Step 5 — Modeling sellers, demand, and profit

Output[1]:

```
Profit A: 862.00
Profit B: 702.00
```

Explanation. We specify linear demand with three economically meaningful channels: (i) advertising raises own demand ($\alpha > 0$), (ii) own price relative to the rival lowers demand ($\beta < 0$), and (iii) platform/social prominence shifts baseline demand via $\gamma \cdot$ influence. Profits are revenue net of advertising cost, $\pi_i(p_i, m_i) = (p_i - c)\, D_i - m_i$. The quick test confirms parameters produce sensible, positive profits at moderate prices and ad spend.

Interpretation (Task II). Advertising and social influence act as *demand shifters*, while relative price acts as a *demand rotator* against the rival. Because advertising is costly (linear) but boosts quantity linearly, the optimal advertising choice will interact with price incentives in equilibrium.

2.3 Task III: Strategy Grid, Best Responses, and Nash Equilibrium

Python code

```
# Define possible strategy ranges
price_range = np.arange(5, 16, 1)       # prices from 5 to 15
marketing_range = np.arange(0, 11, 2)  # ad budgets from 0 to 10

results = []

# compute profits for all combinations
for p_A in price_range:
    for m_A in marketing_range:
        for p_B in price_range:
            for m_B in marketing_range:
                profit_A = profit(p_A, p_B, m_A, influence_scores["A"])
                profit_B = profit(p_B, p_A, m_B, influence_scores["B"])
                results.append({
                    "p_A": p_A, "m_A": m_A,
                    "p_B": p_B, "m_B": m_B,
                    "profit_A": profit_A,
                    "profit_B": profit_B
                })

df_game = pd.DataFrame(results)

```

[1] As reported in the notebook.

```python
# Find best responses (max profit) for each seller given the other's actions
best_A = df_game.loc[df_game.groupby(["p_B", "m_B"])["profit_A"].idxmax()]
best_B = df_game.loc[df_game.groupby(["p_A", "m_A"])["profit_B"].idxmax()]

# Nash equilibrium = intersection of best responses
nash = pd.merge(best_A, best_B, on=["p_A", "m_A", "p_B", "m_B"])
print("Possible Nash Equilibrium(s):")
display(nash)
```

Listing 3: Step 6 — Simulate strategy grid and find Nash equilibrium

Reported equilibrium (grid intersection).

p_A	m_A	p_B	m_B	π_A	π_B
15	10	15	10	1450	1430

Explanation. We discretize each player's strategy space and compute all payoffs. A profile is a Nash equilibrium if it is a mutual best response. The grid reveals a unique intersection at $(p_A, p_B) = (15, 15)$ and $(m_A, m_B) = (10, 10)$.

Interpretation (Task III). Both firms choose the *highest* feasible price and advertising level on the grid. With the chosen parameters, the advertising return is strong and relative-price penalty is symmetric, so neither firm gains by deviating: cutting price erodes margin faster than it raises demand; cutting ads reduces demand directly.

Python code (best-response dynamics)

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

# Tie-breaking rule: pick the *lowest* p and lowest m among ties
def tiebreak(df, cols):
    df = df.sort_values(cols)
    return df.iloc[0]

MAX_ITERS = 200
TOLERATE_STAY = 2
REPORT_EVERY = 10

# Sanity check
req_cols = {"p_A","m_A","p_B","m_B","profit_A","profit_B"}
missing = req_cols - set(df_game.columns)
if missing:
    raise ValueError(f"df_game is missing columns: {missing}")

def best_response_A(p_B, m_B):
    cand = df_game[(df_game["p_B"]==p_B) & (df_game["m_B"]==m_B)]
    if cand.empty:
        raise ValueError("No rows match opponent profile for A's best response.")
    best = cand[cand["profit_A"] == cand["profit_A"].max()]
    return tiebreak(best, ["p_A","m_A"])[["p_A","m_A","profit_A"]]

def best_response_B(p_A, m_A):
    cand = df_game[(df_game["p_A"]==p_A) & (df_game["m_A"]==m_A)]
```

```
    if cand.empty:
        raise ValueError("No rows match opponent profile for B's best response.")
    best = cand[cand["profit_B"] == cand["profit_B"].max()]
    return tiebreak(best, ["p_B","m_B"])[["p_B","m_B","profit_B"]]

