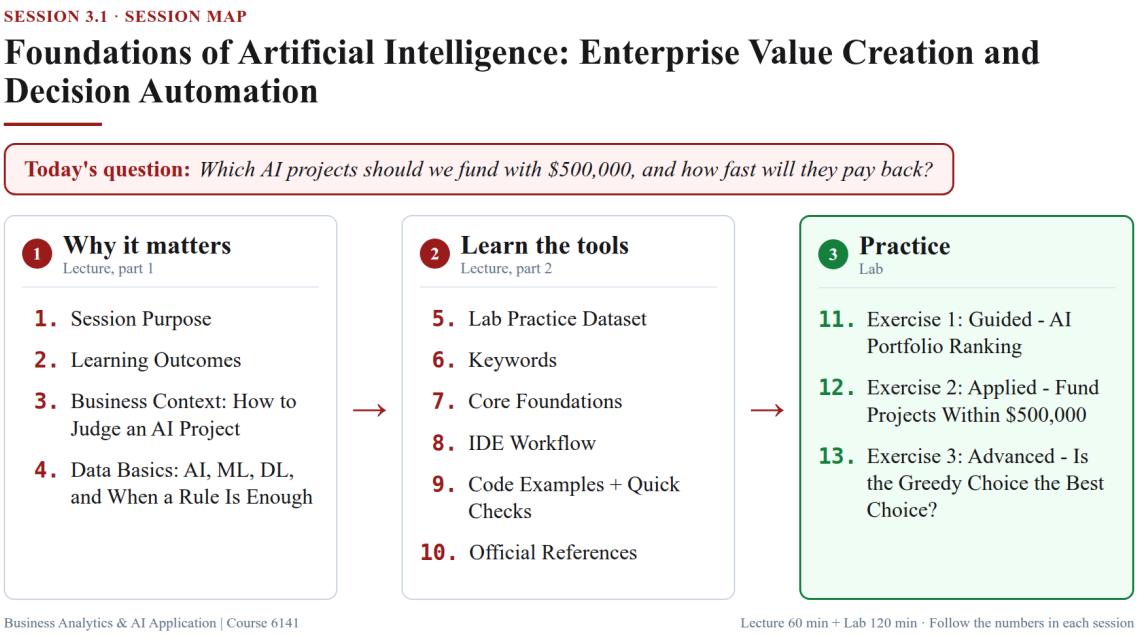
Session 3.1: Foundations of Artificial Intelligence: Enterprise Value Creation and Decision Automation

Course 6141: Business Analytics & AI Application | Reading for Week 9



Schedule: Week 9 | In-person classes (1:2 Delivery Ratio: 60 Min Lecture + 120 Min Lab)

1. Session Purpose

- Five departments have proposed 10 AI projects, costing between $80,000 and $220,000 each
- The total request is $1.5 million, but this year's AI budget is only $500,000
- The COO (Chief Operating Officer) asks: which AI projects should we fund with $500,000, and how fast will they pay back? Today, you learn what AI can and cannot do, and use Python to compare the proposals on the same financial terms

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

1. "Is this project really AI?": tell apart AI (Artificial Intelligence), ML (Machine Learning), and DL (Deep Learning), and decide when a simple rule is enough
2. "Which projects pay back fastest?": calculate the payback period (months until the savings repay the cost) and the 3-year ROI (Return on Investment)

3. “Which projects fit in $500,000?”: allocate a fixed budget with a loop, and fix a `ZeroDivisionError` (dividing by zero)

3. Business Context: How to Judge an AI Project

- Every AI proposal is an investment: money now, savings later. Five numbers make proposals comparable (example: `AI-PRJ-101`, Supply Chain):

Term	Meaning	Example
Implementation cost	one-time cost to build the system	`150,000`
Running cost	yearly cost to keep it working (cloud computing, updates); here 15% of the build cost	`22,500`
Net annual benefit	yearly savings minus the running cost	`294,000 - 22,500 = 271,500`
Payback period	months until the benefit repays the build cost: cost / net benefit x 12	`6.6` months
3-year ROI	(net benefit x 3 - cost) / cost x 100	`443.0`%

- Savings are forecasts, not facts. Two habits keep forecasts honest: include running costs, and compare every project with the same formula.
- Labor hours saved are not the same as money saved: hours only become savings if the freed time is used for other valuable work.

