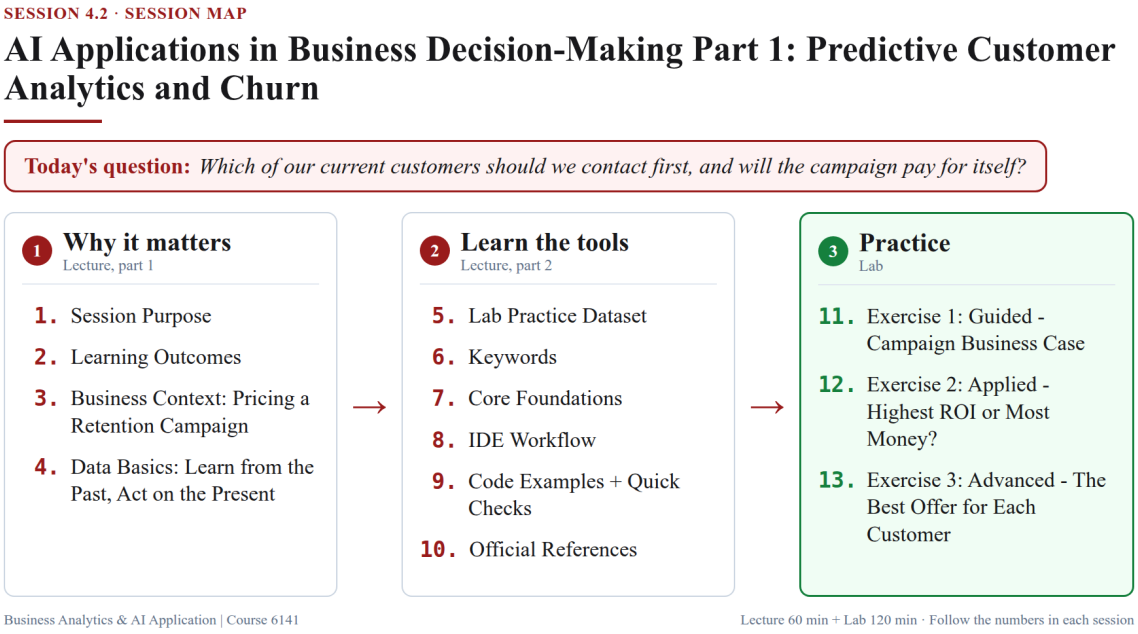
Session 4.2: AI Applications in Business Decision-Making Part 1: Predictive Customer Analytics and Churn

Course 6141: Business Analytics & AI Application | Reading for Week 14



Schedule: Week 14 | In-person classes (1:2 Delivery Ratio: 60 Min Lecture + 120 Min Lab)

1. Session Purpose

- A subscription telecom company loses customers every month to churn (customers cancelling their contract)
- Its retention team can call customers or assign them an account manager, but every contact costs money, and calling customers who would stay anyway wastes it
- The head of customer retention asks: which of our current customers should we contact first, and will the campaign pay for itself? Today, you turn churn predictions into revenue at risk and a campaign plan with a measurable return

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

1. "Who is likely to leave?": train a churn model on past customers and score current customers with a churn probability
2. "How much money is at stake?": calculate revenue at risk (churn probability x yearly customer value) and rank customers by it
3. "Will the campaign pay off?": calculate a campaign's expected net benefit and ROI (Return on Investment), choose who to target, and fix a `ValueError` caused by missing feature columns

3. Business Context: Pricing a Retention Campaign

- Four ideas, one example (current customer `ACT-101`):

Term	Meaning	Example
Churn probability	the model's estimate that this customer leaves within 60 days	`0.876`
Annual value	monthly fee x 12: one year of revenue from this customer	`$2,399.88`
Revenue at risk	churn probability x annual value: the revenue we expect to lose	`$2,102.29`
Expected saved revenue	revenue at risk x success rate of the campaign	`$2,102.29 x 0.40 = $840.92`

- CLV (Customer Lifetime Value) is the total value of a customer over the whole relationship, often several years. Today we use one year of value: simpler, and a cautious basis for spending decisions.
- A campaign is worth it for a customer when expected saved revenue is larger than the contact cost. Contacting a customer who would stay anyway, or whose value is small, wastes budget.
- Net benefit vs ROI: net benefit = saved revenue - cost, in dollars; ROI = net benefit / cost x 100, in percent. The plan with the highest ROI is not always the plan that earns the most money.