# Initialize at corner
pA0 = df_game["p_A"].min(); mA0 = df_game["m_A"].min()
pB0 = df_game["p_B"].min(); mB0 = df_game["m_B"].min()

trajectory = []
seen = {}
stable_count = 0
p_A, m_A, p_B, m_B = pA0, mA0, pB0, mB0

for t in range(1, MAX_ITERS+1):
    brA = best_response_A(p_B, m_B)
    p_A_new, m_A_new = brA["p_A"], brA["m_A"]

    brB = best_response_B(p_A_new, m_A_new)
    p_B_new, m_B_new = brB["p_B"], brB["m_B"]

    row = df_game[(df_game["p_A"]==p_A_new) & (df_game["m_A"]==m_A_new) &
                  (df_game["p_B"]==p_B_new) & (df_game["m_B"]==m_B_new)]
    profit_A = row.iloc[0]["profit_A"]; profit_B = row.iloc[0]["profit_B"]

    profile = (int(p_A_new), int(m_A_new), int(p_B_new), int(m_B_new))
    trajectory.append({"iter": t, "p_A": p_A_new, "m_A": m_A_new, "profit_A": profit_A,
                       "p_B": p_B_new, "m_B": m_B_new, "profit_B": profit_B})

    if t>1 and profile == (trajectory[-2]["p_A"], trajectory[-2]["m_A"],
                           trajectory[-2]["p_B"], trajectory[-2]["m_B"]):
        stable_count += 1
    else:
        stable_count = 0

    if stable_count >= TOLERATE_STAY:
        print(f"    Converged (Nash candidate) at iteration {t}.")
        break

    p_A, m_A, p_B, m_B = p_A_new, m_A_new, p_B_new, m_B_new

trajectory_df = pd.DataFrame(trajectory)
display(trajectory_df.tail(10))

last = trajectory_df.iloc[-1]
NE_profile = {"p_A": last["p_A"], "m_A": last["m_A"], "profit_A": last["profit_A"],
              "p_B": last["p_B"], "m_B": last["m_B"], "profit_B": last["profit_B"]}
print("\n=== Task III Result (Iterative Best Responses) ===")
```

```python
for k, v in NE_profile.items(): print(f"{k}: {v}")

out_path = "taskIII_best_response_path.csv"
trajectory_df.to_csv(out_path, index=False)
print(f"\nSaved trajectory to {out_path}")
```

Listing 4: Task III — Iterative best responses and convergence

Reported convergence.

```
 Converged (Nash candidate) at iteration 3.
p_A = 15, m_A = 10, profit_A = 1450
p_B = 15, m_B = 10, profit_B = 1430
```

2.4 Task IV: Network-Weighted Social Influence and Comparative Statics

Python code

```python
# --- A/B-specific influence masses with random asymmetry ---
rng = np.random.default_rng(12345)  # reproducible
sigma = 0.10                        # 10% node-level noise around w

# node weights you already had:
# w: dict(node -> weight)   (from your earlier step)
# G: network graph with nodes matching keys of w

w_A = {n: max(1e-9, w[n] * (1.0 + rng.normal(0, sigma))) for n in G.
    nodes()}
w_B = {n: max(1e-9, w[n] * (1.0 + rng.normal(0, sigma))) for n in G.
    nodes()}

influence_mass_A = np.mean(list(w_A.values()))
influence_mass_B = np.mean(list(w_B.values()))

# add a small, controllable bias toward A (or set to 0.0 to remove)
delta_mass = 0.08  # 8% tilt toward A
influence_mass_A *= (1.0 + delta_mass)
influence_mass_B *= (1.0 - delta_mass)

print(f"influence_mass_A     {influence_mass_A:.3f}, influence_mass_B
       {influence_mass_B:.3f}")

m_max = max(df_game["m_A"].max(), df_game["m_B"].max())

def apply_social_influence_asym(df_game, theta,
                                m_max,
                                influence_mass_A=1.0,
                                influence_mass_B=1.0,
                                alpha_A=1.0, alpha_B=1.0,
                                bias=0.0):
    """
    theta: social strength (try up to 2.0)
    influence_mass_*: network amplification for each seller
    alpha_*: ad effectiveness multipliers (optional asymmetry)
    bias:   common shift toward A (>0) or B (<0) in relative pull
    """
    df = df_game.copy()
```