4. Data Basics: AI, ML, DL, and When a Rule Is Enough

- Three nested ideas:

Term	What it is	Business example
AI (Artificial Intelligence)	any computer system that does a task we link to human thinking	a chatbot answering customers
ML (Machine Learning)	AI that **learns a rule from past examples** instead of being told the rule	flagging likely fraud from past fraud cases
DL (Deep Learning)	ML with many-layered neural networks, for images, sound, and text	reading scanned invoices

- Rule or ML? Ask how stable and clear the decision is:

  - **Use a rule** when the policy is fixed and written down: "sales tax is 5% of the price". A rule is cheaper, exact, and easy to explain.
  - **Use ML** when the pattern is complex or changes over time, and you have many past examples with known outcomes: "which invoices look like fraud?"
- ML makes a prediction; a person or a business rule still decides what to do with it.

5. Lab Practice Dataset

- File Link: `data/session_03_01_dataset.csv`
- Audited 10-row AI project proposal dataset (fictional teaching data), one row per proposal:
  - `Project_ID`: Proposal code (e.g., `AI-PRJ-101`).
  - `Department_Name`: Sponsoring department (Supply Chain, Customer Service, Marketing, Finance, Fraud Operations).
  - `Implementation_Cost_USD`: One-time build cost in U.S. Dollars.
  - `Annual_Labor_Hours_Saved_Count`: Staff hours the system is expected to save per year.
  - `Projected_Cost_Savings_USD`: Expected yearly savings in U.S. Dollars.
  - `Payback_Period_Months_Count`: The department's own payback estimate. It ignores running costs, so it is shorter than the payback you calculate today (for example, `6.1` against `6.6` months for `AI-PRJ-101`).

6. Keywords (Core Programming Concepts)

- `Artificial Intelligence (AI)`:
  - What: computer systems that do tasks we link to human thinking
  - Why: the umbrella term in every vendor pitch; ask which kind of AI is inside
- `Machine Learning (ML)`:
  - What: a system that learns a decision rule from past examples
  - Why: handles patterns too complex to write down by hand
- `Deep Learning (DL)`:
  - What: ML with many-layered neural networks
  - Why: needed for images, sound, and free text; costly to build and run
- `Rule-Based System`:
  - What: code that follows rules a person wrote, such as `if amount > 250000:`
  - Why: the right choice when the policy is clear and fixed
- `Payback Period`:
  - What: months until the net benefit repays the build cost
  - Why: shows how fast the money comes back
- `ROI (Return on Investment)`:
  - What: total gain compared with the cost, in percent
  - Why: compares projects of different sizes on one scale

7. Core Programming & Computational Foundations

- **7.1 Rule or learning: two ways to make a decision**
  - *The idea:* A computer can decide in two ways. In a **rule-based** system, a person writes the rule, such as "flag every invoice above $250,000". In a **machine learning** system, the computer finds the rule itself by trying many candidate rules on past examples with known answers and keeping the one that makes the fewest mistakes. Machine learning is useful when the right rule is hard to see by eye; a rule is better when the policy is clear and fixed.
  - *See it on real data:* the 10 past invoices in Micro-Example 2 (5 honest, 5 fraud).

