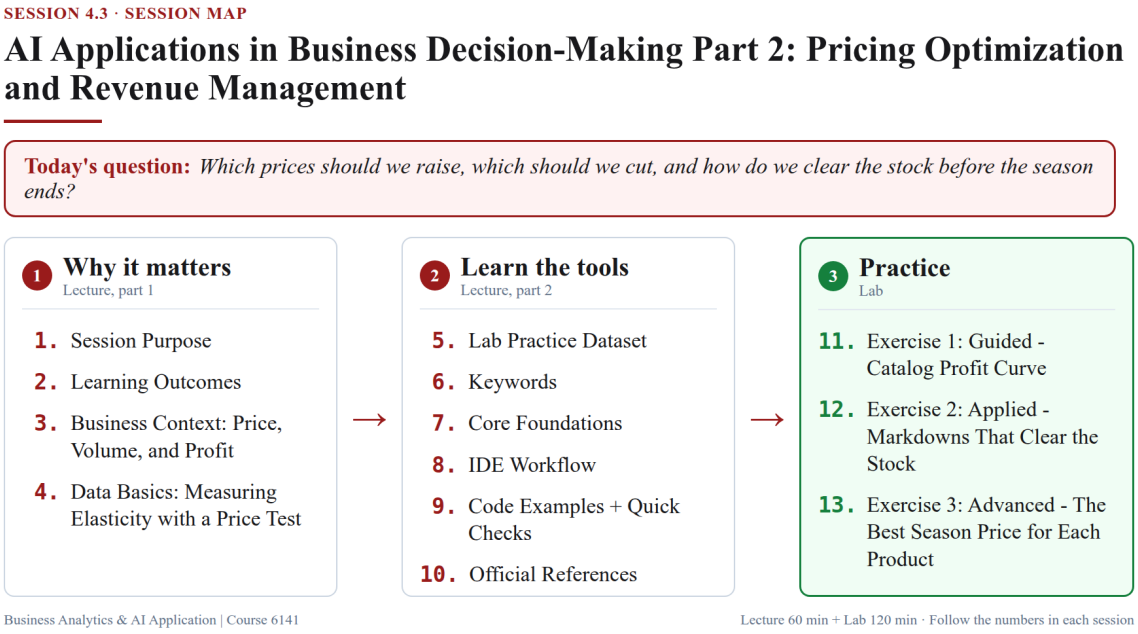
Session 4.3: AI Applications in Business Decision-Making Part 2: Pricing Optimization and Revenue Management

Course 6141: Business Analytics & AI Application | Reading for Week 15



Schedule: Week 15 | In-person classes (1:2 Delivery Ratio: 60 Min Lecture + 120 Min Lab)

1. Session Purpose

- An online retailer sells 12 seasonal products; the season ends in 12 days, and stock left after that must be sold off cheaply
- Prices were set by gut feeling: some may be too low (leaving profit on the table), others too high (leaving stock unsold)
- The pricing manager asks: which prices should we raise, which should we cut, and how do we clear the stock before the season ends? Today, you measure how customers react to price and simulate profit before changing a single price

2. Expected Learning Outcomes (By the end of this session, you will be able to)

1. "How strongly do customers react to price?": estimate the price elasticity of demand (the % change in units sold for a 1% change in price) from a price test
2. "Would a price change raise profit?": simulate units, revenue, and gross profit for several price changes before making them
3. "How do we clear the stock in time?": calculate the price cut each product needs to sell out before the season ends, and fix a `SyntaxError` (an unclosed bracket)

3. Business Context: Price, Volume, and Profit

- A lower price sells more units but earns less on each one; a higher price earns more per unit but sells fewer. Profit depends on both:

Term	Meaning	Example (`PRD-200`)
Unit margin	price - unit cost	`$49.99 - $22.50 = $27.49`
Gross margin %	unit margin / price x 100	`55.0%`
Daily gross profit	unit margin x daily units	`$27.49 x 32 = $879.68`
Days of stock	inventory / daily units	`450 / 32 = 14.1` days

- Price elasticity of demand (PED): if elasticity is `-1.5`, a 1% price increase loses about 1.5% of units. When elasticity is between `0` and `-1`, customers barely react; below `-1`, they react strongly.
- The goal is profit, not revenue: a cut can raise revenue and still lower profit, because each extra unit brings in less margin.
- Seasonal stock: units left at the season's end lose most of their value. Our retailer sells leftovers to a liquidator (a company that buys surplus stock) for 40% of their unit cost, a loss of 60% of the cost on every unsold unit.