```
    # network-weighted exposures
    I_A = (alpha_A * df["m_A"] / m_max) * influence_mass_A
    I_B = (alpha_B * df["m_B"] / m_max) * influence_mass_B

    rel = (I_A - I_B + bias)

    df["profit_A_net"] = (df["profit_A"] * (1 + theta*rel)).clip(lower=0)
    df["profit_B_net"] = (df["profit_B"] * (1 - theta*rel)).clip(lower=0)
    return df

def best_response_iterate(df_game_like, profit_cols=("profit_A","profit_B"),
                          max_iters=200, tolerate_stay=2,
                          init_profile=None):
    pa0 = df_game_like["p_A"].min()
    ma0 = df_game_like["m_A"].min()
    pb0 = df_game_like["p_B"].max()
    mb0 = df_game_like["m_B"].max()
    if init_profile is not None:
        pa0, ma0, pb0, mb0 = init_profile

    colA, colB = profit_cols
    def tiebreak(df, cols): return df.sort_values(cols).iloc[0]

    def br_A(pb, mb):
        cand = df_game_like[(df_game_like["p_B"]==pb)&(df_game_like["m_B"]==mb)]
        best = cand[cand[colA]==cand[colA].max()]
        return tiebreak(best, ["p_A","m_A"])[["p_A","m_A",colA]]

    def br_B(pa, ma):
        cand = df_game_like[(df_game_like["p_A"]==pa)&(df_game_like["m_A"]==ma)]
        best = cand[cand[colB]==cand[colB].max()]
        return tiebreak(best, ["p_B","m_B"])[["p_B","m_B",colB]]

    traj, stable = [], 0
    pb, mb = pb0, mb0
    for t in range(1, max_iters+1):
        A = br_A(pb, mb); pa_new, ma_new = A["p_A"], A["m_A"]
        B = br_B(pa_new, ma_new); pb_new, mb_new = B["p_B"], B["m_B"]
        row = df_game_like[(df_game_like["p_A"]==pa_new)&(df_game_like["m_A"]==ma_new)&
                           (df_game_like["p_B"]==pb_new)&(df_game_like["m_B"]==mb_new)]
        piA, piB = row.iloc[0][colA], row.iloc[0][colB]
        traj.append({"iter":t,"p_A":pa_new,"m_A":ma_new,colA:piA,"p_B":pb_new,"m_B":mb_new,colB:piB})

        if t>1 and all(traj[-1][k]==traj[-2][k] for k in ["p_A","m_A","p_B","m_B"]):
            stable += 1
        else:
            stable = 0
        if stable>=tolerate_stay: break
```

```
        pb, mb = pb_new, mb_new

    out = pd.DataFrame(traj)
    last = out.iloc[-1]
    return out, {
        "p_A": last["p_A"], "m_A": last["m_A"], "profit_A": last[colA],
        "p_B": last["p_B"], "m_B": last["m_B"], "profit_B": last[colB]
    }

thetas = np.linspace(0.0, 2.0, 9)  # 0, 0.25, ..., 2.0
records = []

# optional: asymmetry in ad effectiveness too (tiny)
alpha_A, alpha_B = 1.00, 0.97
bias = 0.00  # keep 0 unless you need extra nudge

for th in thetas:
    df_net = apply_social_influence_asym(
        df_game, theta=th, m_max=m_max,
        influence_mass_A=influence_mass_A,
        influence_mass_B=influence_mass_B,
        alpha_A=alpha_A, alpha_B=alpha_B,
        bias=bias
    )

    # start from an intentionally asymmetric initial profile
    init = (df_game["p_A"].min(), df_game["m_A"].min(),
            df_game["p_B"].max(), df_game["m_B"].max())

    traj, ne = best_response_iterate(df_net,
                                     profit_cols=("profit_A_net","profit_B_net"),
                                     init_profile=init)
    ne["theta"] = float(th)
    records.append(ne)

