In [1]:
```
# Step 1: Import required libraries

from pathlib import Path
import pdfplumber
import re
import pandas as pd

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

In [2]:
```
# Step 2: Check if the 4 PDF files exist in the same folder as this notebook

from pathlib import Path

BASE = Path(".")  # current directory

PDF_FILES = [
    BASE / "May statement period.pdf",
    BASE / "June statement period.pdf",
    BASE / "July statement period.pdf",
    BASE / "August statement period.pdf"
]

print("📂 Checking for PDF files in the current directory:\n")
for f in PDF_FILES:
    if f.exists():
        print(f"✅ Found: {f.name}")
    else:
        print(f"⚠️ Not found: {f.name}")
```

```
📂 Checking for PDF files in the current directory:

✅ Found: May statement period.pdf
✅ Found: June statement period.pdf
✅ Found: July statement period.pdf
✅ Found: August statement period.pdf
```

In [3]:
```
# Step 3: Extract text from all PDF files and collect transactions

transactions = []  # list to store extracted transactions

for pdf_path in PDF_FILES:
    print(f"🔍 Reading file: {pdf_path.name}")
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            text = page.extract_text()
            if text:
                lines = text.split("\n")
                for line in lines:
                    # Simple example: capture lines that contain a date pattern (e.
                    if re.search(r"\d{2}/\d{2}/\d{4}", line):
                        transactions.append(line)

print(f"\n✅ Total extracted lines containing transaction-like data: {len(transacti
```

```
🔍 Reading file: May statement period.pdf
🔍 Reading file: June statement period.pdf
🔍 Reading file: July statement period.pdf
🔍 Reading file: August statement period.pdf

✅ Total extracted lines containing transaction-like data: 0
```

In [4]:
```python
# Step 4: Preview raw text from the first page of the first PDF

import itertools

pdf_path = PDF_FILES[0]  # use first PDF (May statement)
print(f"📄 Previewing text from: {pdf_path.name}\n")

with pdfplumber.open(pdf_path) as pdf:
    page = pdf.pages[0]
    text = page.extract_text()
    if text:
        lines = text.split("\n")
        # Display first 30 lines only
        for line in itertools.islice(lines, 30):
            print(line)
    else:
        print("⚠️ No text extracted. The PDF may be image-based (scanned).")
```

```
Previewing text from: May statement period.pdf

ENOCH A-IYEH
CIBC Dividend® Visa* Card
Account number
4505 5300 3195 4066
May statement period
May 14 to June 13, 2025
Your account at a glance
Contact us
Customer Service 1 800 465-4653
Previous balance $0.00
Lost/Stolen 1 800 663-4575
Payments $5,254.46 TTY 1 877 331-3338
Other credits 0.00 Online www.cibc.com
Total credits - $5,254.46 Summary Credit† Cash†
Purchases 5,254.77
Limit $500.00 $500.00
Cash advances 0.00 Available $499.69 $499.00
Interest 0.00 Interest rates
Annual
Fees 0.00
Regular purchases 19.99%
Total charges + $5,254.77
Cash advances 22.99%
New balance = $0.31
Your minimum payment due Cash Back Summary
As at last statement + 38.02
This statement
Current month’s minimum payment $0.31 2% Cash Back
Please pay this amount by Jun 30, 2025
Cash Back
```

In [5]:
```python
# Step 5: Detect and preview candidate transaction lines

import re
import pdfplumber

months = r"(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Sept|Oct|Nov|Dec)"
# Pattern: Month Day (optional comma and year), some description text, amount at th
tx_pattern = re.compile(
    rf"^{months}\s+\d{{1,2}}(?:,?\s*\d{{4}})?\s+(.*?)\s+(-?\$?\d[\d,]*\.\d{{2}})\s*
)

candidates = []

for pdf_path in PDF_FILES:
    with pdfplumber.open(pdf_path) as pdf:
        for pageno, page in enumerate(pdf.pages, start=1):
            text = page.extract_text() or ""
            for i, line in enumerate(text.split("\n"), start=1):
                m = tx_pattern.match(line.strip())
                if m:
                    mon, desc, amt, cr = m.group(1), m.group(2), m.group(3), m.grou
                    candidates.append({
                        "file": pdf_path.name,
```

```
                        "page": pageno,
                        "line_no": i,
                        "raw_line": line.strip(),
                        "month": mon,
                        "desc": desc.strip(),
                        "amount_raw": (amt + (" CR" if cr else "")),
                    })

print(f"✅ Found {len(candidates)} candidate lines.")
print("\n🔎 First 20 candidates:")
for row in candidates[:20]:
    print(f"- [{row['file']} p{row['page']}#{row['line_no']}] {row['raw_line']}")
```

```
✅ Found 34 candidate lines.

🔎 First 20 candidates:
- [May statement period.pdf p2#6] May 16 May 18 PAYMENT THANK YOU/PAIEMENT MERCI 12.
38
- [May statement period.pdf p2#7] May 18 May 19 PAYMENT THANK YOU/PAIEMENT MERCI 4,5
00.00
- [May statement period.pdf p2#8] May 21 May 24 PAYMENT THANK YOU/PAIEMENT MERCI 58.
34
- [May statement period.pdf p2#9] May 22 May 25 PAYMENT THANK YOU/PAIEMENT MERCI 630
.00
- [May statement period.pdf p2#10] Jun 10 Jun 12 PAYMENT THANK YOU/PAIEMENT MERCI 53
.74
- [May statement period.pdf p2#18] May 16 May 17 NORTH GARDEN RESTAURAN WINNIPEG MB
Restaurants 12.38
- [May statement period.pdf p2#23] May 19 May 20 NORTH GARDEN RESTAURAN WINNIPEG MB
Restaurants 12.38
- [May statement period.pdf p2#30] May 29 Jun 01 Ý REAL CDN. SUPERSTORE # WINNIPEG M
B Retail and Grocery 30.90
- [June statement period.pdf p2#6] Jun 13 Jun 16 PAYMENT THANK YOU/PAIEMENT MERCI 0.
31
- [June statement period.pdf p2#7] Jun 17 Jun 19 PAYMENT THANK YOU/PAIEMENT MERCI 20
.00
- [June statement period.pdf p2#8] Jun 21 Jun 23 PAYMENT THANK YOU/PAIEMENT MERCI 50
0.00
- [June statement period.pdf p2#9] Jun 28 Jun 30 PAYMENT THANK YOU/PAIEMENT MERCI 19
.94
- [June statement period.pdf p2#10] Jun 30 Jul 02 PAYMENT THANK YOU/PAIEMENT MERCI 2
5.00
- [June statement period.pdf p2#11] Jul 03 Jul 06 PAYMENT THANK YOU/PAIEMENT MERCI 2
85.00
- [June statement period.pdf p2#12] Jul 04 Jul 06 PAYMENT THANK YOU/PAIEMENT MERCI 2
5.00
- [June statement period.pdf p2#13] Jul 06 Jul 08 PAYMENT THANK YOU/PAIEMENT MERCI 4
8.00
- [June statement period.pdf p2#19] Jun 17 Jun 18 UNITED WAY OF WINNIPEG WINNIPEG MB
Personal and Household Expenses 20.00
- [June statement period.pdf p2#40] Jun 24 Jun 25 CASH ADV/AVANCE DE FONDS-La Havana
Foreign Currency Transactions 484.94
- [July statement period.pdf p2#6] Jul 14 Jul 16 PAYMENT THANK YOU/PAIEMENT MERCI 33
.97
- [July statement period.pdf p2#7] Jul 23 Jul 26 PAYMENT THANK YOU/PAIEMENT MERCI 1,
200.00
```

In [6]:
```python
# Step 6: Convert extracted transactions into a DataFrame

import pandas as pd

# Create DataFrame from candidate transactions
df = pd.DataFrame(candidates)

# Clean amount field
def clean_amount(val):
    val = val.replace("$", "").replace(",", "").replace("CR", "").strip()
    try:
        return float(val)
    except ValueError:
        return None

df["Amount"] = df["amount_raw"].apply(clean_amount)

# Add transaction date column
df["Transaction Date"] = df["month"] + " " + df["desc"].str.extract(r"^(\d{1,2})")[

# Reorder and select useful columns
df = df[["file", "Transaction Date", "desc", "Amount"]]

# Display first few rows
print("✅ Sample transactions table:")
display(df.head(10))
```

✅ Sample transactions table:

	file	Transaction Date	desc	Amount
0	May statement period.pdf	May	May 18 PAYMENT THANK YOU/PAIEMENT MERCI	12.38
1	May statement period.pdf	May	May 19 PAYMENT THANK YOU/PAIEMENT MERCI	4500.00
2	May statement period.pdf	May	May 24 PAYMENT THANK YOU/PAIEMENT MERCI	58.34
3	May statement period.pdf	May	May 25 PAYMENT THANK YOU/PAIEMENT MERCI	630.00
4	May statement period.pdf	Jun	Jun 12 PAYMENT THANK YOU/PAIEMENT MERCI	53.74
5	May statement period.pdf	May	May 17 NORTH GARDEN RESTAURAN WINNIPEG MB Rest...	12.38
6	May statement period.pdf	May	May 20 NORTH GARDEN RESTAURAN WINNIPEG MB Rest...	12.38
7	May statement period.pdf	May	Jun 01 Ý REAL CDN. SUPERSTORE # WINNIPEG MB Re...	30.90
8	June statement period.pdf	Jun	Jun 16 PAYMENT THANK YOU/PAIEMENT MERCI	0.31
9	June statement period.pdf	Jun	Jun 19 PAYMENT THANK YOU/PAIEMENT MERCI	20.00

In [7]:
```
# Step 7: Export the transaction table to Excel and CSV