Approach	The rule	Mistakes on the 10 invoices
Rule-based	a person chose "above 250,000"	`1` (the honest `260000` invoice is flagged)
Learned	the search tried every amount as a threshold and found "300,000 or more"	`0`

  - *Common mistake:* treating zero mistakes on past data as a promise. A rule fitted to old examples can miss tomorrow's new pattern. Session 3.3 tests models on data they have not seen.
  - *More examples:*

Decision	Rule or ML?	Why
Add 5% sales tax to every order	rule	the policy is written down and never guesses
Approve expenses under $500 automatically	rule	a clear, fixed company policy
Which invoices look like fraud?	ML	the pattern is complex and changes as fraudsters adapt
Which customers will cancel next month?	ML	many signals combine in ways no one can write down
Read the amount on a scanned invoice	ML (deep learning)	images need many-layered models

  - *Remember:* clear, fixed policy: write a rule; complex pattern with many past answers: learn it.
- **7.2 Payback and ROI: comparing investments on the same terms**
  - *The idea:* Every proposal is money now for savings later. To compare proposals fairly, apply the same formulas to all of them, including the yearly running cost (here 15% of the build cost).

Measure	Formula	`AI-PRJ-101`
Net annual benefit	savings - 0.15 x cost	`294000 - 22500 = 271500`
Payback (months)	cost / net benefit x 12	`150000 / 271500 x 12 = 6.6`

Measure	Formula	AI-PRJ-101
3-year ROI (%)	(net benefit x 3 - cost) / cost x 100	443.0

  - *Common mistake:* ignoring the running cost. The department's own estimate for AI-PRJ-101 is 6.1 months, which is 150000 / 294000 x 12: it leaves out the $22,500 a year needed to keep the system working.
  - *More examples:*

Project	Cost	Net benefit	Payback	3-year ROI
AI-PRJ-110 (Fraud Operations)	190000	515500	4.4 months	713.9%
AI-PRJ-102 (Customer Service)	80000	143000	6.7 months	436.2%
AI-PRJ-103 (Marketing)	120000	178000	8.1 months	345.0%
AI-PRJ-104 (Finance)	200000	362000	6.6 months	443.0%

    The cheapest project (AI-PRJ-102) is not the best return, and the most expensive one (AI-PRJ-110) pays back fastest: size alone says little.
  - *Remember:* same formula for every project, running costs included.

- **7.3 Division by zero: an error in Python, a silent `inf` in pandas**
  - *The idea:* Dividing by zero has no answer. Python and pandas react differently:

Code	Result
150000.0 / 0.0	stops: ZeroDivisionError: float division by zero
pd.Series([4200.0]) / pd.Series([0.0])	inf (infinity), with no error

  - *Why the pandas case is dangerous:* inf counts as larger than every real number, so a project with zero cost would jump to the top of an "hours per dollar" ranking: sorting [3.0, inf, 1.0] from largest gives [inf, 3.0, 1.0].
  - *The fix:* check before dividing, for example if annual_benefit > 0:; for columns, remove or flag zero values before you divide.
  - *More examples:*

Code	Result	What it tells you
`pd.Series([10.0, 20.0]) / pd.Series([2.0, 0.0])`	`[5.0, inf]`	the second division silently becomes infinity
`pd.Series([0.0]) / pd.Series([0.0])`	`[NaN]`	zero divided by zero is "not a number"
`np.isinf(pd.Series([5.0, np.inf]))`	`[False, True]`	find the infinite values
`150000.0 / 0.0`	`ZeroDivisionError`	plain Python stops instead

  - *Remember:* a zero in the bottom of a fraction must be caught, not ignored.

- **7.4 Choosing under a budget: the greedy method**
  - *The idea:* With more good projects than money, you must choose. The **greedy** method is the simplest plan: sort the projects by value per dollar, go down the list, and fund each project that still fits in the remaining budget.
  - *See it on real data:* sorting by hours saved per $1,000 and funding within $500,000.

Step	Project	Cost	Fits?	Budget used
1	`AI-PRJ-102`	`80000`	yes	`80000`
2	`AI-PRJ-107`	`90000`	yes	`170000`
3	`AI-PRJ-110`	`190000`	yes	`360000`
4	`AI-PRJ-105`	`175000`	no (`535000`)	`360000`
…	…	…	…	…
last	`AI-PRJ-103`	`120000`	yes	`480000`

    Result: 4 projects, `16,100` hours a year.
  - *Common mistake:* assuming greedy is always best. It never looks back, so it can fill the last gap with a weak project. Exercise 3 checks every combination and finds `16,900` hours for $495,000.
  - *More examples:* the result depends on what you sort by.