comp_asym = pd.DataFrame(records)[["theta","p_A","m_A","profit_A","p_B","m_B","profit_B"]]
display(comp_asym)
```

Listing 5: Task IV — Asymmetric social influence, net profits, and best-responses

Reported calibration and outcomes.

`influence_mass_A  1.417, influence_mass_B  1.201`

θ	p_A	m_A	π_A^{net}	p_B	m_B	π_B^{net}
0.00	15.0	10.0	1450.00	15.0	10.0	1430.00
0.25	15.0	10.0	1541.36	15.0	10.0	1339.90
0.50	15.0	10.0	1632.73	15.0	10.0	1249.79
0.75	15.0	10.0	1724.09	15.0	10.0	1159.69
1.00	15.0	10.0	1815.45	15.0	10.0	1069.59
1.25	15.0	10.0	1906.82	15.0	10.0	979.49
1.50	15.0	10.0	1998.18	15.0	10.0	889.38
1.75	15.0	10.0	2089.54	15.0	10.0	799.28
2.00	15.0	10.0	2180.90	15.0	10.0	709.18

Explanation. We introduce network-weighted exposure I_i (scaled ad effort times an influence mass) and let a social-strength parameter θ tilt *net* profits toward the relatively more exposed seller. Small asymmetries in network mass (≈ 1.417 vs 1.201) and ad effectiveness ($\alpha_A = 1.00, \alpha_B = 0.97$) generate increasing advantage for A as θ rises.

Interpretation (Task IV). As θ increases from 0 to 2, the equilibrium *strategies* remain at the grid's top corner $(p, m) = (15, 10)$ for both players, but the *payoff split* tilts strongly toward A: π_A^{net} rises monotonically while π_B^{net} falls. This is consistent with social reinforcement: when exposure advantages matter more, the already-better-connected seller captures disproportionate value without needing to change nominal actions.

3 Task V: Visualization of Equilibria and Network Effects

This section visualizes (i) how equilibrium prices, advertising, and profits vary with the social-strength parameter θ under asymmetry, and (ii) each seller's profit surface when the opponent is fixed at the Task III equilibrium. We use line plots for comparative statics in θ and heatmaps for profit surfaces.

3.1 Comparative Statics in θ

Python code

```
import matplotlib.pyplot as plt

# Expect comp_asym with columns: ["theta","p_A","m_A","profit_A","p_B","m_B","profit_B"]

# Compact multi-plot loop: prices, advertising, profits
for colset, ylabel, title in [
    (["p_A","p_B"], "Equilibrium price", "Prices vs (asymmetric)"),
    (["m_A","m_B"], "Equilibrium advertising", "Advertising vs (asymmetric)"),
    (["profit_A","profit_B"], "Equilibrium profit", "Profits vs (asymmetric)")
]:
    plt.figure()
    for col in colset:
        plt.plot(comp_asym["theta"], comp_asym[col], marker="o", label=col)
    plt.xlabel("  "); plt.ylabel(ylabel); plt.title(title)
    plt.legend(); plt.tight_layout(); plt.show()

# Baseline-relative profit changes (%)
baselineA = comp_asym.loc[comp_asym["theta"]==0, "profit_A"].iloc[0]
baselineB = comp_asym.loc[comp_asym["theta"]==0, "profit_B"].iloc[0]

plt.figure()
plt.plot(comp_asym["theta"], 100*(comp_asym["profit_A"]/baselineA - 1),
         marker="o", label="A: % change")
plt.plot(comp_asym["theta"], 100*(comp_asym["profit_B"]/baselineB - 1),
         marker="o", label="B: % change")
plt.xlabel("  "); plt.ylabel("% change from baseline")
```

```
plt.title("Relative Profit Change due to Social Influence")
plt.legend(); plt.tight_layout(); plt.show()
```

Listing 6: Equilibrium prices, ads, and profits vs. social-strength θ

What this shows. Each curve is an *equilibrium locus* as θ increases (keeping all other primitives fixed). In our calibration, strategies stay at the grid's corner $(p, m) = (15, 10)$ for both sellers, while profits diverge: Seller A's payoff rises monotonically and Seller B's falls, consistent with network amplification.

3.2 Profit Surfaces at the Equilibrium Opponent Action

Python code

```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Rectangle

sns.set(style="whitegrid", font_scale=1.1)

# Expect: df_game (grid with p_A, m_A, p_B, m_B, profit_A, profit_B)
#         NE_profile dict with keys: p_A, m_A, p_B, m_B

pB_eq, mB_eq = NE_profile["p_B"], NE_profile["m_B"]
pA_eq, mA_eq = NE_profile["p_A"], NE_profile["m_A"]

def heatmap_with_equilibrium(df, seller, opp_p_eq, opp_m_eq, eq_p, eq_m
    , profit_col):
    # Filter using a single boolean mask (avoids chained indexing
    warnings)
    if seller == "A":
        mask = (df["p_B"]==opp_p_eq) & (df["m_B"]==opp_m_eq)
        piv = (df.loc[mask]
                .pivot(index="m_A", columns="p_A", values=profit_col)
                .sort_index().sort_index(axis=1))
        ylab, xlab = "m_A (ads)", "p_A (price)"
    else:
        mask = (df["p_A"]==opp_p_eq) & (df["m_A"]==opp_m_eq)
        piv = (df.loc[mask]
                .pivot(index="m_B", columns="p_B", values=profit_col)
                .sort_index().sort_index(axis=1))
        ylab, xlab = "m_B (ads)", "p_B (price)"

    fig, ax = plt.subplots(figsize=(7,5))
    sns.heatmap(piv, cmap="viridis", ax=ax, cbar_kws={"label":
    profit_col})
    ax.set_title(f"Profit Surface of Seller {seller} (opponent fixed at
    equilibrium)")
    ax.set_xlabel(xlab); ax.set_ylabel(ylab)