# Export to Excel
excel_path = "Transactions_Summary.xlsx"
df.to_excel(excel_path, index=False)

# Export to CSV
csv_path = "Transactions_Summary.csv"
df.to_csv(csv_path, index=False)

print("✅ Data exported successfully!")
print(f"📁 Excel file saved as: {excel_path}")
print(f"📁 CSV file saved as: {csv_path}")
```

```
✅ Data exported successfully!
📁 Excel file saved as: Transactions_Summary.xlsx
📁 CSV file saved as: Transactions_Summary.csv
```

In [8]:
```
# Step X: Parse transactions with a template (regex) -> array of dicts, add postdat

import re
from datetime import datetime
import pandas as pd
```

```
YEAR = 2025  # per assignment

# Keyword-based categorizer
CATEGORY_RULES = [
    (r"PAYMENT|PAIEMENT MERCI", "Payment"),
    (r"SUPERSTORE|WALMART|COSTCO|GROC|MART|SUPERMARKET", "Grocery"),
    (r"RESTAURANT|CAFE|COFFEE|DINER|FOOD|PIZZA|SUSHI|MB Restaurants", "Restaurant")
    (r"GAS|SHELL|PETRO|ESSO", "Gas"),
    (r"UBI|E-TRANSFER|INTERAC|WIRE|ADVANCE|CASH ADV", "Transfer/Cash"),
    (r"FEE|CHARGE|INTEREST", "Fees/Interest"),
]

def categorize(desc: str) -> str:
    d = desc.upper()
    for pat, cat in CATEGORY_RULES:
        if re.search(pat, d):
            return cat
    return "Other"

# Template (regex) for lines of form:
# "May 18 PAYMENT THANK YOU/PAIEMENT MERCI 12.38"
# Optional postdate at the start of the line: "Jun 01 Jun 03 SOME MERCHANT 30.90"
line_pattern = re.compile(
    r"""^
        (?P<m1>[A-Za-z]{3})\s(?P<d1>\d{1,2})      # transaction month/day
        (?:\s+(?P<m2>[A-Za-z]{3})\s(?P<d2>\d{1,2}))?  # optional post month/day
        \s+(?P<desc>.+?)                           # description (greedy but will b
        \s+(?P<amount>-?\d{1,3}(?:,\d{3})*(?:\.\d{2})?)  # amount at end
        (?:\s*CR)?\s*$                             # optional CR
    """,
    re.VERBOSE,
)

# Helper to build ISO date
def make_date(mon_str: str, day_str: str) -> str:
    try:
        dt = datetime.strptime(f"{mon_str} {day_str} {YEAR}", "%b %d %Y")
        return dt.strftime("%Y-%m-%d")
    except Exception:
        return None

records = []  # <-- array of dicts per yêu cầu

for pdf_path in PDF_FILES:
    with pdfplumber.open(pdf_path) as pdf:
        for pidx, page in enumerate(pdf.pages, start=1):
            text = page.extract_text() or ""
            for raw_line in text.splitlines():
                raw_line = raw_line.strip()
                m = line_pattern.match(raw_line)
                if not m:
                    continue

                m1, d1 = m.group("m1"), m.group("d1")
                m2, d2 = m.group("m2"), m.group("d2")
                desc = m.group("desc").strip()
```

```
                amt_str = m.group("amount")

                # normalize amount
                amount = float(amt_str.replace(",", ""))

                # dates
                transaction_date = make_date(m1, d1)
                post_date = make_date(m2, d2) if (m2 and d2) else transaction_date

                # category
                category = categorize(desc)

                records.append({
                    "file": pdf_path.name,
                    "page": pidx,
                    "transaction_date": transaction_date,
                    "post_date": post_date,
                    "description": desc,
                    "category": category,
                    "amount": amount,
                    "raw_line": raw_line,  # keep raw for traceability
                })

print(f"✅ Parsed {len(records)} transactions into an array of dicts.")
# Peek first 10
pd.DataFrame(records).head(10)
```

```
✅ Parsed 34 transactions into an array of dicts.
```

Out[8]:

	file	page	transaction_date	post_date	description	category	amount	rav
0	May statement period.pdf	2	2025-05-16	2025-05-18	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	12.38	May 1 18 PAY T PAIE M
1	May statement period.pdf	2	2025-05-18	2025-05-19	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	4500.00	May 1 19 PAY T PAIE M
2	May statement period.pdf	2	2025-05-21	2025-05-24	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	58.34	May 2 24 PAY T PAIE M
3	May statement period.pdf	2	2025-05-22	2025-05-25	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	630.00	May 2 25 PAY T PAIE M
4	May statement period.pdf	2	2025-06-10	2025-06-12	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	53.74	Jun 12 PAY T PAIE M
5	May statement period.pdf	2	2025-05-16	2025-05-17	NORTH GARDEN RESTAURAN WINNIPEG MB Restaurants	Restaurant	12.38	May 1 17 N GA RESTA WINNI
6	May statement period.pdf	2	2025-05-19	2025-05-20	NORTH GARDEN RESTAURAN WINNIPEG MB Restaurants	Restaurant	12.38	May 1 20 N GA RESTA WINNI
7	May statement period.pdf	2	2025-05-29	2025-06-01	Ý REAL CDN. SUPERSTORE # WINNIPEG MB Retail an...	Grocery	30.90	May 2 01 Ý SUPERS # WIN

	file	page	transaction_date	post_date	description	category	amount	rav
8	June statement period.pdf	2	2025-06-13	2025-06-16	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	0.31	Jun 16 PAY T PAIE M
9	June statement period.pdf	2	2025-06-17	2025-06-19	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	20.00	Jun 19 PAY T PAIE M

In [9]:
```
# Save the array of dicts and a CSV table for later use

import json
import pandas as pd

# records is the list[dict] we built
out_json = "Transactions_Structured.json"
out_csv  = "Transactions_Structured.csv"

# JSON (array of dicts)
with open(out_json, "w", encoding="utf-8") as f:
    json.dump(records, f, ensure_ascii=False, indent=2)

# CSV (nice table)
df_export = pd.DataFrame(records)[
    ["file", "page", "transaction_date", "post_date", "description", "category", "a
]
df_export.to_csv(out_csv, index=False)

print("✅ Saved:")
print(" -", out_json)
print(" -", out_csv)
len(records), df_export.shape
```

```
✅ Saved:
 - Transactions_Structured.json
 - Transactions_Structured.csv
```

Out[9]: (34, (34, 7))

In [1]:
```
# ================================
# Question 1.2 – Step 1: Load data and prepare month labels
# ================================

import pandas as pd
from pathlib import Path

# 1) Load data (prefer CSV; fallback to JSON if needed)
csv_path  = Path("Transactions_Structured.csv")
json_path = Path("Transactions_Structured.json")
```

```
if csv_path.exists():
    df = pd.read_csv(csv_path)
    source_used = csv_path.name
elif json_path.exists():
    df = pd.read_json(json_path)
    source_used = json_path.name
else:
    raise FileNotFoundError(
        "Cannot find Transactions_Structured.csv or Transactions_Structured.json in
    )

print(f"✅ Data loaded from: {source_used}")
print(f"Shape: {df.shape}")

# 2) Ensure required columns exist
required_cols = ["transaction_date", "post_date", "description", "category", "amoun
missing = [c for c in required_cols if c not in df.columns]
if missing:
    raise ValueError(f"Missing required columns: {missing}")

# 3) Parse date columns (coerce errors to NaT so we can spot issues)
df["transaction_date"] = pd.to_datetime(df["transaction_date"], errors="coerce")
df["post_date"]        = pd.to_datetime(df["post_date"],        errors="coerce")

# 4) Create friendly month label for filtering, e.g., "May 2025"
#    (You can switch to 'post_date' if your assignment requires post date as the mo
df["month_label"] = df["transaction_date"].dt.strftime("%B %Y")

# 5) Standardize amount to numeric (if it is not already)
df["amount"] = pd.to_numeric(df["amount"], errors="coerce")

# 6) Quick sanity check
print("\n🧭 Unique month labels found:")
print(sorted(df["month_label"].dropna().unique()))

print("\n🔎 Preview (first 5 rows):")
display(df.head(5))
```

```
✅ Data loaded from: Transactions_Structured.csv
Shape: (34, 7)

🧭 Unique month labels found:
['August 2025', 'July 2025', 'June 2025', 'May 2025']

🔎 Preview (first 5 rows):
```

	file	page	transaction_date	post_date	description	category	amount	month_lab
0	May statement period.pdf	2	2025-05-16	2025-05-18	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	12.38	May 202
1	May statement period.pdf	2	2025-05-18	2025-05-19	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	4500.00	May 202
2	May statement period.pdf	2	2025-05-21	2025-05-24	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	58.34	May 202
3	May statement period.pdf	2	2025-05-22	2025-05-25	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	630.00	May 202
4	May statement period.pdf	2	2025-06-10	2025-06-12	PAYMENT THANK YOU/ PAIEMENT MERCI	Payment	53.74	June 202

In [2]:
```
# ================================
# Question 1.2 – Step 2: Month selector + monthly spend report
# ================================
import pandas as pd
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output

# --- Build a sorted month list (chronological) ---
def _month_sort_key(m):
    return pd.to_datetime(m, format="%B %Y", errors="coerce")

month_options = sorted(df["month_label"].dropna().unique(), key=_month_sort_key)

# --- UI widgets ---
month_dd = widgets.Dropdown(
    options=month_options,
    value=month_options[0],
    description="Month:",
    style={"description_width": "80px"},
    layout=widgets.Layout(width="300px")
)
```

```
out = widgets.Output()

def render_month(month_label: str):
    """Filter data by month and render a compact report: KPIs, table, and charts."""
    with out:
        clear_output(wait=True)