Greedy order	Funded (`AI-PRJ-...`)	Cost and hours
hours saved per $1,000 (Exercise 2)	`102`, `107`, `110`, `103`	`$480,000`, `16,100` hours
3-year ROI	`110`, `105`, `102`	`$445,000`, `16,100` hours
most hours first	`110`, `105`, `107`	`$455,000`, `16,400` hours
cheapest first	`102`, `107`, `103`, `108`	`$420,000`, `12,300` hours

  - *Remember:* greedy is fast and simple; with few options, check the combinations too.

8. Development Workflow in Google Antigravity IDE

- `[1]` File Explorer: create your script with your name and the session code (for example, `Reacher_S31_demo_1.py`), and check the dataset is in `data/`.

- `[2]` Code Editor: write a few lines, run, repeat. Small steps keep errors small.
- `[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_S31_demo_1.py` (pattern: `YourName_S31_demo_1.py`).
- *Micro-Example 1 (Payback and ROI for Every Proposal):* `YourName_S31_demo_1.py`

```
# 1. Input: load the 10 proposals
import pandas as pd
df = pd.read_csv("data/session_03_01_dataset.csv")

# 2. Process: net benefit, payback, and 3-year ROI
df["Net_Annual_Benefit_USD"] = (
    df["Projected_Cost_Savings_USD"] - df["Implementation_Cost_USD"] * 0.15
)
df["Payback_Months"] = (
    df["Implementation_Cost_USD"] / df["Net_Annual_Benefit_USD"] * 12
).round(1)
df["ROI_3yr_Pct"] = (
    (df["Net_Annual_Benefit_USD"] * 3 - df["Implementation_Cost_USD"])
    / df["Implementation_Cost_USD"] * 100
).round(1)

# 3. Output: the 3 proposals with the highest ROI
top3 = df.sort_values("ROI_3yr_Pct", ascending=False).head(3)
print(top3[["Project_ID", "Department_Name", "Payback_Months", "ROI_3yr_Pct"]])
```

  - *How it works, line by line:*
    - `df["Implementation_Cost_USD"] * 0.15` is the yearly running cost; subtracting it from the projected savings gives the net annual benefit, for all 10 proposals at once.
    - The payback divides the build cost by the net benefit and multiplies by 12 to turn years into months.
    - The 3-year ROI compares 3 years of net benefit with the build cost, in percent (section 7.2).
    - `sort_values("ROI_3yr_Pct", ascending=False).head(3)` keeps the 3 best returns.
  - *Read the output:*

```
   Project_ID   Department_Name  Payback_Months  ROI_3yr_Pct
9  AI-PRJ-110  Fraud Operations             4.4        713.9
4  AI-PRJ-105  Fraud Operations             4.5        705.3
5  AI-PRJ-106      Supply Chain             6.6        445.6
```

    - Both Fraud Operations projects pay back in under 5 months and return about 7 times their cost over 3 years.
    - There is a big gap after them: the third-best return (`445.6`%) is far lower. Fraud Operations saves about $2.8 a year for every $1 of build cost; the other projects save between $1.6 and $2.0.

- *Quick Check 1 (`.rank()`):*
  - New tool: `column.rank(ascending=False)` gives each row its place, 1 for the largest. Equal values share the average place, such as `4.5`. Why: a ranking column is easy to show in a decision memo.
  - Task: add one line at the end of part **2. Process**: `df["ROI_Rank"] = df["ROI_3yr_Pct"].____(ascending=False)`, then add `"ROI_Rank"` to the printed columns.
  - Check: `AI-PRJ-110` has rank `1.0` and `AI-PRJ-105` has rank `2.0`.