4. Data Basics: Measuring Elasticity with a Price Test

- To measure how customers react, the retailer changed the price of `PRD-202` once a week for 5 weeks and recorded daily units sold:

Week	Price	Daily units
1	`$59.99`	`27.9`
3	`$69.99`	`22.0`
5	`$79.99`	`18.1`

- Elasticity is the slope of a straight line through `log(units)` against `log(price)`. A logarithm (log) turns percentage changes into differences, so the slope reads directly as "% change in units for a 1% change in price".

- `np.polyfit(x, y, 1)` fits that straight line and returns its slope and intercept. Result: elasticity `-1.51`.
- Using elasticity for a small price change: new units = old units x (1 + elasticity x price change). For `+5%`: `1 + (-1.51 x 0.05) = 0.9245`, so about 7.6% fewer units. This straight-line rule is an approximation that works for changes of up to about 20%.

5. Lab Practice Dataset

- Two files (fictional teaching data):
  - Catalog: `data/session_04_03_dataset.csv`, 12 products, one row per product.
  - Price test: `data/session_04_03_price_test.csv`, 5 weeks of tested prices for `PRD-202`.
- Catalog columns:
  - `Product_ID`: Product code (e.g., `PRD-201`).
  - `Current_Price_USD`: Our current price in U.S. Dollars.
  - `Competitor_Price_USD`: The main competitor's price for the same product.
  - `Inventory_Units_Count`: Units in stock.
  - `Historical_Daily_Sales_Units`: Average units sold per day at the current price.
  - `Unit_Cost_USD`: What one unit costs us, delivered to the warehouse.
- Price test columns: `Test_Week` (1 to 5), `Product_ID`, `Test_Price_USD`, and `Avg_Daily_Units` (average units sold per day that week).

6. Keywords (Core Programming Concepts)

- `Price Elasticity of Demand (PED)`:
  - What: the % change in units sold for a 1% change in price
  - Why: tells you whether a price rise loses a few customers or many
- `Unit Margin` and `Gross Profit`:
  - What: price minus unit cost, and unit margin times units sold
  - Why: the measure a pricing decision should maximize
- `Price Simulation`:
  - What: computing units and profit for several possible prices before choosing one
  - Why: test a decision on paper, not on real customers
- `Days of Stock`:
  - What: inventory divided by daily sales
  - Why: shows which products will not sell out in time
- `Markdown`:
  - What: a planned price cut to sell stock faster
  - Why: a smaller, earlier cut usually loses less than a fire sale at the end

- `Salvage Value`:
  - What: what leftover stock is still worth after the season
  - Why: decides how costly it is to be left with unsold units

7. Core Programming & Computational Foundations

- **7.1 Price elasticity: how strongly customers react**
  - *The idea:* Price elasticity of demand (PED) is the percentage change in units sold for a 1% change in price. It is usually negative: a higher price sells fewer units. With an elasticity of `-1.51`, a 1% price rise loses about 1.51% of units. Values between `0` and `-1` mean customers barely react; values below `-1` mean they react strongly.
  - *See it on real data:* the price test for `PRD-202`, week by week.

