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

## THE BUSINESS SITUATION

- 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

Today's flow: 60 min lecture, then 120 min hands-on lab

## EXPECTED OUTCOMES

### By the end of today, 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)

# Session Map: Follow the Numbers

**Today's question:** *Which prices should we raise, which should we cut, and how do we clear the stock before the season ends?*

## 1 Why it matters
Lecture, part 1

1. **Session Purpose**
   answer today's question
2. **Learning Outcomes**
   3 skills you will show today
3. **Business Context**
   Price, Volume, and Profit
4. **Data Basics**
   Measuring Elasticity with a Price Test

→

## 2 Learn the tools
Lecture, part 2

5. **Lab Dataset**
   2 CSVs
6. **Keywords**
   6 terms
7. **Core Foundations**
   5 ideas, one at a time
8. **IDE Setup**
   name files, run in the Terminal
9. **Code Examples**
   3 demos + Quick Checks
10. **References**
    3 official docs

→

## 3 Practice
Lab

11. **Exercise 1: Guided**
    Catalog Profit Curve
12. **Exercise 2: Applied**
    Markdowns That Clear the Stock
13. **Exercise 3: Advanced**
    The Best Season Price for Each Product

# Price, Volume, and Profit

## KEY POINTS

- A lower price sells more units but earns less on each one; a higher price earns more per unit but sells fewer.
- Price elasticity of demand (PED): if elasticity is `-1.5`, a 1% price increase loses about 1.5% of units.
- 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.

## SEE IT ON REAL DATA

| 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 |

# Measuring Elasticity with a Price Test

## KEY POINTS

- To measure how customers react, the retailer changed the price of `PRD-202` once a week for 5 weeks and recorded daily units sold:
- Elasticity is the slope of a straight line through `log(units)` against `log(price)`.
- `np.polyfit(x, y, 1)` fits that straight line and returns its slope and intercept.
- Using elasticity for a small price change: new units = old units x (1 + elasticity x price change).

## AT A GLANCE

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

# The Data for Today

## FILES

- `data/session_04_03_dataset.csv`
- `data/session_04_03_price_test.csv`

Price test columns: `Test_Week` (1 to 5), `Product_ID`, `Test_Price_USD`, and `Avg_Daily_Units` (average units sold per day that week).

## 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. |

# 6 Terms You Need Today

TERM 1

### `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

TERM 2

### `Unit Margin` and `Gross Profit`

**What:** price minus unit cost, and unit margin times units sold

**Why:** the measure a pricing decision should maximize

TERM 3

### `Price Simulation`

**What:** computing units and profit for several possible prices before choosing one

**Why:** test a decision on paper, not on real customers

TERM 4

### `Days of Stock`

**What:** inventory divided by daily sales

**Why:** shows which products will not sell out in time

TERM 5

### `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

TERM 6

### `Salvage Value`

**What:** what leftover stock is still worth after the season

**Why:** decides how costly it is to be left with unsold units

# How It Works

7.1

## Price elasticity

**Remember:** elasticity = % change in units / % change in price; estimate it from several observations.

7.2

## Why logs

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

7.3

## Simulating a price change

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

7.4

## Stock and time

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

7.5

## Leftover stock has a cost

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

# 7.1 Price elasticity: On Real Data

**SEE IT ON REAL DATA**

| 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` |

**Remember:** elasticity = % change in units / % change in price; estimate it from several observations.

# Estimate Elasticity from the Price Test

YourName_S43_demo_1.py — Input → Process → Output

```
# 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

- `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.

## 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.

# Quick Check 1: `.pct_change()`

NEW TOOL

**`.pct_change()`**

`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.

YOUR TASK

## Fill in the Blank

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.

# Simulate Profit for Several Price Changes

YourName_S43_demo_2.py — Input → Process → Output

```python
# 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

- `elasticity = -1.51` uses the value measured in Micro-Example 1 for every product.
- `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%`.

## READ THE OUTPUT

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

- Profit rises from `-10` % up to `+5` %, then falls again at `+10` %: there is a best point in between.

# Quick Check 2: `max(..., key=...)`

NEW TOOL

**`max(..., key=...)`**

`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.

YOUR TASK

## Fill in the Blank

Add one line to part **3. Output**: `print("Best change:", ___(profit_curve, key=profit_curve.get))`

**Check:** the Terminal shows `Best change: +5%`.

# Which Products Will Not Sell Out in 12 Days?

YourName_S43_demo_3.py — Input → Process → Output

```python
# 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

- `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
...
```

- 6 of 12 products would still have stock at the end of the season.

# Quick Check 3: `np.where()`

NEW TOOL

### `np.where()`

`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[]`.

YOUR TASK

### Fill in the Blank

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`.

# Where to Look Things Up

REFERENCE

## NumPy API Reference: numpy.polyfit

https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

REFERENCE

## NumPy API Reference: numpy.where

https://numpy.org/doc/stable/reference/generated/numpy.where.html

REFERENCE

## Phillips: Pricing and Revenue Optimization (Stanford Business Books)

https://www.sup.org/books/title/?id=7234

REFERENCE

## Talluri & van Ryzin: The Theory and Practice of Revenue Management (Springer Standards)

https://link.springer.com/book/10.1007/b139000

# Three Exercises

EXERCISE 1: GUIDED

## Catalog Profit Curve

**Goal:** Simulate a wider range of price changes

**File:**
`YourName_S43_pricing.py`

**Check:** gross margins from `52.5`% to `55.0`%

EXERCISE 2: APPLIED

## Markdowns That Clear the Stock

**Goal:** Find the price cut each at-risk product needs

**File:**
`YourName_S43_practice.py`

**Check:** `PRD-200` needs the deepest cut (`-11.4`%)

EXERCISE 3: ADVANCED

## The Best Season Price for Each Product

**Goal:** Find each product's best price for the season

**File:**
`YourName_S43_challenge.py`

**Check:** 5 cuts, 4 rises, 3 unchanged

Full steps for all three exercises: sections 11-13 of the course page.