- *Micro-Example 2 (A Fixed Rule vs a Rule Learned from Examples):* `YourName_S31_demo_2.py`

```python
# 1. Input: 10 past invoices, checked by auditors (fictional)
import pandas as pd
past = pd.DataFrame({
    "Amount_USD": [120000, 180000, 210000, 240000, 260000,
                   300000, 330000, 410000, 450000, 520000],
    "Was_Fraud": [False] * 5 + [True] * 5,
})

# 2. Process: a rule a person chose vs a threshold learned from the data
rule_flags = past["Amount_USD"] > 250000
rule_mistakes = (rule_flags != past["Was_Fraud"]).sum()

best_threshold, best_mistakes = None, len(past)
for threshold in past["Amount_USD"]:
    flags = past["Amount_USD"] >= threshold
    mistakes = (flags != past["Was_Fraud"]).sum()
    if mistakes < best_mistakes:
        best_threshold, best_mistakes = threshold, mistakes

# 3. Output: compare the mistakes
print("Fixed rule (above 250,000) mistakes:", rule_mistakes)
print("Learned threshold:", best_threshold, "mistakes:", best_mistakes)
```

  - *How it works, line by line:*
    - `pd.DataFrame({...})` builds 10 fictional past invoices with known answers. `[False] * 5 + [True] * 5` makes a list of 5 `False` followed by 5 `True`.
    - `past["Amount_USD"] > 250000` applies the person's rule and gives one `True` / `False` per invoice.
    - `(rule_flags != past["Was_Fraud"]).sum()` counts the rows where the rule and the auditors disagree: the mistakes.

    - The `for` loop tries each invoice amount as a candidate threshold, counts its mistakes, and keeps the threshold with the fewest. This search is the core idea of machine learning (section 7.1).
    - `best_threshold, best_mistakes = threshold, mistakes` updates two variables in one line.

  – *Read the output:*

```
Fixed rule (above 250,000) mistakes: 1
Learned threshold: 300000 mistakes: 0
```

    - The person's rule flags the honest `260000` invoice: 1 mistake.
    - The search found that "300,000 or more" separates honest and fraudulent invoices perfectly in these examples.
    - With only 10 examples, this is an illustration of learning, not a rule to deploy.

- *Quick Check 2 (`.mean()` of True/False):*

  – New tool: `.mean()` on a `True` / `False` column gives the share of `True` values, because `True` counts as 1. Why: tells a manager what share of invoices a rule would send to review.
  – Task: add one line to part **3. Output**: `print("Share flagged by the fixed rule:", rule_flags.____())`
  – Check: the Terminal shows `Share flagged by the fixed rule: 0.6`, so 6 of 10 invoices would go to review.

- *Micro-Example 3 (A Shortlist That Does Not Fit the Budget):* `YourName_S31_demo_3.py`

```python
# 1. Input: load the proposals and add the payback period
import pandas as pd
df = pd.read_csv("data/session_03_01_dataset.csv")
df["Net_Annual_Benefit_USD"] = (
    df["Projected_Cost_Savings_USD"] - df["Implementation_Cost_USD"] * 0.15
)
df["Payback_Months"] = (
    df["Implementation_Cost_USD"] / df["Net_Annual_Benefit_USD"] * 12
).round(1)

# 2. Process: fast payback AND many hours saved
shortlist = df[
    (df["Payback_Months"] <= 7.0)
    & (df["Annual_Labor_Hours_Saved_Count"] >= 4000)
]
shortlist_cost = shortlist["Implementation_Cost_USD"].sum()

# 3. Output: compare the shortlist with the budget
print(shortlist[["Project_ID", "Department_Name", "Payback_Months"]])
print(f"Shortlist cost: ${shortlist_cost:,.0f} against a budget of $500,000")
```