Weeks	Price change	Units change	Elasticity of that step
1 to 2	`+8.3%`	`-12.5%`	`-1.51`
2 to 3	`+7.7%`	`-9.8%`	`-1.28`
3 to 4	`+7.1%`	`-10.9%`	`-1.53`
4 to 5	`+6.7%`	`-7.7%`	`-1.15`

    Each single step is noisy. Fitting one line through all 5 weeks gives one steadier estimate: `-1.51`.
  - *Common mistake:* trusting one week-to-week comparison. Sales also move for reasons other than price (weather, holidays, a competitor's promotion); use several test weeks.
  - *More examples:*

Elasticity	Units after a 10% price rise	Revenue effect (1.10 x units factor)
`-0.5`	`-5%`	`+4.5%`: revenue rises
`-1.0`	`-10%`	about `-1%`: revenue almost flat
`-1.51` (ours)	`-15.1%`	`-6.6%`: revenue falls
`-2.0`	`-20%`	`-12%`: revenue falls sharply

  - *Remember:* elasticity = % change in units / % change in price; estimate it from several observations.
- **7.2 Why logs: percentages become straight lines**
  - *The idea:* Elasticity is about percentage changes, but a straight line measures fixed changes. The natural logarithm (`np.log`) turns one into the other: a 10% rise in price always adds the same amount, about `0.095`, to the log of the price, whatever the starting price. So on a log-log chart, constant elasticity becomes a straight line, and its slope is the elasticity.
  - *See it on real data:*

Price	np.log(price)
59.99	4.094
69.99	4.248
79.99	4.382

  - *In code:* `slope, intercept = np.polyfit(np.log(prices), np.log(units), 1)` fits the line; the `1` means a straight line. The slope is the elasticity.
  - *More examples:*

Code	Result	What it tells you
`np.log(1)`	`0.0`	the log of 1 is 0
`np.log(70) - np.log(60)`	`0.154`	close to the % change, `16.7%`: logs measure relative change
`np.log(1.10)`	`0.0953`	a 10% rise, whatever the starting price
`np.exp(0.0953)`	`1.1`	`np.exp()` undoes `np.log()`

  - *Remember:* log price against log units: the slope is the elasticity.

- **7.3 Simulating a price change**
  - *The idea:* Before changing a price, compute what would happen. For a small change, new units = old units x (1 + elasticity x price change). Then profit = (new price - unit cost) x new units. Writing this once as a function, `simulate(price_change)`, guarantees every scenario uses the same formula.
  - *See it on real data:* `PRD-200` (price `$49.99`, cost `$22.50`, 32 units a day).

Price change	New price	Daily units	Daily revenue	Daily gross profit
-10%	$44.99	36.83	$1,657.11	$828.39
0%	$49.99	32.00	$1,599.68	$879.68
+5%	$52.49	29.58	$1,552.85	$887.21

    The cut brings the most revenue but the least profit; the rise loses units but earns the most.
  - *Common mistake:* maximizing revenue. Every extra unit sold at a lower price adds less margin, so revenue can rise while profit falls.
  - *More examples:* the straight-line rule vs the exact constant-elasticity formula, `(1 + change) ** elasticity`.

Price change	Straight-line units factor	Exact factor
+5%	0.9245	0.9290
+10%	0.8490	0.8660
-10%	1.1510	1.1724
+20%	0.6980	0.7593

Price change	Straight-line units factor	Exact factor

The two agree closely for small changes and drift apart for large ones: keep simulated changes within about 20%.

- *Remember:* simulate profit, not revenue, and keep changes small.

- **7.4 Stock and time: days of stock and markdowns**
  - *The idea:* Seasonal stock has a deadline. Days of stock = inventory / daily units shows how long the stock lasts at today's price. If that is longer than the days left in the season, the product needs a markdown (a planned price cut) to sell faster.
  - *See it on real data:* `PRD-202`, 12 days left in the season.

Step	Calculation	Result
Days of stock now	`290 / 22`	`13.2` days: too slow
Units needed per day	`290 / 12`	`24.17`
Units must rise by	`24.17 / 22 - 1`	`9.9%`
Price change needed	`9.9% / -1.51`	`-6.5%`

The last step reverses the elasticity rule: if units must rise by a given percentage, the price must change by that percentage divided by the elasticity.

- *More examples:*

Product	Days of stock	Units needed per day	Price change to sell out in 12 days
`PRD-200`	`14.1`	`37.50`	`-11.4%`
`PRD-210`	`12.5`	`4.17`	`-2.8%`: almost on time
`PRD-204`	`12.0`	`15.00`	`0%`: sells out exactly on time
`PRD-203`	`11.1`	`17.50`	`+5.2%`: sells out early, so the price could rise

- *Remember:* compare days of stock with days left; cut just enough to sell out.