4. Data Basics: Learn from the Past, Act on the Present

- Two groups of customers, two files:

File	Customers	Has the answer (`Churned`)?	Used to
History	14 past customers	yes	train the model
Active	10 current customers	no, not yet known	score and target

- A retention campaign only makes sense for customers who have not left yet. So the model learns from the history and predicts for the active customers.
- What drives churn in our history: after scaling, the model's coefficients are `-0.718` for tenure (longer customers stay), `+0.897` for support calls, and `+0.878` for usage drop (more calls and falling usage signal leaving).

5. Lab Practice Dataset

- Two files (fictional teaching data):
  - History: `data/session_04_02_dataset.csv`, 14 past customers with their outcome; 5 of them churned.
  - Active: `data/session_04_02_active.csv`, 10 current customers with the same columns except the outcome.
- Columns:
  - `Subscriber_ID`: Customer code (`SUB-...` in the history, `ACT-...` for active customers).
  - `Tenure_Months_Count`: Months as a customer.
  - `Monthly_Subscription_Fee_USD`: Monthly fee in U.S. Dollars.
  - `Customer_Support_Calls_Count`: Calls to customer support in the last 90 days.
  - `Usage_Drop_Last_30_Days_Pct`: How much the customer's use of the service fell in the last 30 days, in percent.
  - `Churned_Within_60_Days_Binary` (history only): `1` = left within 60 days, `0` = stayed.

6. Keywords (Core Programming Concepts)

- `Churn`:
  - What: customers ending their relationship with the company
  - Why: in subscription businesses, keeping a customer is usually cheaper than winning a new one
- `Churn Probability`:
  - What: the model's estimate, from 0 to 1, that a customer will leave
  - Why: ranks customers by risk instead of treating everyone the same
- `Revenue at Risk`:
  - What: churn probability x annual value
  - Why: turns a probability into dollars a manager can budget against
- `Customer Lifetime Value (CLV)`:
  - What: the total value of a customer over the whole relationship
  - Why: decides how much a customer is worth saving
- `Net Benefit` and `ROI`:
  - What: saved revenue minus cost (dollars), and net benefit divided by cost (percent)
  - Why: the two numbers a finance team asks for before approving a campaign
- `Targeting Threshold`:
  - What: the minimum churn probability for a customer to be contacted
  - Why: sets the size and cost of the campaign

7. Core Programming & Computational Foundations

- **7.1 Learn from the past, predict the present**
  - *The idea:* A churn model needs examples with known outcomes, so it learns from past customers whose story has ended: they either left or stayed. It then scores today's customers, whose outcome is still unknown. Two tables, one model: `fit` on the history, `predict_proba` on the active customers.
  - *See it on real data:* how the 14 past customers differ.

Average of past customers	Stayed (9)	Left (5)
Months as a customer	`36.9`	`6.8`
Support calls (last 90 days)	`1.1`	`4.6`
Usage drop (last 30 days)	`9.4%`	`46.0%`

    Customers who left were newer, called support more, and had stopped using the service. The model turns these patterns into a probability for each current customer.
  - *Common mistake:* scoring the history instead of the active customers. The history has already happened; only current customers can still be saved.
  - *More examples:*

Code	Result	What it tells you
`history["Churned_Within_60_Days_Binary"].mean()`	`0.357`	5 of 14 past customers left
`model.score(history[features], history["Churned_Within_60_Days_Binary"])`	`0.929`	13 of 14 correct on the training rows
`active.shape`	`(10, 5)`	10 current customers, no answer column
`model.named_steps["classify"].coef_`	tenure `-0.718`, calls `0.897`, usage drop `0.878`	the signals that push toward leaving

  - *Remember:* fit on the past, score the present.