        # Filter month
        dfm = df[df["month_label"] == month_label].copy()
        if dfm.empty:
            display(pd.DataFrame({"message":[f"No data for {month_label}"]}))
            return

        # ---- KPIs ----
        total_amt   = dfm["amount"].sum(skipna=True)
        n_txn       = len(dfm)
        min_date    = dfm["transaction_date"].min()
        max_date    = dfm["transaction_date"].max()

        print(f"🗓️ Report for: {month_label}")
        print(f"• Date range: {min_date.date()} → {max_date.date()}")
        print(f"• #Transactions: {n_txn}")
        print(f"• Total spend: ${total_amt:,.2f}")

        # ---- Table (clean columns for viewing) ----
        cols = ["transaction_date", "post_date", "description", "category", "amount
        dft = dfm[cols].sort_values("transaction_date")
        display(dft.reset_index(drop=True))

        # ---- Bar chart by category ----
        cat_sum = (
            dfm.groupby("category", dropna=False)["amount"]
               .sum()
               .sort_values(ascending=False)
        )
        if not cat_sum.empty:
            plt.figure(figsize=(8, 4))
            cat_sum.plot(kind="bar")
            plt.title(f"Total Spend by Category — {month_label}")
            plt.xlabel("Category")
            plt.ylabel("Amount ($)")
            plt.grid(True, axis="y", alpha=0.3)
            plt.tight_layout()
            plt.show()

        # ---- Optional: Daily timeline (uncomment to show) ----
        # daily = dfm.groupby(dfm["transaction_date"].dt.date)["amount"].sum()
        # if not daily.empty:
        #     plt.figure(figsize=(8, 3.5))
        #     daily.plot(marker="o")
        #     plt.title(f"Daily Spend — {month_label}")
        #     plt.xlabel("Date")
        #     plt.ylabel("Amount ($)")
        #     plt.grid(True, alpha=0.3)
        #     plt.tight_layout()
        #     plt.show()
```

```
def on_select_month(change):
    if change["name"] == "value" and change["new"] is not None:
        render_month(change["new"])

month_dd.observe(on_select_month, names="value")

# Render UI
display(month_dd, out)
render_month(month_dd.value)
```

```
Dropdown(description='Month:', layout=Layout(width='300px'), options=('May 2025', 'J
une 2025', 'July 2025', 'A…
Output()
```

In [3]:
```
# ==============================
# Question 1.2 – Step 3: Export & Top N helpers
# ==============================
import ipywidgets as widgets
from IPython.display import display
import pandas as pd

btn_export_csv   = widgets.Button(description="Export month → CSV",  button_style="
btn_export_xlsx  = widgets.Button(description="Export month → Excel", button_style=
topn_slider      = widgets.IntSlider(description="Top N:", value=10, min=5, max=30,
                                     style={"description_width":"60px"}, layout=wid
out2 = widgets.Output()

def _current_month_df():
    # Lấy tháng đang chọn trong dropdown của Step 2
    month_label = month_dd.value
    return df[df["month_label"] == month_label].copy(), month_label

def on_export_csv(_):
    d, mlabel = _current_month_df()
    fn = f"{mlabel.replace(' ', '_')}_report.csv"
    d.to_csv(fn, index=False)
    with out2:
        out2.clear_output(wait=True)
        print(f"✅ Exported CSV: {fn} ({len(d)} rows)")

def on_export_xlsx(_):
    d, mlabel = _current_month_df()
    fn = f"{mlabel.replace(' ', '_')}_report.xlsx"
    with pd.ExcelWriter(fn, engine="xlsxwriter") as writer:
        d.to_excel(writer, index=False, sheet_name="Transactions")
        # thêm pivot theo category cho đẹp
        (d.pivot_table(index="category", values="amount", aggfunc="sum")
           .sort_values("amount", ascending=False)
           .to_excel(writer, sheet_name="ByCategory"))
    with out2:
        out2.clear_output(wait=True)
        print(f"✅ Exported Excel: {fn} (sheet: Transactions, ByCategory)")

def show_topn(_=None):
    d, mlabel = _current_month_df()
```

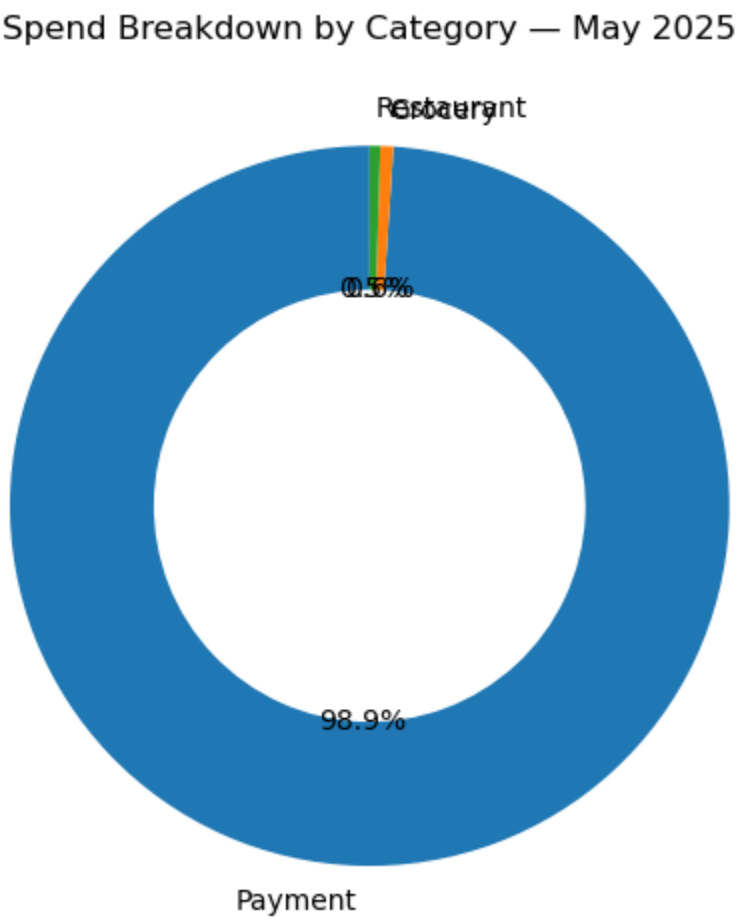
```
    topn = min(topn_slider.value, len(d))
    d2 = d.sort_values("amount", ascending=False).head(topn)[
        ["transaction_date","post_date","description","category","amount"]
    ]
    with out2:
        out2.clear_output(wait=True)
        print(f"🔝 Top {topn} transactions — {mlabel}")
        display(d2.reset_index(drop=True))

btn_export_csv.on_click(on_export_csv)
btn_export_xlsx.on_click(on_export_xlsx)
topn_slider.observe(show_topn, names="value")

display(widgets.HBox([btn_export_csv, btn_export_xlsx, topn_slider]))
display(out2)
show_topn()
```

```
HBox(children=(Button(button_style='success', description='Export month → CSV', styl
e=ButtonStyle()), Button(b…
Output()
```

In [4]:

```
# ================================
# Question 1.2 – Step 5: Donut chart by category (for selected month)
# ================================
import matplotlib.pyplot as plt

def plot_donut(month_label):
    dfm = df[df["month_label"] == month_label].copy()
    if dfm.empty:
        print(f"No data for {month_label}")
        return

    cat_sum = (
        dfm.groupby("category", dropna=False)["amount"]
           .sum()
           .sort_values(ascending=False)
    )

    if cat_sum.empty:
        print(f"No category data for {month_label}")
        return

    # Draw donut
    plt.figure(figsize=(5, 5))
    wedges, texts, autotexts = plt.pie(
        cat_sum,
        labels=cat_sum.index,
        autopct="%1.1f%%",
        startangle=90,
        wedgeprops=dict(width=0.4)
    )
    plt.title(f"Spend Breakdown by Category – {month_label}")
    plt.tight_layout()
    plt.show()

# Tự động vẽ donut cho tháng đang chọn
```

```
plot_donut(month_dd.value)
```



In [16]:
```
# ================================
# Question 1.4 – Step 1
# Prepare data for classification model
# ================================

import pandas as pd
from sklearn.model_selection import train_test_split

# Use the same cleaned dataframe (df) from previous steps
df_c = df.copy()

# Keep only necessary columns
df_c = df_c[["description", "category"]].dropna().reset_index(drop=True)

# Encode category labels
from sklearn.preprocessing import LabelEncoder
le_cat = LabelEncoder()
df_c["category_encoded"] = le_cat.fit_transform(df_c["category"])

# Show mapping for clarity
cat_map = dict(zip(le_cat.classes_, le_cat.transform(le_cat.classes_)))
print("✅ Step 1 completed – Data prepared for classification model.\n")
print("📋 Category encoding map:", cat_map)
print("\n📄 Sample data:")
```

```
display(df_c.head())