- **7.5 Leftover stock has a cost: salvage value**
  - *The idea:* Units left after the season are sold to a liquidator for 40% of their cost, a loss of 60% of the cost on each one. So the true season profit is profit on units sold minus the loss on units left.
  - *See it on real data:* `PRD-200` over the 12-day season (450 units in stock).

Price	Units sold	Units left	Season profit
`$49.99` (no change)	`384`	`66`	`$9,665.16`
`$44.99` (`-10%`)	`442`	`8`	`$9,832.00`

Price	Units sold	Units left	Season profit

The cut earns less per unit but avoids most of the leftover loss: `$167` more profit for the season.

– *More examples:*

Product and price	Sold	Left over	Season profit
`PRD-202` at `$69.99`	`264`	`26`	`$9,530.16`
`PRD-202` at `$66.49` (`-5%`)	`284`	`6`	`$9,676.31`: the cut pays
`PRD-210` at `$149.99`	`48`	`2`	`$3,706.32`
`PRD-210` at `$142.49` (`-5%`)	`50`	`0`	`$3,574.50`: selling out is not worth the cut
`PRD-204` at `$94.49` (`+5%`)	`166`	`14`	`$8,566.96`: a rise leaves stock behind

– *Remember:* count the cost of leftovers when you choose a price.

8. Development Workflow in Google Antigravity IDE

- `[1]` File Explorer: create your script with your name and the session code (for example, `Reacher_S43_demo_1.py`), and check both files are in `data/`.
- `[2]` Code Editor: write a few lines, run, repeat. Small steps keep errors small. The editor highlights matching brackets when you place the cursor next to one.
- `[3]` AI Agent Chat: on an error, capture the traceback (Python's error report) with `Win + Shift + S` (macOS: `Cmd + Shift + 4`), paste it with `Ctrl + V`, and ask why it failed.
- `[4]` Terminal: run `python` followed by your script name. Return code `0` means the script finished with no errors.

9. Hands-On Coding Micro-Examples

- Every example has three parts: **1. Input**, **2. Process**, and **3. Output**. Name each file with your own name and the session code, for example `Reacher_S43_demo_1.py` (pattern: `YourName_S43_demo_1.py`).
- *Micro-Example 1 (Estimate Elasticity from the Price Test):* `YourName_S43_demo_1.py`

```
# 1. Input: 5 weeks of tested prices for PRD-202
import numpy as np
import pandas as pd
test = pd.read_csv("data/session_04_03_price_test.csv")

# 2. Process: a straight line through log(units) against log(price)
slope, intercept = np.polyfit(np.log(test["Test_Price_USD"]),
```

```
                                np.log(test["Avg_Daily_Units"]), 1)
elasticity = round(slope, 2)

# 3. Output: the elasticity and what it means
print("Price elasticity of demand:", elasticity)
print(f"A 1% price rise loses about {abs(elasticity)}% of units sold.")
```

  - *How it works, line by line:*
    - `import numpy as np` loads NumPy, the number library that pandas is built on.
    - `np.log(...)` takes the logarithm of every price and every unit count (section 7.2).
    - `np.polyfit(x, y, 1)` fits the best straight line through the 5 points and returns its slope and intercept. The slope is the elasticity.
    - `round(slope, 2)` keeps 2 decimals; `abs()` drops the minus sign for the sentence.
  - *Read the output:*

```
Price elasticity of demand: -1.51
A 1% price rise loses about 1.51% of units sold.
```

    - `-1.51` is below `-1`: customers of this product react strongly to price.
    - A 10% rise would lose about 15% of units; whether that raises profit depends on the margin, which Micro-Example 2 checks.