- **7.2 Expected value: turning a probability into dollars**
  - *The idea:* A probability alone does not tell a manager how much is at stake. Multiply it by what the customer is worth, and you get the **expected** loss: what you would lose on average across many customers like this one. For one customer, the loss is either all or nothing; across a whole portfolio, the expected values add up reliably.

Customer	Churn probability	Annual value	Revenue at risk
`ACT-101`	`0.876`	`$2,399.88`	`$2,102.29`
`ACT-105`	`0.941`	`$1,199.88`	`$1,129.09`

Customer	Churn probability	Annual value	Revenue at risk
ACT-110	0.448	$2,399.88	$1,075.15
ACT-102	0.026	$1,799.88	$46.80

  - *What the table shows:* ACT-105 is the most likely to leave, but ACT-101 has more money at risk because it pays twice as much. ACT-110 has less than a 50% chance of leaving, yet almost as much at risk as ACT-105.
  - *Common mistake:* ranking by probability alone. It sends the retention team to the most likely leavers, not to the biggest losses.
  - *More examples:*

Code	Result	What it tells you
active["Revenue_At_Risk_USD"].sum()	$7,659.53	total expected loss
top 3 customers' share of it	63.3%	most of the risk sits with a few customers
active["Annual_Value_USD"].sum()	$16,918.80	all revenue from the 10 customers
revenue at risk / all revenue	45.3%	almost half of this group's revenue is in danger
active["Churn_Probability"].mean()	0.461	the average customer here is close to a coin flip

  - *Remember:* revenue at risk = probability x value; rank by dollars.

- **7.3 The economics of a campaign: net benefit and ROI**
  - *The idea:* Contacting a customer costs money, and it does not always work. For each customer, the expected saved revenue is churn probability x success rate x annual value. The campaign pays off for that customer only if this is larger than the contact cost.
  - *See it on real data:* a call costs $75 and keeps 40% of customers who would leave.

Customer	Expected saved revenue	Minus cost	Net
ACT-101	0.876 x 0.40 x 2399.88 = 840.92	- 75	+765.92
ACT-110	430.06	- 75	+355.06
ACT-104	154.07	- 75	+79.07

  - *Two summary numbers:* net benefit (saved revenue - cost, in dollars) and ROI (net benefit / cost x 100, in percent). A small, targeted campaign often has the highest ROI; a broader one can earn more dollars. Exercise 2 shows both on our data.
  - *More examples:*

Customer	Call net (cost $75, 40% success)	VIP net (cost $250, 60% success)	Verdict
`ACT-101`	`+765.92`	`+1011.38`	VIP pays more
`ACT-104`	`+79.07`	`-18.90`	only the call pays
`ACT-107`	`+132.34`	`+61.01`	the call pays more
`ACT-102`	`-56.28`	`-221.92`	no offer pays: contact costs more than it saves

  - *Remember:* contact a customer only when expected savings exceed the contact cost.

- **7.4 Matching the offer to the customer**
  - *The idea:* Not every customer deserves the same offer. A cheap offer (a call) suits customers with modest value or moderate risk; an expensive offer (a personal account manager) pays only for customers who are both valuable and very likely to leave. Rules applied in order, from cheapest to most valuable, express this.
  - *See it on real data:* Micro-Example 3's rules.

Rule (applied in this order)	Customers
everyone starts with "Newsletter"	all 10
probability `>= 0.4`: "Retention_Call"	overwrites 6
probability `>= 0.7` **and** value `>= 1500`: "VIP_Account_Manager"	overwrites 2: `ACT-101`, `ACT-108`

    `ACT-105` has the highest risk (`0.941`) but stays with a call: its annual value, `$1,199.88`, is below the VIP limit.
  - *Common mistake:* applying the rules in the wrong order. If the VIP rule ran first, the call rule would then overwrite it, and nobody would get a VIP manager.
  - *More examples:*

Rule change	Result	What it tells you
VIP limit lowered to `$1,000`	VIP: `ACT-101`, `ACT-105`, `ACT-108`	`ACT-105` now qualifies
VIP probability raised to `0.8`	VIP: `ACT-101`, `ACT-108`	no change for this group
call threshold lowered to `0.3`	`6` customers at or above it	the same 6 as at `0.4`: nobody sits between 0.3 and 0.4

  - *Remember:* set a default, then overwrite from the cheapest offer to the most valuable.