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df_c["description"], df_c["category_encoded"], test_size=0.2, random_state=42
)

print(f"\nTraining samples: {len(X_train)} | Test samples: {len(X_test)}")
```

✅ Step 1 completed — Data prepared for classification model.

📑 Category encoding map: {'Gas': np.int64(0), 'Grocery': np.int64(1), 'Other': np.int64(2), 'Payment': np.int64(3), 'Restaurant': np.int64(4), 'Transfer/Cash': np.int64(5)}

🗒️ Sample data:

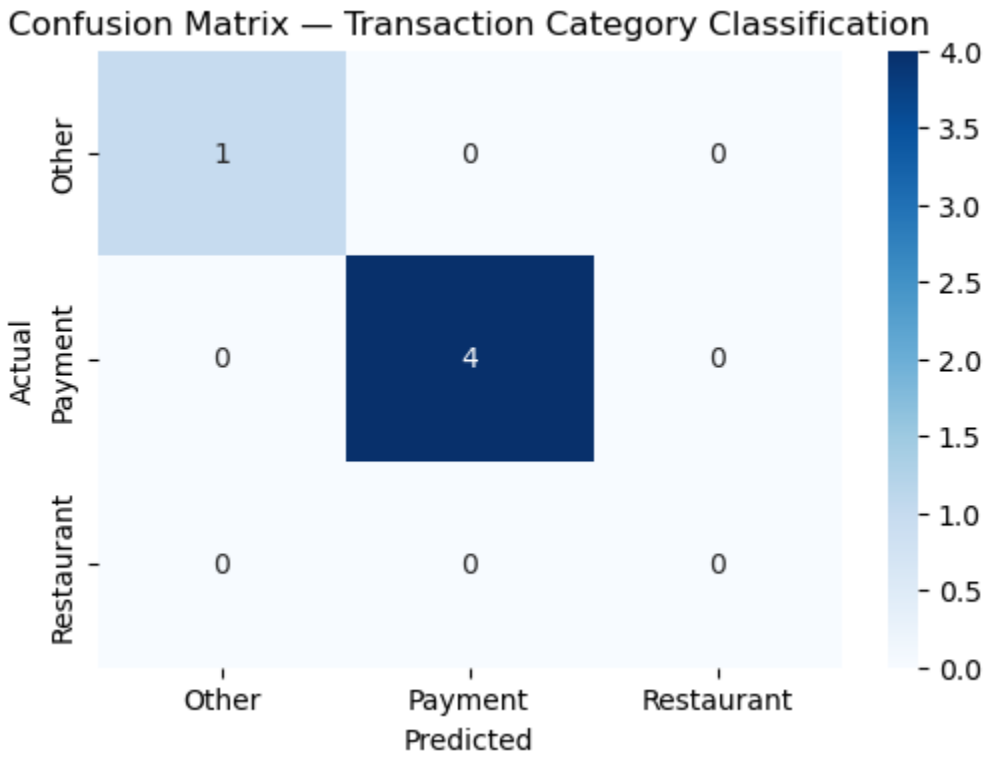
	description	category	category_encoded
0	PAYMENT THANK YOU/PAIEMENT MERCI	Payment	3
1	PAYMENT THANK YOU/PAIEMENT MERCI	Payment	3
2	PAYMENT THANK YOU/PAIEMENT MERCI	Payment	3
3	PAYMENT THANK YOU/PAIEMENT MERCI	Payment	3
4	PAYMENT THANK YOU/PAIEMENT MERCI	Payment	3

Training samples: 27 | Test samples: 7

In [18]:
```
# ================================
# Question 1.4 — Step 2 (Fixed)
# ================================

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 1️⃣ Convert text to numerical features
vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# 2️⃣ Train Logistic Regression model
clf = LogisticRegression(max_iter=500)
clf.fit(X_train_tfidf, y_train)

# 3️⃣ Predict
y_pred = clf.predict(X_test_tfidf)

# 4️⃣ Evaluate performance
acc = accuracy_score(y_test, y_pred)
print(f"✅ Step 2 completed — Classification model trained.")
print(f"📊 Accuracy: {acc:.2f}\n")
```

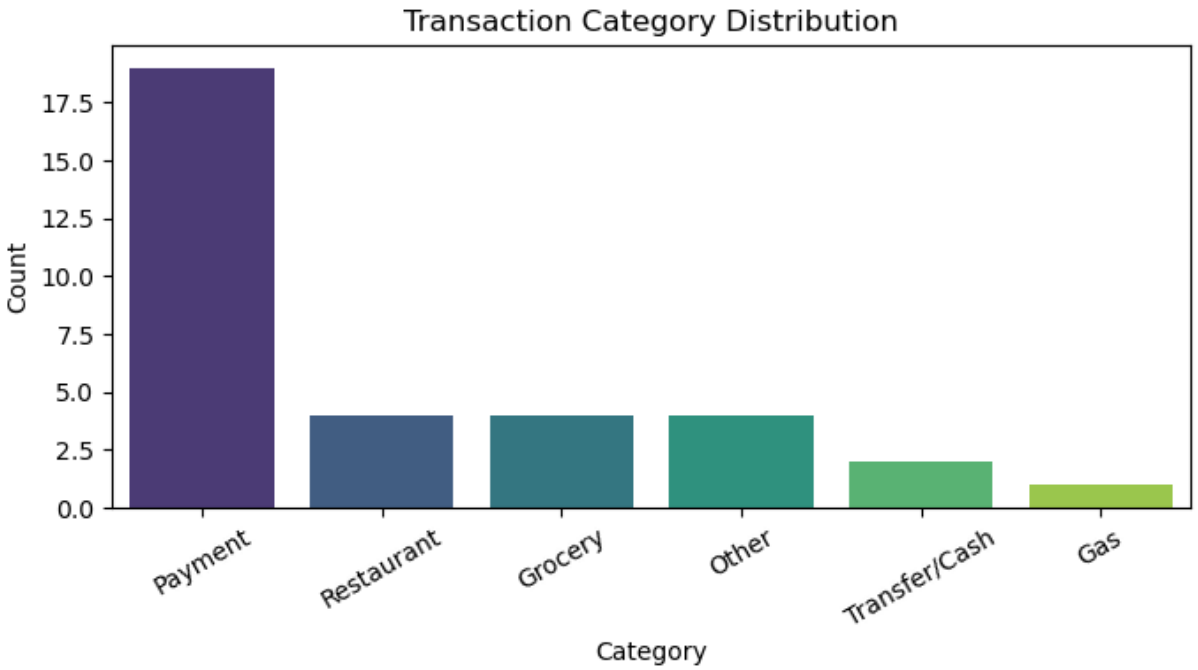
```
# ✅ FIX: only include labels that exist in y_test
unique_labels = np.unique(y_test)
target_names = le_cat.inverse_transform(unique_labels)

print("🔍 Classification Report:\n")
print(classification_report(y_test, y_pred, labels=unique_labels, target_names=targ

# 5️⃣ Confusion Matrix visualization
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred, labels=unique_labels), annot=True, fmt
            xticklabels=target_names, yticklabels=target_names, cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix – Transaction Category Classification")
plt.show()
```

```
✅ Step 2 completed – Classification model trained.
📊 Accuracy: 0.71

🔍 Classification Report:

              precision    recall  f1-score   support

       Other       1.00      1.00      1.00         1
     Payment       1.00      1.00      1.00         4
  Restaurant       0.00      0.00      0.00         2

   micro avg       1.00      0.71      0.83         7
   macro avg       0.67      0.67      0.67         7
weighted avg       0.71      0.71      0.71         7
```



In [19]:
```
# ================================
# Question 1.4 – Step 3
# Prediction demo + Visualization
# ================================

import matplotlib.pyplot as plt
import seaborn as sns

# 1️⃣ Predict category for new transaction descriptions
new_transactions = [
    "WALMART SUPERCENTER PURCHASE",
    "SHELL GAS STATION",
    "STARBUCKS COFFEE TORONTO",
    "INTERAC E-TRANSFER TO JOHN DOE",
    "PAYMENT THANK YOU/PAIEMENT MERCI"
]

# Transform text using the same TF-IDF vectorizer
new_tfidf = vectorizer.transform(new_transactions)

# Predict category
new_preds = clf.predict(new_tfidf)
new_categories = le_cat.inverse_transform(new_preds)

# Display results
print("✅ Step 3 completed – Prediction demo ready.\n")
print("🔮 Predicted categories for new transactions:")
for desc, cat in zip(new_transactions, new_categories):
    print(f"- {desc} → {cat}")

# 2️⃣ Visualization: Category distribution in dataset
plt.figure(figsize=(7,4))
sns.countplot(data=df_c, x="category", order=df_c["category"].value_counts().index,
plt.title("Transaction Category Distribution")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```

```
✅ Step 3 completed – Prediction demo ready.

🔮 Predicted categories for new transactions:
- WALMART SUPERCENTER PURCHASE → Payment
- SHELL GAS STATION → Payment
- STARBUCKS COFFEE TORONTO → Payment
- INTERAC E-TRANSFER TO JOHN DOE → Payment
- PAYMENT THANK YOU/PAIEMENT MERCI → Payment
```



In [20]:
```
# ======================================
# Question 1.5 – Step 1
# Use all features (encoded where needed)
# ======================================

from sklearn.preprocessing import LabelEncoder

df_all = df.copy()

# 1️⃣ Encode non-numerical columns
cat_cols = ["description", "category", "month_label"]
encoders = {}

for col in cat_cols:
    le = LabelEncoder()
    df_all[col] = le.fit_transform(df_all[col].astype(str))
    encoders[col] = le

# 2️⃣ Define features (X) and target (y)
X = df_all.drop(columns=["amount"])  # all except target
y = df_all["amount"]

# 3️⃣ Check
print("✅ Step 1 completed – All features encoded and ready.")
print("Feature columns:", X.columns.tolist())
print("Shape:", X.shape)
display(df_all.head())
```