- *Quick Check 1 (`.pct_change()`):*
  - New tool: `column.pct_change()` gives the change from the previous row, as a share (`0.08` = 8%). Why: shows the week-to-week reactions behind the single slope.
  - Task: add one line to part **3. Output**: `print((test[["Test_Price_USD", "Avg_Daily_Units"]].__________() * 100).round(1))`
  - Check: the second row shows `8.3` (price up 8.3%) and `-12.5` (units down 12.5%). The first row is `NaN`, because week 1 has no previous week.

- *Micro-Example 2 (Simulate Profit for Several Price Changes):* `YourName_S43_demo_2.py`

```
# 1. Input: the catalog and the elasticity from Micro-Example 1
import pandas as pd
df = pd.read_csv("data/session_04_03_dataset.csv")
elasticity = -1.51

# 2. Process: one function, used for every price change
def simulate(price_change):
    price = df["Current_Price_USD"] * (1 + price_change)
    units = df["Historical_Daily_Sales_Units"] * (1 + elasticity * price_change)
    return round(((price - df["Unit_Cost_USD"]) * units).sum(), 2)

profit_curve = {}
for change in [-0.10, -0.05, 0.0, 0.05, 0.10]:
    profit_curve[f"{change * 100:+.0f}%"] = simulate(change)
```

```
# 3. Output: daily gross profit for the whole catalog at each change
for label, profit in profit_curve.items():
    print(f"{label}: ${profit:,.2f}")
```

  - *How it works, line by line:*
    - `elasticity = -1.51` uses the value measured in Micro-Example 1 for every product. In a real business, each product group would get its own test.
    - `simulate(price_change)` computes the new price, the new units (section 7.3), and the total daily gross profit of the whole catalog.
    - The loop runs the function for 5 price changes and stores each result in a dictionary, labeled like `+5%`.
    - `f"{change * 100:+.0f}%"` writes the change with its sign: `+5%`, `-10%`.
  - *Read the output:*

```
-10%: $6,881.85
-5%: $7,163.83
+0%: $7,342.85
+5%: $7,418.90
+10%: $7,391.98
```

    - Profit rises from `-10%` up to `+5%`, then falls again at `+10%`: there is a best point in between.
    - A 5% rise adds about `$76` a day across the catalog. The margins are high (above 52%), so the profit on the remaining units outweighs the units lost.

- *Quick Check 2 (`max(..., key=...)`):*
  - New tool: `max(profit_curve, key=profit_curve.get)` returns the label with the highest profit. Why: finds the best scenario without reading the list by eye.
  - Task: add one line to part **3. Output**: `print("Best change:", ___(profit_curve, key=profit_curve.get))`
  - Check: the Terminal shows `Best change: +5%`.

- *Micro-Example 3 (Which Products Will Not Sell Out in 12 Days?):* `YourName_S43_demo_3.py`

```
# 1. Input: the catalog
import pandas as pd
df = pd.read_csv("data/session_04_03_dataset.csv")
season_days = 12

# 2. Process: days of stock and our price gap to the competitor
df["Days_Of_Stock"] = (
    df["Inventory_Units_Count"] / df["Historical_Daily_Sales_Units"]
).round(1)
df["Price_Gap_Pct"] = (
    (df["Current_Price_USD"] - df["Competitor_Price_USD"])
    / df["Competitor_Price_USD"] * 100
).round(1)
at_risk = df[df["Days_Of_Stock"] > season_days]
```

```
# 3. Output: the products that need action
print(at_risk[["Product_ID", "Days_Of_Stock", "Price_Gap_Pct"]])
```

  - *How it works, line by line:*
    - `Days_Of_Stock` divides inventory by daily sales (section 7.4).
    - `Price_Gap_Pct` compares our price with the competitor's: negative means we are cheaper.
    - `df[df["Days_Of_Stock"] > season_days]` keeps the products that will not sell out in 12 days.
  - *Read the output:*

	Product_ID	Days_Of_Stock	Price_Gap_Pct
0	PRD-200	14.1	-3.9
1	PRD-201	13.6	2.5
2	PRD-202	13.2	-2.8
9	PRD-209	13.0	3.7
10	PRD-210	12.5	-1.3
11	PRD-211	13.3	3.2

    - 6 of 12 products would still have stock at the end of the season.
    - `PRD-201`, `PRD-209`, and `PRD-211` are also more expensive than the competitor: a price cut there has two good reasons.
    - `PRD-200` is already cheaper than the competitor, yet still too slow: its stock is simply too large for the season.