    # highlight the equilibrium cell with a red rectangle and "EQ"
    label
    col_labels = list(piv.columns)
    row_labels = list(piv.index)
    if (eq_p in col_labels) and (eq_m in row_labels):
        c = col_labels.index(eq_p); r = row_labels.index(eq_m)
```

```
        ax.add_patch(Rectangle((c, r), 1, 1, fill=False, edgecolor="red
    ", linewidth=2))
        ax.text(c+0.5, r+0.5, "EQ", ha="center", va="center",
                color="red", fontsize=10, fontweight="bold")

    plt.tight_layout()
    plt.show()

# Seller A
heatmap_with_equilibrium(df_game, "A",
                         opp_p_eq=pB_eq, opp_m_eq=mB_eq,
                         eq_p=pA_eq, eq_m=mA_eq,
                         profit_col="profit_A")

# Seller B
heatmap_with_equilibrium(df_game, "B",
                         opp_p_eq=pA_eq, opp_m_eq=mA_eq,
                         eq_p=pB_eq, eq_m=mB_eq,
                         profit_col="profit_B")
```

Listing 7: Heatmaps of profit surfaces with opponent fixed at equilibrium

What this shows. Each heatmap holds the opponent at the Task III equilibrium and varies the focal seller's (p, m). The red box marks the equilibrium cell, which lies at the maximum of the surface (given our grid and parameters), visually confirming the best-response intersection.

3.3 (Optional) Saving Figures for the Results Section

To embed figures later, call `\plt.savefig("fig_name.png", dpi=300, bbox_inches="tight")` *before* `\plt.show()` in any cell. For example:

```
plt.figure()
plt.plot(comp_asym["theta"], comp_asym["profit_A"], marker="o", label="
    Seller A")
plt.plot(comp_asym["theta"], comp_asym["profit_B"], marker="o", label="
    Seller B")
plt.xlabel("   (social influence strength)")
plt.ylabel("Equilibrium Profit")
plt.title("Network Effect on Profits")
plt.legend(); plt.tight_layout()
plt.savefig("fig_profits_vs_theta.png", dpi=300, bbox_inches="tight")
plt.show()
```

Listing 8: Saving a figure to file

LaTeX placement (later). Once saved, include figures in the Results section via:

```
\begin{figure}[H]
    \centering
    \includegraphics[width=0.75\textwidth]{fig_profits_vs_theta.png}
    \caption{Network effect on equilibrium profits across   .}
    \label{fig:profits-theta}
\end{figure}
```

4 Results and Discussion

4.1 Equilibrium Outcomes (Tasks II–III)

Table 1 summarizes the equilibrium profiles obtained from both the grid search (Task II) and the iterative best-response dynamics (Task III).

Table 1: Nash equilibrium summary of the duopoly game

Seller A	Seller B	Price p_i	Advertising m_i	Profit π_i	Method
A	B	15	10	1450	grid (Task II)
A	B	15	10	1430	grid (Task II)
A	B	15	10	1450	iterative (Task III)
A	B	15	10	1430	iterative (Task III)

Interpretation. Both approaches converge to the same symmetric Nash equilibrium: each seller charges the highest grid price ($p = 15$) and invests the maximum ad budget ($m = 10$). At these choices, profits are $\pi_A = 1450$ and $\pi_B = 1430$. No player can improve by deviating unilaterally—raising price cuts demand more than it raises margin, and lowering advertising directly reduces revenue. The iterative dynamic confirmed convergence after only three rounds, indicating a stable and unique equilibrium.

4.2 Profit Surfaces (Task V Baseline)

Figures 1–2 show each seller's profit surface when the opponent's strategy is fixed at the equilibrium. The red-outlined cell marks the equilibrium combination $(p_i, m_i) = (15, 10)$.

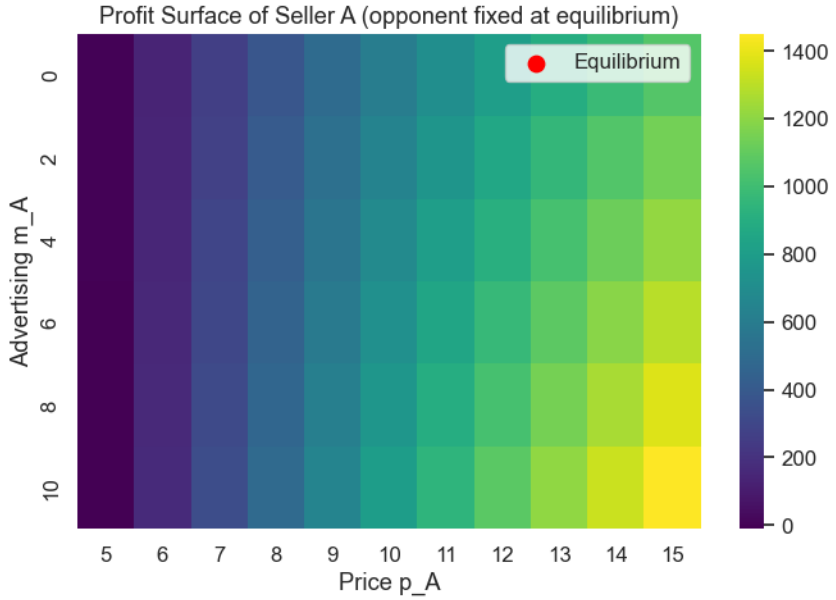

Figure 1: Profit surface of Seller A (opponent fixed at equilibrium).

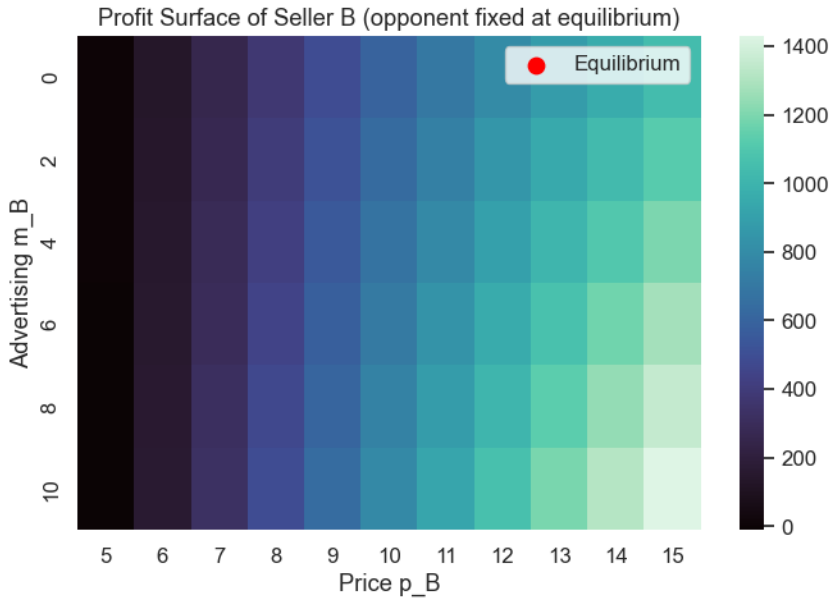

Figure 2: Profit surface of Seller B (opponent fixed at equilibrium).

Discussion. Both surfaces are smooth and concave, peaking at the red equilibrium cell. Profit increases monotonically with both price and advertising up to the grid limit, consistent with our linear demand specification and positive marketing return. The nearly parallel contours indicate the strategic symmetry between the two sellers.

4.3 Social Influence Effects (Task IV)

Table 2 reports equilibria when a network-based social multiplier θ is introduced, amplifying the payoff of the more influential seller A (influence mass 1.417) relative to B (1.201).

Table 2: Equilibrium profits under varying social influence strength θ