8. Development Workflow in Google Antigravity IDE

- `[1]` File Explorer: create your script with your name and the session code (for example, `Reacher_S42_demo_1.py`), and check both files are 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_S42_demo_1.py` (pattern: `YourName_S42_demo_1.py`).
- *Micro-Example 1 (Learn from the Past, Score the Present):* `YourName_S42_demo_1.py`

```python
# 1. Input: past customers (with outcomes) and current customers
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
history = pd.read_csv("data/session_04_02_dataset.csv")
active = pd.read_csv("data/session_04_02_active.csv")
features = ["Tenure_Months_Count", "Customer_Support_Calls_Count",
            "Usage_Drop_Last_30_Days_Pct"]

# 2. Process: train on history, then score the active customers
model = Pipeline([
    ("scale", StandardScaler()),
    ("classify", LogisticRegression()),
])
model.fit(history[features], history["Churned_Within_60_Days_Binary"])
active["Churn_Probability"] = model.predict_proba(active[features])[:, 1].round(3)

# 3. Output: the active customers, most at risk first
print(active.sort_values("Churn_Probability", ascending=False)[
    ["Subscriber_ID", "Tenure_Months_Count", "Churn_Probability"]
])
```

  - *How it works, line by line:*
    - Two files are loaded: `history` (past customers, with the answer) and `active` (current customers, without it).
    - The `Pipeline` scales the 3 features and fits a logistic regression on the history (section 7.1).
    - `model.predict_proba(active[features])[:, 1]` scores the current customers: the second column is the probability of leaving.
    - `.sort_values(..., ascending=False)` lists the riskiest customers first.
  - *Read the output:*

```
  Subscriber_ID  Tenure_Months_Count  Churn_Probability
4       ACT-105                    7              0.941
7       ACT-108                    3              0.900
0       ACT-101                    5              0.876
2       ACT-103                   10              0.675
9       ACT-110                   18              0.448
...
1       ACT-102                   40              0.026
```

  - The top 3 have all been customers for less than 8 months, like the past customers who left.
  - The longest-standing customers (`ACT-102`, 40 months; `ACT-106`, 50 months) are almost certain to stay: under 3%.

- *Quick Check 1 (`named_steps` and `.coef_`):*
  - New tool: `model.named_steps["classify"].coef_` returns the learned coefficients of the logistic regression inside the pipeline. Why: shows which signals push a customer toward leaving.
  - Task: add one line to part **3. Output**: `print(dict(zip(features, model.named_steps["classify"].____[0].round(3).tolist())))`
  - Check: tenure `-0.718`, support calls `0.897`, usage drop `0.878`. A negative coefficient lowers the churn probability; a positive one raises it.

- *Micro-Example 2 (Revenue at Risk):* `YourName_S42_demo_2.py`

```python
# 1. Input: train and score, as in Micro-Example 1
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
history = pd.read_csv("data/session_04_02_dataset.csv")
active = pd.read_csv("data/session_04_02_active.csv")
features = ["Tenure_Months_Count", "Customer_Support_Calls_Count",
            "Usage_Drop_Last_30_Days_Pct"]
model = Pipeline([("scale", StandardScaler()),
                  ("classify", LogisticRegression())])
model.fit(history[features], history["Churned_Within_60_Days_Binary"])
active["Churn_Probability"] = model.predict_proba(active[features])[:, 1].round(3)

# 2. Process: one year of value, and the part of it at risk
active["Annual_Value_USD"] = (active["Monthly_Subscription_Fee_USD"] *
 ↪  12).round(2)
active["Revenue_At_Risk_USD"] = (
    active["Churn_Probability"] * active["Annual_Value_USD"]
).round(2)
ranked = active.sort_values("Revenue_At_Risk_USD", ascending=False)

# 3. Output: the ranking and the total
print(ranked[["Subscriber_ID", "Churn_Probability", "Annual_Value_USD",
```

```
            "Revenue_At_Risk_USD"]])
print(f"Total revenue at risk: ${active['Revenue_At_Risk_USD'].sum():,.2f}")
```