- *Quick Check 3 (`np.where()`):*
  - New tool: `np.where(condition, value_if_true, value_if_false)` fills a whole column with one of two values. Why: a one-line alternative to setting a default and overwriting it with `.loc[]`.
  - Task: add `import numpy as np` to part **1. Input**, and one line at the end of part **2. Process**: `df["Action"] = np._____(df["Days_Of_Stock"] > season_days, "Markdown", "Keep")`. Then print `df["Action"].value_counts()`.
  - Check: `Keep` 6, `Markdown` 6.

10. Official Python Documentation & References

- NumPy API Reference: `numpy.polyfit`.
- NumPy API Reference: `numpy.where`.
- Phillips: Pricing and Revenue Optimization (Stanford Business Books).
- Talluri & van Ryzin: The Theory and Practice of Revenue Management (Springer Standards).

11. Exercise 1: Guided Practice - Catalog Profit Curve

- *Step 1: Open Script:* In Google Antigravity IDE, create a new script named `YourName_S43_pricing.py`, replacing `YourName` with your own name (for example, `Reacher_S43_pricing.py`).

- *Step 2: Simulate a Wider Range of Price Changes:* Type the following code:

```
# 1. Input: the catalog and the measured elasticity
import pandas as pd
df = pd.read_csv("data/session_04_03_dataset.csv")
elasticity = -1.51

# 2. Process: today's unit economics, then a profit curve from -15% to +15%
df["Unit_Margin_USD"] = df["Current_Price_USD"] - df["Unit_Cost_USD"]
df["Gross_Margin_Pct"] = (df["Unit_Margin_USD"] / df["Current_Price_USD"] *
 ↪  100).round(1)
rows = []
for change in [-0.15, -0.10, -0.05, 0.0, 0.05, 0.10, 0.15]:
    price = df["Current_Price_USD"] * (1 + change)
    units = df["Historical_Daily_Sales_Units"] * (1 + elasticity * change)
    rows.append({
        "Price_Change_Pct": round(change * 100),
        "Daily_Units": round(units.sum(), 1),
        "Daily_Revenue_USD": round((price * units).sum(), 2),
        "Daily_Gross_Profit_USD": round(((price - df["Unit_Cost_USD"]) *
         ↪  units).sum(), 2),
    })
curve = pd.DataFrame(rows)

# 3. Output: margins today, then the curve
print("Gross margin % by product:", df["Gross_Margin_Pct"].tolist())
print(curve)
```

- *Step 3: Interpret:* In two comment lines, explain why the price change with the highest revenue is not the one with the highest profit.
- *Step 4: Execute in Terminal:* Run `python YourName_S43_pricing.py` in Panel `[4]` and confirm return code `0`.
- *Success Check:* gross margins range from `52.5`% to `55.0`%. Gross profit peaks at `+5` (`$7,418.90`), while revenue is highest at `-15` (`$14,218.32`): the change that brings in the most money is not the one that keeps the most. Cutting prices by 15% sells the most units but earns the least profit (`$6,496.89`).

12. Exercise 2: Applied Practice - Markdowns That Clear the Stock

- *Coding Challenge:* In `YourName_S43_practice.py`, calculate the price cut each at-risk product needs to sell out in 12 days.
- *Task 1 (Required Price Change):* For the products with more than 12 days of stock (Micro-Example 3):
  - `Required_Daily_Units` = `Inventory_Units_Count` / 12
  - `Price_Change_Pct` = (`Required_Daily_Units` / `Historical_Daily_Sales_Units` - 1) / elasticity x 100, rounded to 1 decimal. It is negative: a cut.

  - This reverses the rule in section 4: if units must rise by a given share, the price must change by that share divided by the elasticity.