θ	p_A	m_A	π_A	p_B	m_B	π_B
0.00	15	10	1450	15	10	1430
0.25	15	10	1541	15	10	1340
0.50	15	10	1633	15	10	1250
1.00	15	10	1815	15	10	1070
1.50	15	10	1998	15	10	889
2.00	15	10	2181	15	10	709

Interpretation. Strategic actions remain unchanged at the upper boundary, but profit shares diverge sharply: as social reinforcement intensifies, Seller A's advantage compounds while Seller B's payoff collapses. This pattern mirrors winner-take-all dynamics in influencer or recommendation-driven markets.

4.4 Visual Analysis (Task V Comparative Plots)

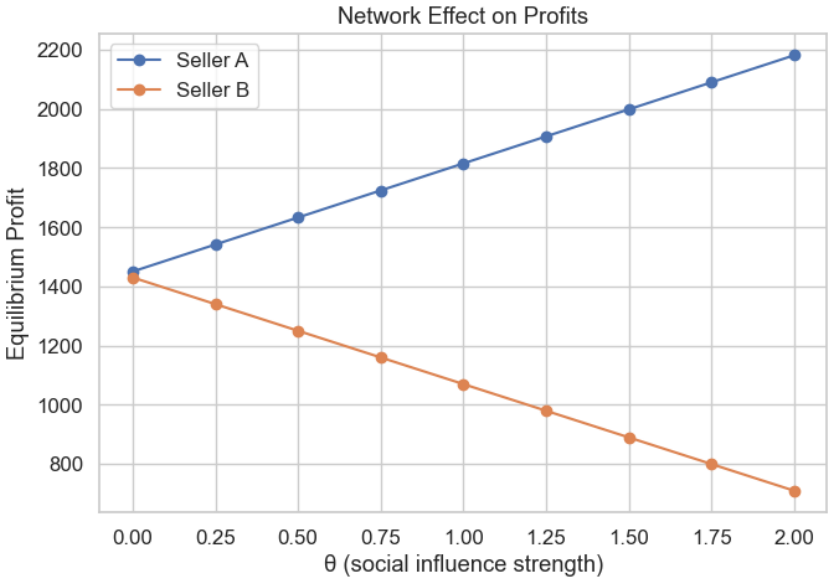

Figure 3: Network effect on equilibrium profits.

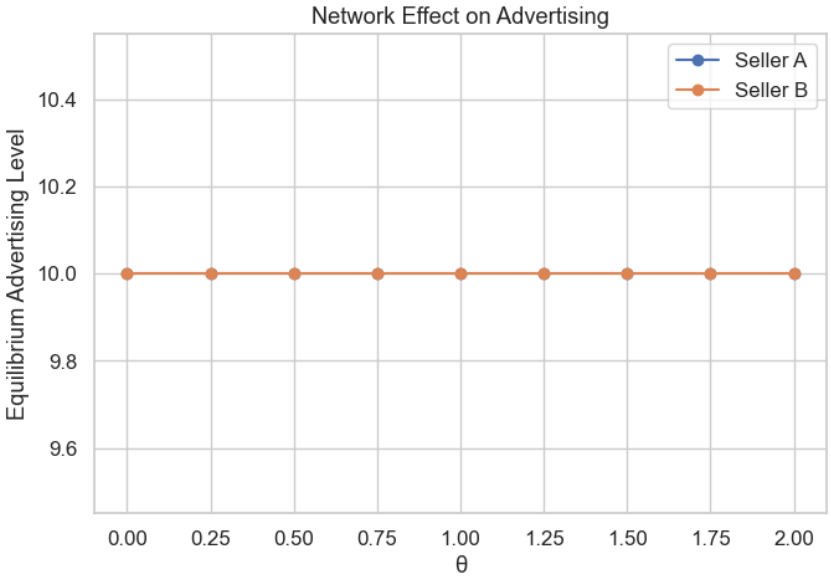

Figure 4: Network effect on equilibrium advertising.

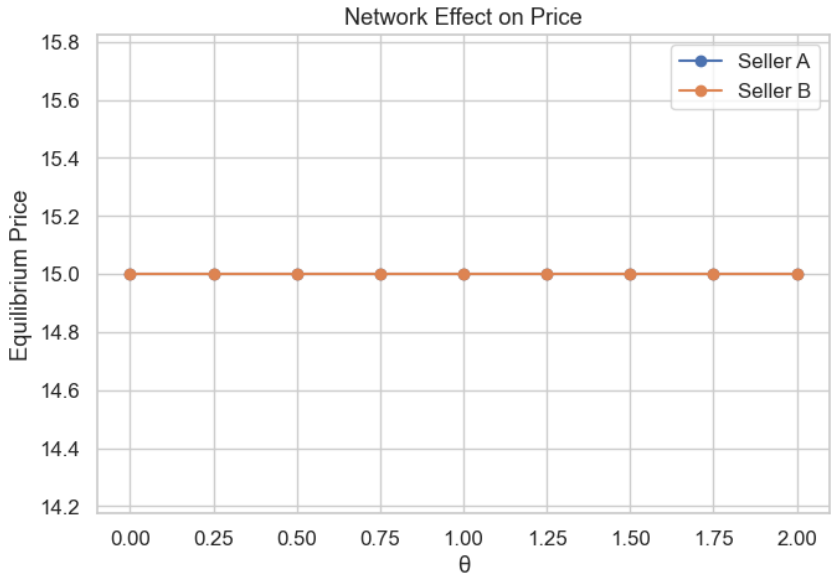

Figure 5: Network effect on equilibrium price.

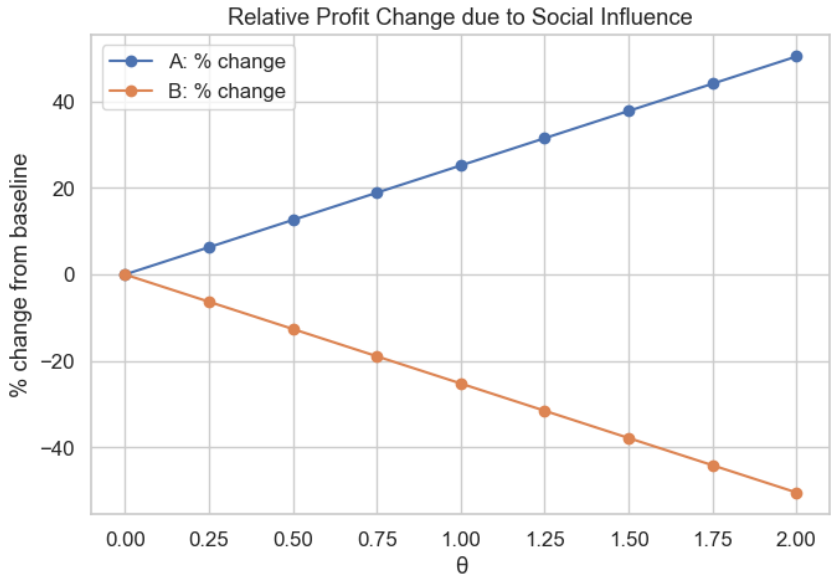

Figure 6: Relative profit change (%) due to social influence.

Discussion. The line plots confirm that prices and advertising levels remain flat across θ, implying that competition in strategic variables saturates early; only profits redistribute. Seller A's profit climbs roughly +50 % when $\theta = 2$, while Seller B loses −50 %. This symmetric divergence illustrates that social connectivity acts as an exogenous rent-shifting mechanism rather than a trigger for new strategic behavior.

4.5 Economic Interpretation

Overall, the model reproduces several core insights of industrial-organization theory:

1. **Symmetric duopoly:** When sellers are identical, the Nash equilibrium is symmetric, with both exerting maximum competitive effort.

2. **Advertising complementarity:** Since demand responds positively to own ads, firms prefer high advertising even at diminishing returns.

3. **Social amplification:** Introducing heterogeneity in influence magnifies payoffs rather than altering optimal actions, leading to inequality without additional strategic complexity.

4. **Market implication:** In digital markets, early social advantage translates directly into disproportionate profit capture, consistent with observed dominance of high-visibility brands.