  – *How it works, line by line:*

    - Part 1 recomputes the net benefit and payback, as in Micro-Example 1.
    - The two conditions keep projects that pay back within 7 months **and** save at least 4,000 hours a year; each condition is in its own brackets, joined with `&` (Session 1.4).
    - `shortlist["Implementation_Cost_USD"].sum()` adds up what the shortlist would cost.
    - The f-string `${shortlist_cost:,.0f}` prints the total with thousands separators.
  - *Read the output:*

```
   Project_ID   Department_Name  Payback_Months
0  AI-PRJ-101      Supply Chain             6.6
3  AI-PRJ-104           Finance             6.6
4  AI-PRJ-105  Fraud Operations             4.5
5  AI-PRJ-106      Supply Chain             6.6
8  AI-PRJ-109           Finance             6.7
9  AI-PRJ-110  Fraud Operations             4.4
Shortlist cost: $1,095,000 against a budget of $500,000
```

    - 6 projects pass both rules, but together they cost more than twice the budget.
    - Filtering tells you which projects are good; it does not tell you which ones to fund. That needs a budget method (section 7.4).

- *Quick Check 3 (`.isin()`):*
  - New tool: `column.isin([...])` is `True` when the value is in the list. Why: answers questions about a few named groups, such as "show only Finance and Fraud Operations".
  - Task: add one line to part **3.** **Output**: `print(len(df[df["Department_Name"].____([ "Finance", "Fraud Operations"])]))`
  - Check: the Terminal shows `4`.

10. Official Python Documentation & References

- Scikit-Learn: Machine Learning in Python Official Overview.
- Russell & Norvig: Artificial Intelligence: A Modern Approach (Reference Standards).
- Agrawal, Gans, Goldfarb: Prediction Machines: The Simple Economics of AI (Harvard Business Review Press).
- Python Library: `itertools.combinations`.

11. Exercise 1: Guided Practice - AI Portfolio Ranking

- *Step 1: Open Script:* In Google Antigravity IDE, create a new script named `YourName_S31_portfolio.py`, replacing `YourName` with your own name (for example, `Reacher_S31_portfolio.py`).
- *Step 2: Rank the Portfolio and Compare Departments:* Type the following code:

```
# 1. Input: load the 10 proposals
import pandas as pd
df = pd.read_csv("data/session_03_01_dataset.csv")

# 2. Process: net benefit and 3-year ROI, then the average by department
```

```
df["Net_Annual_Benefit_USD"] = (
    df["Projected_Cost_Savings_USD"] - df["Implementation_Cost_USD"] * 0.15
)
df["ROI_3yr_Pct"] = (
    (df["Net_Annual_Benefit_USD"] * 3 - df["Implementation_Cost_USD"])
    / df["Implementation_Cost_USD"] * 100
).round(1)
by_department = (
    df.groupby("Department_Name")["ROI_3yr_Pct"].mean().round(1)
    .sort_values(ascending=False)
)

# 3. Output: the full ranking and the department view
ranked = df.sort_values("ROI_3yr_Pct", ascending=False)
print(ranked[["Project_ID", "Department_Name", "ROI_3yr_Pct"]])
print("\nAverage 3-year ROI by department:\n", by_department)
```

- *Step 3: Interpret:* In two comment lines, explain why Fraud Operations ranks first, and which numbers you would ask the Marketing team to double-check.
- *Step 4: Execute in Terminal:* Run `python YourName_S31_portfolio.py` in Panel `[4]` and confirm return code `0`.
- *Success Check:* department averages are Fraud Operations `709.6`, Supply Chain `444.3`, Finance `440.2`, Customer Service `429.0`, and Marketing `342.3`. Fraud Operations saves the most money per dollar spent (about $2.8 of yearly savings per $1 of build cost).