  - *How it works, line by line:*
    - Part 1 repeats the training and scoring of Micro-Example 1.
    - `Monthly_Subscription_Fee_USD * 12` is one year of revenue from each customer.
    - Multiplying by the churn probability gives the revenue at risk (section 7.2).
    - The table is sorted by revenue at risk, and the total is printed with thousands separators.
  - *Read the output:*

```
  Subscriber_ID  Churn_Probability  Annual_Value_USD  Revenue_At_Risk_USD
0       ACT-101              0.876           2399.88              2102.29
7       ACT-108              0.900           1799.88              1619.89
4       ACT-105              0.941           1199.88              1129.09
9       ACT-110              0.448           2399.88              1075.15
...
Total revenue at risk: $7,659.53
```

    - The order differs from Micro-Example 1: `ACT-101` moves to the top because it pays the most, and `ACT-110` jumps to fourth despite its lower probability.
    - Across all 10 customers, about `$7,660` of yearly revenue is expected to be lost if nothing is done.

- *Quick Check 2 (`.cumsum()`):*
  - New tool: `column.cumsum()` gives a running total down the rows. Why: shows how much of the total risk the first few customers carry.
  - Task: add one line at the end of part **2. Process**: `ranked["Running_Total_USD"] = ranked["Revenue_At_Risk_USD"].______().round(2)`, and add `"Running_Total_USD"` to the printed columns.
  - Check: the third row shows `4851.27`: the top 3 customers carry 63% of the total risk.

- *Micro-Example 3 (Assign a Campaign to Each Customer):* `YourName_S42_demo_3.py`

```
# 1. Input: train and score, as in Micro-Example 2
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
history = pd.read_csv("data/session_04_02_dataset.csv")
active = pd.read_csv("data/session_04_02_active.csv")
features = ["Tenure_Months_Count", "Customer_Support_Calls_Count",
            "Usage_Drop_Last_30_Days_Pct"]
model = Pipeline([("scale", StandardScaler()),
                  ("classify", LogisticRegression())])
model.fit(history[features], history["Churned_Within_60_Days_Binary"])
active["Churn_Probability"] = model.predict_proba(active[features])[:, 1].round(3)
```

```
active["Annual_Value_USD"] = (active["Monthly_Subscription_Fee_USD"] *
↪  12).round(2)

# 2. Process: rules from the cheapest to the most valuable campaign
active["Campaign"] = "Newsletter"
active.loc[active["Churn_Probability"] >= 0.4, "Campaign"] = "Retention_Call"
active.loc[
    (active["Churn_Probability"] >= 0.7) & (active["Annual_Value_USD"] >= 1500),
    "Campaign",
] = "VIP_Account_Manager"

# 3. Output: how many customers get each campaign
print(active["Campaign"].value_counts())
print(active[["Subscriber_ID", "Churn_Probability", "Campaign"]])
```

  - *How it works, line by line:*
    - Part 1 scores the customers and computes their annual value, as before.
    - `active["Campaign"] = "Newsletter"` gives everyone the default.
    - The first `.loc[]` line overwrites the default with "Retention_Call" for customers at `0.4` or above.
    - The second `.loc[]` line overwrites again with "VIP_Account_Manager" for customers at `0.7` or above **and** worth `$1,500` or more a year (section 7.4).
    - `value_counts()` counts the customers per campaign.
  - *Read the output:*

```
Campaign
Newsletter            4
Retention_Call        4
VIP_Account_Manager   2
...
4       ACT-105             0.941       Retention_Call
...
7       ACT-108             0.900  VIP_Account_Manager
```

    - 2 customers get a personal account manager, 4 get a call, and 4 only get the newsletter.
    - `ACT-105` gets a call, not a VIP manager, although it is the most likely to leave: it is not valuable enough for the expensive offer.