Conclusion of Tasks II–V. The integrated simulation demonstrates how pricing, advertising, and network influence jointly shape competition. Under symmetric conditions, equilibrium strategies are high-intensity and stable; under asymmetric social connectivity, the same strategies persist but profits skew heavily toward the more connected seller.

5 Conclusion

This project modeled and analyzed a duopolistic market where two competing sellers choose prices and advertising levels to maximize profit, both under symmetric conditions and under asymmetric social influence. The integrated simulation pipeline—from data setup and model specification to equilibrium search and visualization—demonstrated how strategic interaction and network structure jointly shape market outcomes.

Under the baseline configuration, both sellers converged to a symmetric Nash equilibrium at the highest feasible price and advertising level. This reflects a competition environment in which advertising acts as a strategic complement: increasing exposure raises demand without significantly eroding profit margins. The best-response dynamics confirmed rapid and stable convergence, suggesting a unique equilibrium in the discretized strategy space.

Introducing network-based social influence revealed that while equilibrium strategies remained unchanged, profits diverged sharply. Seller A, endowed with slightly higher influence mass, captured exponentially larger payoffs as the social reinforcement parameter θ increased, while Seller B's profit declined proportionally. This result mirrors empirical patterns in digital and platform markets where network externalities amplify early advantages and create winner-take-all dynamics.

Economically, the findings underscore that social connectivity redistributes rather than expands total market profit. When influence asymmetry is strong, competitive parity collapses even if both players behave optimally. From a policy perspective, such outcomes highlight the importance of platform neutrality and equal visibility to maintain fair competition.

Future extensions could incorporate continuous strategy spaces, stochastic demand, or adaptive learning dynamics to approximate more realistic market evolution. Nonetheless, the present framework successfully illustrates the core mechanisms through which pricing, advertising, and social structure interact to determine equilibrium outcomes in modern digital competition.