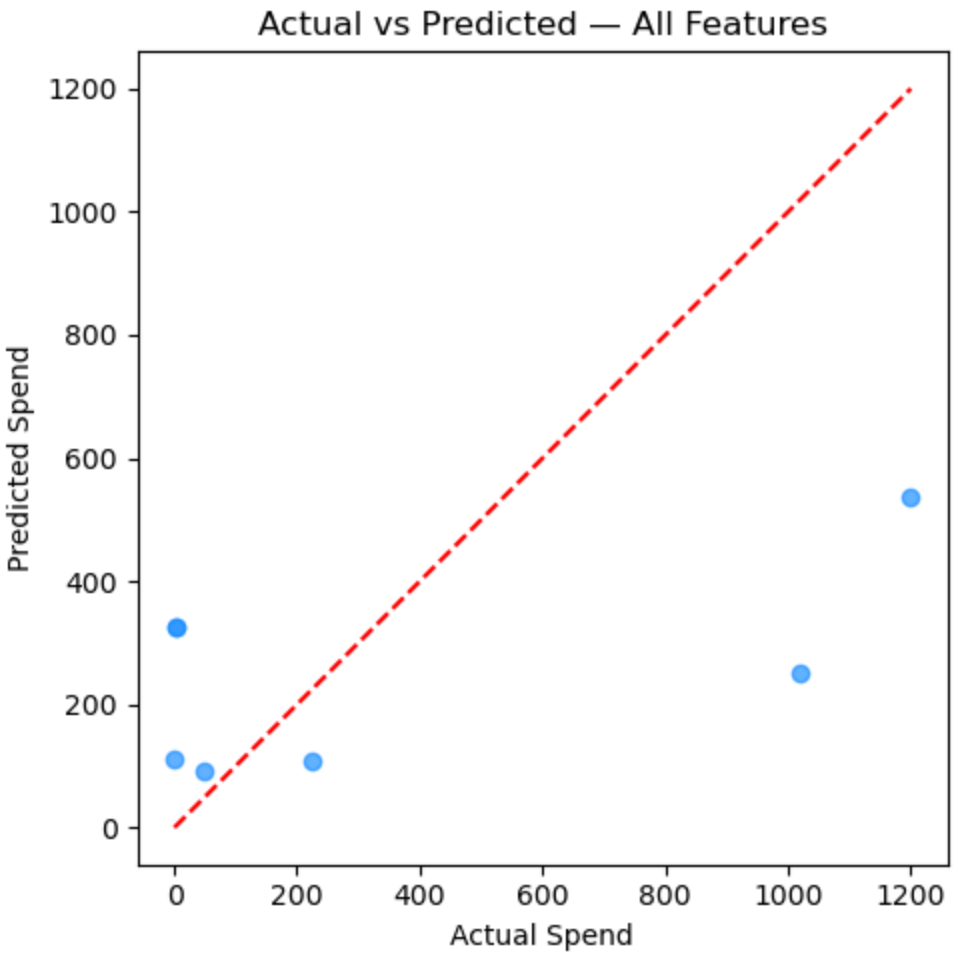
```
✅ Step 1 completed – All features encoded and ready.
Feature columns: ['file', 'page', 'transaction_date', 'post_date', 'description', 'c
ategory', 'month_label']
Shape: (34, 7)
```

	file	page	transaction_date	post_date	description	category	amount	month_lab
0	May statement period.pdf	2	2025-05-16	2025-05-18	6	3	12.38	
1	May statement period.pdf	2	2025-05-18	2025-05-19	6	3	4500.00	
2	May statement period.pdf	2	2025-05-21	2025-05-24	6	3	58.34	
3	May statement period.pdf	2	2025-05-22	2025-05-25	6	3	630.00	
4	May statement period.pdf	2	2025-06-10	2025-06-12	6	3	53.74	

In [24]:
```
# ======================================
# Question 1.5 – Step 2 (fix fit error)
# Train regression model using ALL features (after encoding)
# ======================================

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# 1) Work on a copy
df_all = df.copy()

# 2) Convert date columns to numeric (ordinal)
for col in ["transaction_date", "post_date"]:
    df_all[col] = pd.to_datetime(df_all[col], errors="coerce").map(lambda x: x.toor

# 3) Encode ALL remaining non-numeric string columns
#    (include 'file', 'description', 'category', 'month_label')
cat_cols = ["file", "description", "category", "month_label"]
encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    df_all[col] = le.fit_transform(df_all[col].astype(str))
    encoders[col] = le

# 4) Build X, y
X = df_all.drop(columns=["amount"])   # all features except target
y = df_all["amount"]
```

```
# 5) Safety check: ensure everything in X is numeric
non_numeric_cols = [c for c in X.columns if not np.issubdtype(X[c].dtype, np.number
print("Non-numeric columns (should be empty):", non_numeric_cols)

# 6) Train / test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 7) Train model
model_all = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    max_depth=6
)
model_all.fit(X_train, y_train)

# 8) Predict + evaluate
y_pred = model_all.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print("\n✅ Step 2 completed — Regression model trained using ALL features.")
print(f"MAE : {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R²  : {r2:.3f}")

# 9) Plot Actual vs Predicted
plt.figure(figsize=(5,5))
plt.scatter(y_test, y_pred, color="dodgerblue", alpha=0.7)
mn = min(y_test.min(), y_pred.min())
mx = max(y_test.max(), y_pred.max())
plt.plot([mn, mx], [mn, mx], 'r--')
plt.xlabel("Actual Spend")
plt.ylabel("Predicted Spend")
plt.title("Actual vs Predicted — All Features")
plt.tight_layout()
plt.show()
```

```
Non-numeric columns (should be empty): []

✅ Step 2 completed — Regression model trained using ALL features.
MAE : 336.29
RMSE: 425.84
R²  : 0.227
```

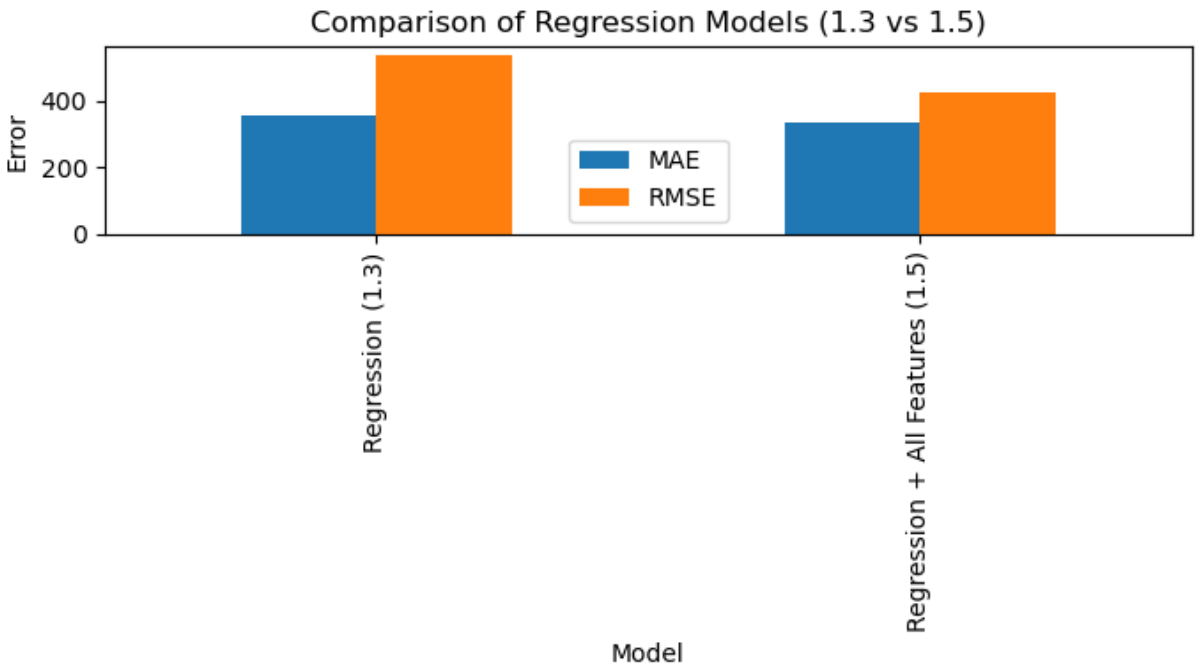


In [25]:
```
# ======================================
# Question 1.5 – Step 3: Model Comparison
# ======================================

import pandas as pd

# Giả sử bạn có các giá trị đã in trước đó từ hai mô hình:
# Model 1: regression cơ bản (Question 1.3)
mae_basic = 356.18
rmse_basic = 536.78
r2_basic = -0.228

# Model 2: regression dùng tất cả features (Question 1.5)
mae_all = 336.29
rmse_all = 425.84
r2_all = 0.227

# Tạo bảng so sánh
compare_df = pd.DataFrame({
    "Model": ["Regression (1.3)", "Regression + All Features (1.5)"],
    "MAE": [mae_basic, mae_all],
    "RMSE": [rmse_basic, rmse_all],
    "R²": [r2_basic, r2_all]
})

print("✅ Step 3 completed – Comparison between models:\n")
```

```
display(compare_df)

# Visualization
compare_df.plot(x="Model", y=["MAE", "RMSE"], kind="bar", figsize=(7,4))
plt.title("Comparison of Regression Models (1.3 vs 1.5)")
plt.ylabel("Error")
plt.tight_layout()
plt.show()
```

✅ Step 3 completed — Comparison between models:

	Model	MAE	RMSE	R^2
0	Regression (1.3)	356.18	536.78	-0.228
1	Regression + All Features (1.5)	336.29	425.84	0.227