- *Quick Check 3 (`pd.cut()`):*
  - New tool: `pd.cut(column, bins=[0, 0.4, 0.7, 1.0], labels=["Low", "Medium", "High"])` puts each value into a band. Why: managers think in risk bands, not in decimals.
  - Task: add two lines to part **3. Output**: `active["Risk_Band"] = pd.___(active["Churn_Probability"], bins=[0, 0.4, 0.7, 1.0], labels=["Low", "Medium", "High"])` and `print(active["Risk_Band"].value_counts())`
  - Check: `Low` 4, `Medium` 3, `High` 3.

10. Official Python Documentation & References

- Scikit-Learn Logistic Regression: predict_proba Documentation.
- Pandas API Reference: `pandas.cut`.
- Fader & Hardie: Probability Models for Customer-Base Analysis (Wharton Marketing Standards).

11. Exercise 1: Guided Practice - Campaign Business Case

- *Step 1: Open Script:* In Google Antigravity IDE, create a new script named `YourName_S42_churn.py`, replacing `YourName` with your own name (for example, `Reacher_S42_churn.py`).
- *Step 2: Price a Campaign for Customers at 0.5 or Above:* Type the following code:

```
# 1. Input: train on history, score the active customers
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
history = pd.read_csv("data/session_04_02_dataset.csv")
active = pd.read_csv("data/session_04_02_active.csv")
features = ["Tenure_Months_Count", "Customer_Support_Calls_Count",
            "Usage_Drop_Last_30_Days_Pct"]
model = Pipeline([("scale", StandardScaler()),
                  ("classify", LogisticRegression())])
model.fit(history[features], history["Churned_Within_60_Days_Binary"])
active["Churn_Probability"] = model.predict_proba(active[features])[:, 1].round(3)
active["Annual_Value_USD"] = (active["Monthly_Subscription_Fee_USD"] *
 ↪  12).round(2)

# 2. Process: target, then expected saved revenue, cost, net benefit, ROI
success_rate = 0.40
cost_per_contact = 75.0
targeted = active[active["Churn_Probability"] >= 0.5]
saved = (targeted["Churn_Probability"] * success_rate
         * targeted["Annual_Value_USD"]).sum()
cost = len(targeted) * cost_per_contact
net_benefit = saved - cost
roi_pct = net_benefit / cost * 100

# 3. Output: the business case
print(f"Customers targeted: {len(targeted)}")
print(f"Expected saved revenue: ${saved:,.2f}")
print(f"Campaign cost: ${cost:,.2f}")
print(f"Net benefit: ${net_benefit:,.2f} | ROI: {roi_pct:.1f}%")
```

  - Expected saved revenue multiplies three things per customer: the chance they would leave, the chance the call keeps them, and what they are worth.

- *Step 3: Interpret:* In two comment lines, explain why we multiply by the churn probability instead of counting each targeted customer's full value.
- *Step 4: Execute in Terminal:* Run `python YourName_S42_churn.py` in Panel `[4]` and confirm return code `0`.
- *Success Check:* `4` customers targeted; expected saved revenue `$2,199.68`; cost `$300.00`; net benefit `$1,899.68`; ROI `633.2%`.