12. Exercise 2: Applied Practice - Fund Projects Within $500,000

- *Coding Challenge:* In `YourName_S31_practice.py`, fund projects in order of hours saved per dollar until the $500,000 budget runs out, then check a new proposal's payback safely.
- *Task 1 (Greedy Allocation):*
  - Add `Hours_per_1000_USD` = `Annual_Labor_Hours_Saved_Count` / `Implementation_Cost_USD` x 1000, and sort by it from highest to lowest.
  - Start with `budget = 500000`, `spent = 0`, and `funded = []`.
  - Loop over the sorted rows with `for project, cost in zip(df["Project_ID"], df["Implementation_Cost_USD"]):`. If `spent + cost <= budget`, add the project to `funded` and add `cost` to `spent`.
  - `zip(a, b)` walks through two columns side by side, giving one pair of values per row.
  - Print the funded projects, the amount spent, and the total hours saved by the funded projects.
- *Task 2 (Read the ZeroDivisionError & AI Diagnosis):* Write a function `payback_months(cost, annual_benefit)` that returns `cost / annual_benefit * 12`. Call it for a new proposal with no proven savings yet: `payback_months(50000.0, 0.0)`. The script stops with `ZeroDivisionError: float division by zero`. Capture the traceback with `Win + Shift + S`, paste it into Panel `[3]` (AI Agent Chat) with `Ctrl + V`, and ask why.

- *Task 3 (Safe Division):* Change the function: if `annual_benefit <= 0`, return `None`; otherwise return the payback. When the result is `None`, print `"No payback: no proven savings"`.
- *Success Check:* the loop funds `AI-PRJ-102`, `AI-PRJ-107`, `AI-PRJ-110`, and `AI-PRJ-103` for `$480,000` and `16,100` hours a year; `payback_months(50000.0, 0.0)` now prints `No payback: no proven savings`; return code `0`.

13. Exercise 3: Advanced Practice - Is the Greedy Choice the Best Choice?

- *Goal:* Test the simple "most hours per dollar first" method from Exercise 2 against every possible combination of projects, and tell the COO whether the simple method left value on the table.
- *What you need:*
  - File: `data/session_03_01_dataset.csv` (described in **5. Lab Practice Dataset**).
  - Script name: `YourName_S31_challenge.py`.
  - Your Exercise 2 result (`16,100` hours for `$480,000`).
  - Tools you already know: lists, `for` loops, `if`, `.sum()`, `.isin()`, and f-strings.
- *Requirements (follow them in order):*
  1. List every possible group of projects: all groups of 1 project, of 2 projects, and so on, up to all 10.
  2. For each group, add up the build cost. Skip groups that cost more than `$500,000`.
  3. For the groups that fit, add up `Annual_Labor_Hours_Saved_Count`, and keep the group with the most hours. If two groups tie on hours, keep the cheaper one.
  4. Count how many groups you checked in total and how many fit the budget.
  5. Print the best group's project IDs, cost, and hours, then the difference in hours compared with the greedy result.
  6. Print a two-line memo for the COO: what to fund, and why the simple method was not enough.
- *New tools for this exercise:*
  - `from itertools import combinations` loads a tool that lists groups. `combinations(["A", "B", "C"], 2)` gives `("A", "B")`, `("A", "C")`, and `("B", "C")`.
  - `for size in range(1, 11):` repeats with `size` = 1, 2, …, 10; inside it, `for group in combinations(project_ids, size):` walks through every group of that size.
  - `list(group)` turns a group into a list, so you can use `df[df["Project_ID"].isin(list(group))]` to select its rows.
- *Hints:*
  - Make `project_ids = list(df["Project_ID"])` first.
  - Keep the best result in three variables, `best_hours`, `best_cost`, and `best_group`, and update them inside the loop.
  - With 10 projects there are `1,023` groups, so the loop finishes in a few seconds.
- *Self-Check (expected results):*
  - Groups checked: `1023`; groups that fit the budget: `163`.
  - Best group: `AI-PRJ-101`, `AI-PRJ-102`, `AI-PRJ-105`, `AI-PRJ-107`, costing `$495,000` and saving `16,900` hours a year.

– That is `800` more hours than the greedy result. Business meaning: “most hours per dollar first” picked the cheap Marketing project `AI-PRJ-103` to fill the last gap, while a different mix used the budget better. With few options, checking every combination is cheap and finds the true best plan.