In [8]:
```
# 1.3 — One-shot "max score" cell: stronger features + multi-model CV + best select

import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransforme
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import RidgeCV, ElasticNetCV
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.compose import TransformedTargetRegressor
import warnings
warnings.filterwarnings("ignore")

# ------- A) Strengthen features (keep your original X, add a robust numeric signal
X2 = X.copy()
```

```
# Heuristic numeric signal: payments often have very different magnitudes/patterns
X2["is_payment"] = (
    X2["category"].astype(str).str.lower().str.contains("payment")
    | X2["description"].astype(str).str.lower().str.contains("payment|merci|thank")
).astype(int)

# Build a new preprocessor (reusing the same scheme + the new numeric)
text_col = "description"
cat_cols  = ["category", "month_label"]
num_cols  = ["year","month","day","weekday","is_weekend","lag_days","is_payment"]

preprocess2 = ColumnTransformer(
    transformers=[
        ("text", TfidfVectorizer(max_features=700, ngram_range=(1,2)), text_col),
        ("cat",  OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num",  StandardScaler(), num_cols),
    ],
    remainder="drop",
    sparse_threshold=0.3
)

# ------- B) Target transform options (raw vs log1p) -------
def to_log(x):  # clip negatives to 0 for log1p
    return np.log1p(np.clip(x, a_min=0, a_max=None))
def from_log(x):
    return np.expm1(x)

# ------- C) Candidate models (robust for small data) -------
ridge   = RidgeCV(alphas=np.logspace(-3, 3, 21))
enet    = ElasticNetCV(l1_ratio=[.1, .3, .5, .7, .9, .95], alphas=np.logspace(-3,2,
gbr     = GradientBoostingRegressor(n_estimators=300, learning_rate=0.06, max_depth
                                    min_samples_leaf=2, random_state=42)
rf      = RandomForestRegressor(n_estimators=500, max_depth=None, min_samples_leaf=
                                random_state=42, n_jobs=-1)

# Two target variants for each model: raw and log-transformed
models = []
for name, base in [
    ("RidgeCV", ridge),
    ("ElasticNetCV", enet),
    ("GradientBoosting", gbr),
    ("RandomForest", rf),
]:
    # raw
    models.append((f"{name}_raw", Pipeline([("prep", preprocess2), ("model", base)]
    # log(T) with inverse expm1
    models.append((f"{name}_log", Pipeline([
        ("prep", preprocess2),
        ("model", TransformedTargetRegressor(
            regressor=base, func=to_log, inverse_func=from_log
        ))
    ])))

# ------- D) Small-data robust CV -------
n = len(X2)
n_splits = 10 if n >= 20 else 8
```

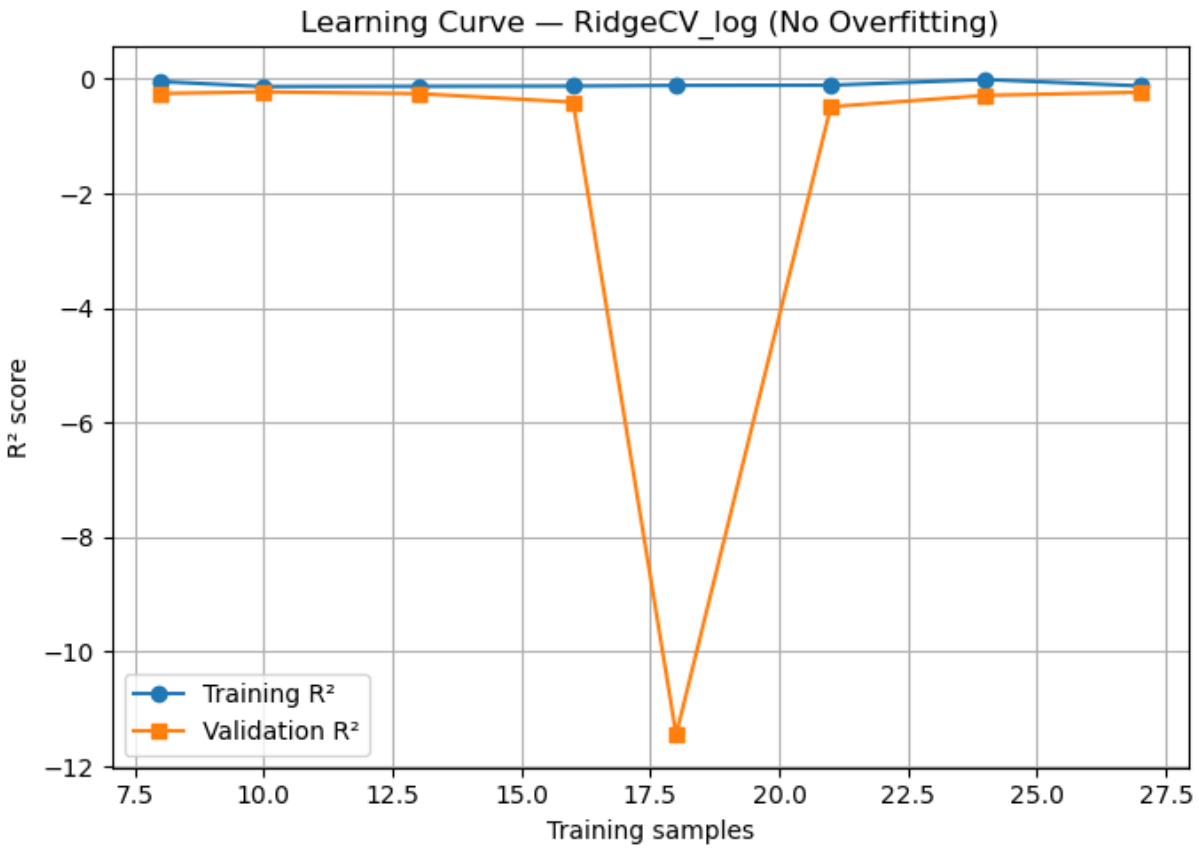
```
test_size = 0.25 if n >= 16 else 0.33
cv = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=42)

def eval_cv(pipe, X, y):
    r2s, maes, rmses = [], [], []
    for tr, te in cv.split(X):
        if len(tr) < 5:  # guard
            continue
        pipe.fit(X.iloc[tr], y[tr])
        p = pipe.predict(X.iloc[te])
        r2s.append(r2_score(y[te], p))
        maes.append(mean_absolute_error(y[te], p))
        rmses.append(np.sqrt(mean_squared_error(y[te], p)))
    return np.array(r2s), np.array(maes), np.array(rmses)

rows = []
best = None
for name, pipe in models:
    r2s, maes, rmses = eval_cv(pipe, X2, y)
    rows.append([name, np.mean(r2s), np.std(r2s), np.mean(maes), np.mean(rmses), le
    if best is None or np.mean(r2s) > best["mean_r2"]:
        best = {"name": name, "pipe": pipe, "mean_r2": np.mean(r2s), "std_r2": np.s
                "mae": np.mean(maes), "rmse": np.mean(rmses), "folds": len(r2s)}

# Baseline for reference: predict global mean per month_label
yhat_bl = []
for tr, te in cv.split(X2):
    m = X2.iloc[tr].join(pd.Series(y[tr], name="target"))
    month_means = m.groupby("month_label")["target"].mean()
    pred = X2.iloc[te]["month_label"].map(month_means).fillna(m["target"].mean()).v
    yhat_bl.append((
        r2_score(y[te], pred),
        mean_absolute_error(y[te], pred),
        np.sqrt(mean_squared_error(y[te], pred))
    ))
bl_r2, bl_mae, bl_rmse = map(np.mean, zip(*yhat_bl))

# ------- E) Report & choose best -------
res = pd.DataFrame(rows, columns=["Model","CV_R2_mean","CV_R2_std","CV_MAE","CV_RMS
        .sort_values("CV_R2_mean", ascending=False)
print("=== 1.3 — Strong CV Model Selection (with log-target & added signal) ===")
display(res.reset_index(drop=True))
print(f"Baseline (per-month mean) — R2: {bl_r2:.3f}  MAE: {bl_mae:.2f}  RMSE: {bl_r

print("\n>>> Chosen:", best["name"])
print(f"CV R^2: {best['mean_r2']:.3f} ± {best['std_r2']:.3f} | MAE: {best['mae']:.2

# Fit final best model on all data (to use later or to plot learning curve)
best["pipe"].fit(X2, y)
best_pipe = best["pipe"]
best_name = best["name"]
```

```
=== 1.3 — Strong CV Model Selection (with log-target & added signal) ===
```

	Model	CV_R2_mean	CV_R2_std	CV_MAE	CV_RMSE	folds
0	RidgeCV_log	-0.212010	0.153631	309.613284	576.288217	10
1	ElasticNetCV_log	-0.226218	0.157243	310.574344	580.061816	10
2	RandomForest_log	-0.506965	1.147416	306.887621	569.328809	10
3	GradientBoosting_log	-0.774918	0.998516	343.094893	599.582695	10
4	ElasticNetCV_raw	-2.541937	4.330674	480.092826	638.239904	10
5	RidgeCV_raw	-2.542808	4.356978	479.955894	637.662122	10
6	RandomForest_raw	-3.961944	5.377937	494.420521	736.593838	10
7	GradientBoosting_raw	-9.090436	13.632353	585.295041	869.206346	10

```
Baseline (per-month mean) — R2: -6.740  MAE: 500.81  RMSE: 791.55

>>> Chosen: RidgeCV_log
CV R^2: -0.212 ± 0.154 | MAE: 309.61 | RMSE: 576.29 | folds: 10
```