- *Task 2 (Read the SyntaxError & AI Diagnosis):* Write the new price line with a missing bracket: `df["New_Price_USD"] = df["Current_Price_USD"] * (1 + df["Price_Change_Pct"] / 100`. Run the script; Python stops before running anything with `SyntaxError: '(' was never closed`. Capture the message with `Win + Shift + S`, paste it into Panel `[3]` (AI Agent Chat) with `Ctrl + V`, and ask where the bracket is missing.
- *Task 3 (Fix and Export):* Close the bracket, round `New_Price_USD` to 2 decimals, and save `Product_ID`, `Current_Price_USD`, `Price_Change_Pct`, and `New_Price_USD` for the at-risk products to `data/markdown_plan.csv` with `index=False`.
- *Success Check:* price changes: `PRD-200` `-11.4%`, `PRD-201` `-8.7%`, `PRD-202` `-6.5%`, `PRD-209` `-5.5%`, `PRD-210` `-2.8%`, and `PRD-211` `-7.4%`. The deepest cut goes to the product with the most stock left relative to its sales, not to the most expensive one.

13. Exercise 3: Advanced Practice - The Best Season Price for Each Product

- *Goal:* Find, for each product, the price that earns the most over the 12-day season, counting the loss on any stock left over. Some prices may go up and some down. Show the manager how much the plan adds compared with keeping today's prices.
- *What you need:*
  - File: `data/session_04_03_dataset.csv` (described in **5. Lab Practice Dataset**).
  - Script name: `YourName_S43_challenge.py`.
  - Tools you already know: functions, `for` loops over rows and price changes, `min()`, `max()`, lists of dictionaries, and `pd.DataFrame()`.
- *Rules of the season:*
  - Season length: `12` days; elasticity: `-1.51`; daily units at a new price: units x (1 + elasticity x change).
  - Units sold in the season: the smaller of (daily units x 12) and the inventory: you cannot sell stock you do not have.
  - Unsold units go to a liquidator at 40% of unit cost, a loss of `0.6 x unit cost` each.
  - Season profit = (price - unit cost) x units sold - 0.6 x unit cost x unsold units.
- *Requirements (follow them in order):*
  1. Write `season_profit(price, unit_cost, daily_units, inventory)` that returns the season profit, following the rules above.
  2. For each product, try the price changes `-20%`, `-15%`, `-10%`, `-5%`, `0%`, `+5%`, `+10%`, `+15%`, and `+20%`. Round each new price to 2 decimals before you use it.
  3. Keep, for each product, the change with the highest season profit, and also record the season profit at `0%` (today's price).
  4. Build one DataFrame with `Product_ID`, `Best_Change_Pct`, `Best_Price_USD`, `Best_Season_Profit_USD`, and `Current_Season_Profit_USD`, profits rounded to 2 decimals.
  5. Print the table, the total of both profit columns, and the difference.
  6. Count how many products get a price cut, how many a price rise, and how many keep today's price.

- *New tools for this exercise:*
  - Looping over rows with `for _, row in df.iterrows():`. Each `row` behaves like a small dictionary: `row["Unit_Cost_USD"]`. The `_` holds the row label, which we do not need.
  - `min(a, b)` returns the smaller of two numbers.
- *Hints:*
  - Test `season_profit()` on `PRD-200` at `0%` first: it should return about `9665.16`, because 66 units are left over.
  - Inside the product loop, keep the best result so far in two variables and update them when a change earns more.
- *Self-Check (expected results):*
  - Best changes: cuts for `PRD-200` (`-10%`), `PRD-201`, `PRD-202`, `PRD-209`, and `PRD-211` (`-5%` each); rises of `+5%` for `PRD-203`, `PRD-205`, `PRD-206`, and `PRD-208`; no change for `PRD-204`, `PRD-207`, and `PRD-210`.
  - Total season profit: `$85,969.93` with the plan against `$83,590.57` at today's prices: `$2,379.36` more.
  - Business meaning: one rule for all products leaves money behind. Products with too much stock need a cut to avoid leftover losses; products that will sell out anyway can carry a higher price. Note that for `PRD-210`, a cut that clears all stock earns less than keeping the price and selling 2 leftover units to the liquidator.