12. Exercise 2: Applied Practice - Highest ROI or Most Money?

- *Coding Challenge:* In `YourName_S42_practice.py`, test targeting thresholds from 0.3 to 0.8 and compare the best ROI with the best net benefit.
- *Task 1 (Threshold Loop):* For each threshold in `[0.3, 0.4, 0.5, 0.6, 0.7, 0.8]`, repeat the business case from Exercise 1 and collect `Threshold`, `Targeted`, `Saved_USD`, `Cost_USD`, `Net_Benefit_USD`, and `ROI_Pct` in one DataFrame.
- *Task 2 (Read the ValueError & AI Diagnosis):* Score the active customers with only two features by mistake: `model.predict_proba(active[["Tenure_Months_Count", "Customer_Support_Calls_Count"]])`. The script stops with `ValueError: The feature names should match those that were passed during fit.` Capture the traceback with `Win + Shift + S`, paste it into Panel `[3]` (AI Agent Chat) with `Ctrl + V`, and ask why the model needs the same columns it was trained on.
- *Task 3 (Two Recommendations):* Fix the scoring, then print two lines: the threshold with the highest `ROI_Pct`, and the threshold with the highest `Net_Benefit_USD`. If thresholds tie, report the higher threshold, since it contacts fewer customers.
- *Success Check:* highest ROI: threshold `0.8` (`762.4%`, 3 customers, net `$1,715.51`); highest net benefit: threshold `0.4` (`$2,387.07`, 6 customers, ROI `530.5%`). The broader campaign earns `$671.56` more, at a lower ROI. If the budget allows, net benefit is what adds to profit.

13. Exercise 3: Advanced Practice - The Best Offer for Each Customer

- *Goal:* Stop using one threshold for everyone. For each current customer, choose the offer with the highest expected net value, or no offer at all, and show the retention head how much more this earns than the best threshold plan from Exercise 2.
- *What you need:*
  - Files: `data/session_04_02_dataset.csv` (history) and `data/session_04_02_active.csv` (active customers).
  - Script name: `YourName_S42_challenge.py`.
  - Tools you already know: the trained pipeline and churn probabilities (Micro-Example 1), annual value (Micro-Example 2), `.loc[]` with conditions, `.value_counts()`, and `.sum()`.
- *Requirements (follow them in order):*
  1. Score the 10 active customers and add `Annual_Value_USD`, as in Micro-Example 2.
  2. Two possible offers:
     - **Call:** costs `$75`, keeps 40% of the customers who would otherwise leave
     - **VIP** (an account manager): costs `$250`, keeps 60%

3. For each customer, compute `Call_Net` = churn probability x 0.40 x annual value - 75, and `VIP_Net` = churn probability x 0.60 x annual value - 250, both rounded to 2 decimals.
4. Set `Best_Offer`:
   - `"VIP"` if `VIP_Net` is above 0 and above `Call_Net`
   - otherwise `"Call"` if `Call_Net` is above 0
   - otherwise `"None"`
5. Add `Expected_Net_USD`: the net value of the chosen offer, or `0` for `"None"`.
6. Print each customer's ID, churn probability, annual value, both net values, best offer, and expected net. Then print the count per offer, the total expected net, and the difference from Exercise 2's best net benefit (`$2,387.07`).

- *New tools for this exercise:*
  - Ordered rules with `.loc[]`: set the default first (`active["Best_Offer"] = "None"`), then overwrite with `"Call"`, then with `"VIP"`. The conditions in step 4 must be written so that the last rule wins only when it should.
  - Copying values row by row from another column: `active.loc[mask, "Expected_Net_USD"] = active.loc[mask, "Call_Net"]`.
- *Hints:*
  - Start `Expected_Net_USD` at `0.0` for every customer.
  - Check one customer by hand first, such as `ACT-101`: `0.876 x 0.60 x 2399.88 - 250 = 1011.38`.
- *Self-Check (expected results):*
  - Offers: VIP for `ACT-101`, `ACT-105`, `ACT-108`, and `ACT-110`; Call for `ACT-103`, `ACT-104`, and `ACT-107`; None for `ACT-102`, `ACT-106`, and `ACT-109`.
  - Total expected net value: `$2,951.44`, which is `$564.37` more than the best threshold plan.
  - Business meaning: matching the offer to the customer beats any single threshold. `ACT-104` has only a `0.214` churn probability but a high annual value, so even a call is worth it, while the expensive VIP offer pays only for valuable, high-risk customers such as `ACT-110`.