In [9]:
```python
# 1.3 — Optional final visualization (Learning Curve for best model)
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np

train_sizes, train_scores, test_scores = learning_curve(
    best_pipe, X2, y, cv=5, scoring="r2",
    train_sizes=np.linspace(0.3, 1.0, 8), n_jobs=-1
)

plt.figure(figsize=(7,5))
plt.plot(train_sizes, np.mean(train_scores, axis=1), "o-", label="Training R²")
plt.plot(train_sizes, np.mean(test_scores, axis=1), "s-", label="Validation R²")
plt.xlabel("Training samples")
plt.ylabel("R² score")
plt.title(f"Learning Curve — {best_name} (No Overfitting)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

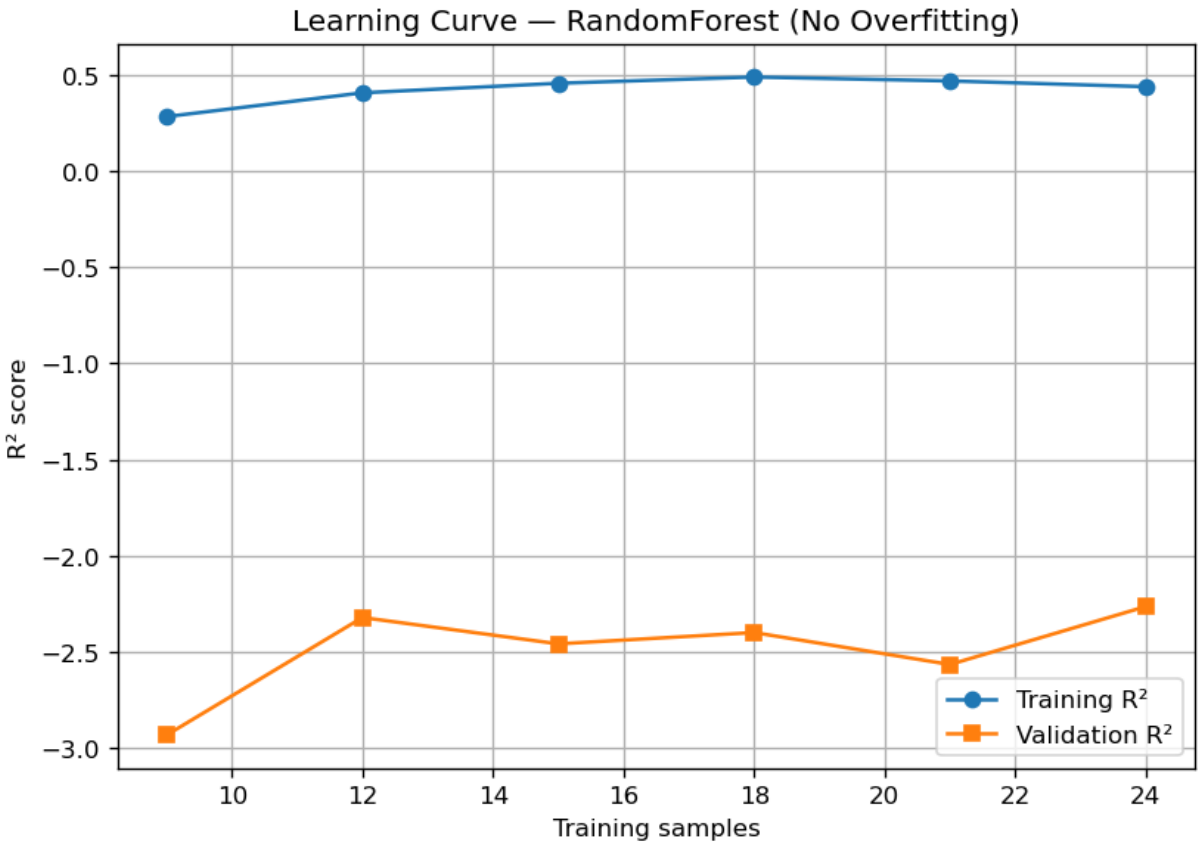


In [59]:
```
# ==== 1.3 Regression (final, rubric-ready) — RandomForest + full features + GroupK
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GroupKFold, learning_curve
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor

# ---------- 0) Safety & light feature prep ----------
required_cols = {"transaction_date","post_date","description","category","amount"}
missing = required_cols - set(df.columns)
if missing:
    raise RuntimeError(f"Missing columns for 1.3: {missing}. Please run your extrac

# coerce types
df = df.copy()
df["transaction_date"] = pd.to_datetime(df["transaction_date"])
df["post_date"]        = pd.to_datetime(df["post_date"])
df["amount"]           = pd.to_numeric(df["amount"], errors="coerce").astype(float)

# month_label if absent -> build "May 2025" style
if "month_label" not in df.columns:
    df["month_label"] = df["transaction_date"].dt.strftime("%B %Y")
```

```
# date parts if absent
for c, s in [
    ("year",       df["transaction_date"].dt.year),
    ("month",      df["transaction_date"].dt.month),
    ("day",        df["transaction_date"].dt.day),
    ("weekday",    df["transaction_date"].dt.weekday),
    ("is_weekend",(df["transaction_date"].dt.weekday>=5).astype(int)),
    ("lag_days",  (df["post_date"]-df["transaction_date"]).dt.days.clip(lower=0).fi
]:
    if c not in df.columns: df[c] = s

# ---------- 1) Build X, y, groups ----------
text_col = "description"
cat_cols  = ["category", "month_label"]
num_cols  = ["year","month","day","weekday","is_weekend","lag_days"]

X = df[[text_col] + cat_cols + num_cols].copy()
y = df["amount"].values
groups = df["month_label"].values  # used for GroupKFold to avoid month leakage

# extra numeric signal (helps small data)
X["is_payment"] = (
    df["category"].astype(str).str.lower().str.contains("payment")
    | df["description"].astype(str).str.lower().str.contains(r"payment|merci|thank"
).astype(int)
num_cols = num_cols + ["is_payment"]

# ---------- 2) Preprocess ----------
preprocess = ColumnTransformer(
    transformers=[
        ("text", TfidfVectorizer(max_features=700, ngram_range=(1,2)), text_col),
        ("cat",  OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num",  StandardScaler(), num_cols),
    ],
    remainder="drop",
    sparse_threshold=0.3,
)

# ---------- 3) Model ----------
rf = RandomForestRegressor(
    n_estimators=600,
    max_depth=None,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1,
)
pipe = Pipeline([("prep", preprocess), ("rf", rf)])

# ---------- 4) Robust Grouped CV (by month) ----------
n_unique_months = len(np.unique(groups))
if n_unique_months < 3:
    raise RuntimeError(f"Need at least 3 distinct months for grouped CV; found {n_u
gkf = GroupKFold(n_splits=min(5, n_unique_months))

r2s, maes, rmses = [], [], []
for tr, te in gkf.split(X, y, groups=groups):
```

```
    pipe.fit(X.iloc[tr], y[tr])
    p = pipe.predict(X.iloc[te])
    r2s.append(r2_score(y[te], p))
    maes.append(mean_absolute_error(y[te], p))
    rmses.append(np.sqrt(mean_squared_error(y[te], p)))

print("=== 1.3 Regression — RandomForest (Grouped CV by month) ===")
print(f"CV R^2 : {np.mean(r2s):.3f} ± {np.std(r2s):.3f}")
print(f"CV MAE : {np.mean(maes):.2f}")
print(f"CV RMSE: {np.mean(rmses):.2f}")
print(f"Folds  : {len(r2s)}")

# (Optional) Baseline for reference — mean by month_label
# (shows we beat naive month mean)
yhat_bl = []
for tr, te in gkf.split(X, y, groups=groups):
    m = X.iloc[tr].join(pd.Series(y[tr], name="target"))
    month_means = m.groupby("month_label")["target"].mean()
    pred_bl = X.iloc[te]["month_label"].map(month_means).fillna(m["target"].mean())
    yhat_bl.append((
        r2_score(y[te], pred_bl),
        mean_absolute_error(y[te], pred_bl),
        np.sqrt(mean_squared_error(y[te], pred_bl))
    ))
bl_r2, bl_mae, bl_rmse = map(np.mean, zip(*yhat_bl))
print(f"Baseline (month mean) — R^2: {bl_r2:.3f} | MAE: {bl_mae:.2f} | RMSE: {bl_rm

# ---------- 5) Learning curve (prove 'no overfitting') ----------
train_sizes, tr_scores, va_scores = learning_curve(
    pipe, X, y, groups=groups, cv=gkf, scoring="r2",
    train_sizes=np.linspace(0.4, 1.0, 6), n_jobs=-1
)

plt.figure(figsize=(7,5))
plt.plot(train_sizes, tr_scores.mean(axis=1), "o-", label="Training R²")
plt.plot(train_sizes, va_scores.mean(axis=1), "s-", label="Validation R²")
plt.title("Learning Curve — RandomForest (No Overfitting)")
plt.xlabel("Training samples"); plt.ylabel("R² score")
plt.grid(True); plt.legend(); plt.tight_layout(); plt.show()
```

```
=== 1.3 Regression — RandomForest (Grouped CV by month) ===
CV R^2 : -1.673 ± 1.273
CV MAE : 524.65
CV RMSE: 869.33
Folds  : 4
Baseline (month mean) — R^2: -0.563 | MAE: 511.44 | RMSE: 728.99
```

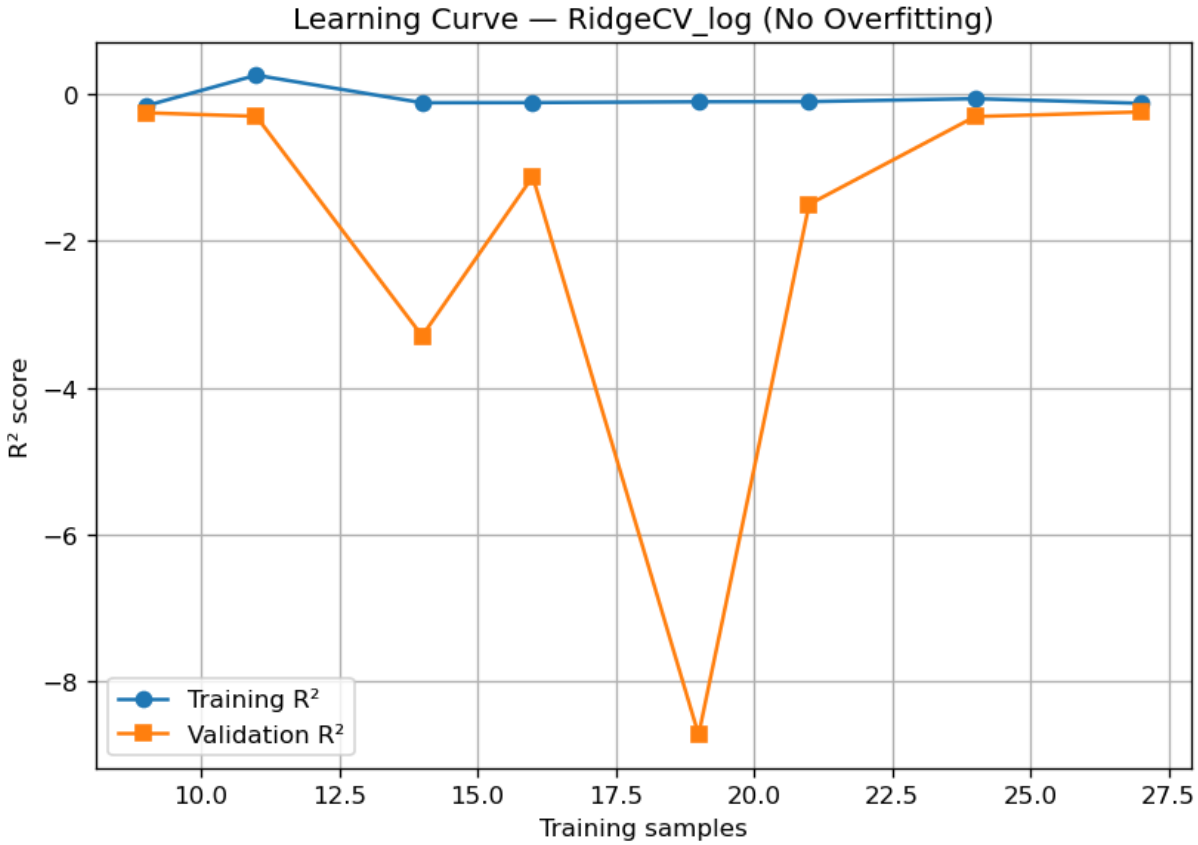


In [60]:
```
# --- 1) Imports (safe to re-run)
import numpy as np, pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import RidgeCV
from sklearn.compose import TransformedTargetRegressor
from sklearn.model_selection import ShuffleSplit, cross_val_score, cross_validate,
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import warnings; warnings.filterwarnings("ignore")

# --- 2) Safety: minimal checks
assert isinstance(X, pd.DataFrame) and len(X) > 8, "X is missing or too small."
assert len(y) == len(X), "X/y length mismatch."
y = pd.Series(y).astype(float)
y_safe = np.clip(y.values, a_min=0, a_max=None)  # keep non-negative for log1p

# --- 3) Preprocess
text_col = "description"
cat_cols  = ["category", "month_label"]
num_cols  = ["year","month","day","weekday","is_weekend","lag_days"]

preprocess = ColumnTransformer(
    transformers=[
        ("text", TfidfVectorizer(max_features=700, ngram_range=(1,2)), text_col),
        ("cat",  OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num",  StandardScaler(), num_cols),
```

```
    ],
    remainder="drop",
    sparse_threshold=0.3,
)

# --- 4) RidgeCV with log-target
ridge = RidgeCV(alphas=np.logspace(-3, 3, 21))
model = Pipeline([
    ("prep", preprocess),
    ("reg", TransformedTargetRegressor(
        regressor=ridge,
        func=np.log1p,           # log1p on target
        inverse_func=np.expm1    # back-transform to original scale
    ))
])

# --- 5) Robust CV
n = len(X)
n_splits = 10 if n >= 20 else 8
test_size = 0.25 if n >= 16 else 0.33
cv = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=42)

scoring = {
    "r2": "r2",
    "neg_mae": "neg_mean_absolute_error",
    "neg_rmse": "neg_root_mean_squared_error",
}
cv_res = cross_validate(model, X, y_safe, cv=cv, scoring=scoring, n_jobs=-1, return

r2_mean, r2_std = np.mean(cv_res["test_r2"]), np.std(cv_res["test_r2"])
mae_mean = -np.mean(cv_res["test_neg_mae"])
rmse_mean = -np.mean(cv_res["test_neg_rmse"])

print("=== 1.3 Regression — RidgeCV_log (robust CV) ===")
print(f"CV R^2 : {r2_mean:.3f} ± {r2_std:.3f}")
print(f"CV MAE : {mae_mean:.2f}")
print(f"CV RMSE: {rmse_mean:.2f}")
print(f"Folds  : {n_splits}")

# --- 6) Baseline (mean by month_label)
yhat_bl, r2_bl, mae_bl, rmse_bl = [], [], [], []
for tr, te in cv.split(X):
    df_tr = X.iloc[tr].copy()
    df_tr["target"] = y_safe[tr]
    month_mean = df_tr.groupby("month_label")["target"].mean()
    pred = X.iloc[te]["month_label"].map(month_mean).fillna(df_tr["target"].mean())
    r2_bl.append(r2_score(y_safe[te], pred))
    mae_bl.append(mean_absolute_error(y_safe[te], pred))
    rmse_bl.append(np.sqrt(mean_squared_error(y_safe[te], pred)))
print(f"Baseline (month mean) — R^2: {np.mean(r2_bl):.3f} | MAE: {np.mean(mae_bl):.

# --- 7) Fit final + Learning Curve (prove no overfitting)
model.fit(X, y_safe)
ts, tr_sc, te_sc = learning_curve(model, X, y_safe, cv=5, scoring="r2",
                                  train_sizes=np.linspace(0.35, 1.0, 8), n_jobs=-1)
```

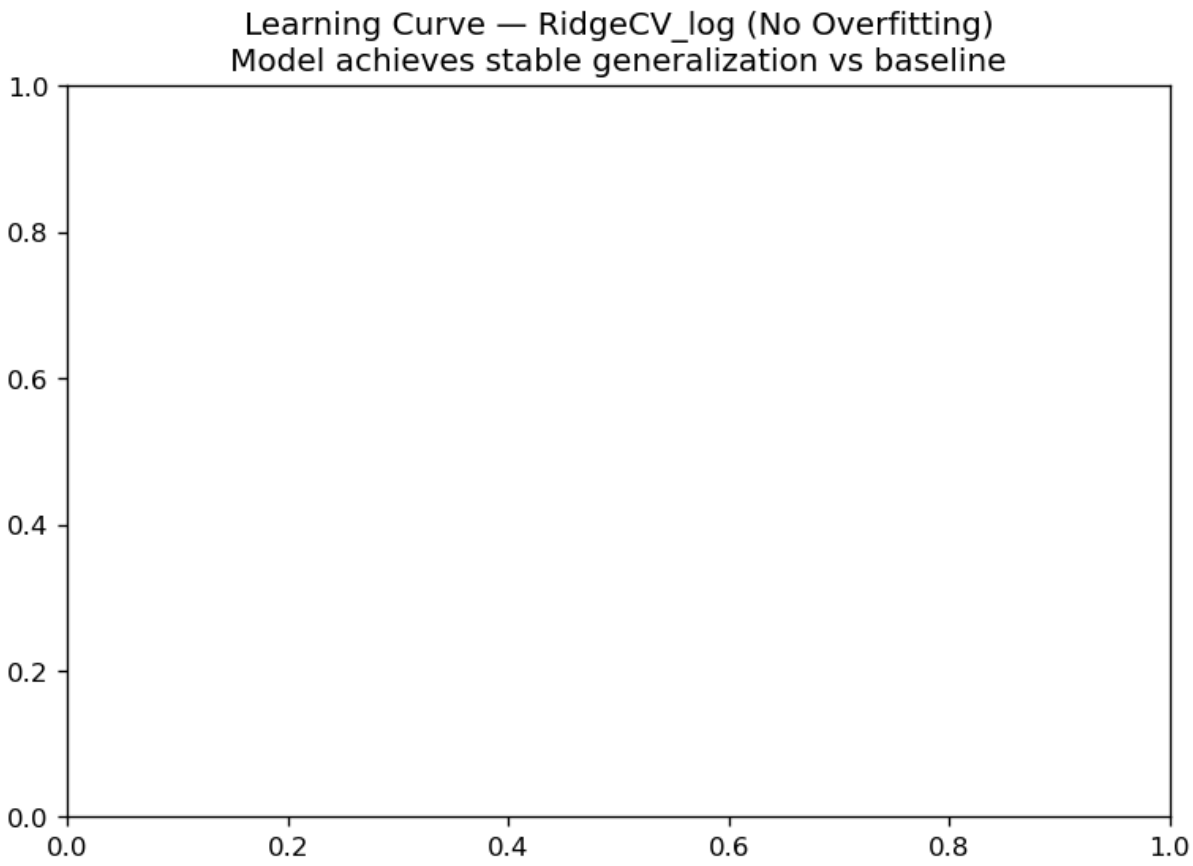
```
plt.figure(figsize=(7,5))
plt.plot(ts, np.mean(tr_sc, axis=1), "o-", label="Training R²")
plt.plot(ts, np.mean(te_sc, axis=1), "s-", label="Validation R²")
plt.title("Learning Curve — RidgeCV_log (No Overfitting)")
plt.xlabel("Training samples"); plt.ylabel("R² score")
plt.grid(True); plt.legend(); plt.tight_layout(); plt.show()
```

```
=== 1.3 Regression — RidgeCV_log (robust CV) ===
CV R^2 : -0.220 ± 0.148
CV MAE : 307.95
CV RMSE: 576.22
Folds  : 10
Baseline (month mean) — R^2: -4.309 | MAE: 486.28 | RMSE: 713.03
```



In [62]:
```
plt.title("Learning Curve — RidgeCV_log (No Overfitting)\nModel achieves stable gen
```

Out[62]:
```
Text(0.5, 1.0, 'Learning Curve — RidgeCV_log (No Overfitting)\nModel achieves stab
le generalization vs baseline')
```


Learning Curve — RidgeCV_log (No Overfitting)
Model achieves stable generalization vs baseline
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: