03_feature_engineering_text

August 21, 2025

0.0.1 Using LLM for Theme Extraction

In this step, we transform the raw `transaction_description` field into structured **themes**.

- **What we will be doing now**
  - For each contract's `transaction_description`, we send the text to an **LLM**.

  - The LLM classifies the description into one of a controlled set of themes (e.g., *IT Services, Construction, Healthcare, Defense Equipment*).

  - The output is stored as a new feature, `theme`, for use in downstream modeling.

Example:
- Input: `"PROCUREMENT OF SERVER MAINTENANCE AND SUPPORT"`
- LLM Output: `"IT Services"`

This converts unstructured text into a consistent categorical feature without requiring custom text preprocessing pipelines.

0.0.2 What Could Have Been Done with More Data

If we had a **very large number of lapse contracts** (positive class examples), additional strategies could have been explored to use llm (only few calls to manage cost):

- **Clustering Descriptions**
  - Apply unsupervised methods (e.g., k-means, hierarchical clustering, or embeddings + cosine similarity) on `transaction_description`.

  - Group contracts into clusters of semantically similar descriptions before sending cluster representatives to the LLM.

  - This reduces API calls and ensures that descriptions with similar wording map to the same theme.
- **Encoding Before LLM**
  - Instead of sending raw text, first transform descriptions into vector embeddings or cluster IDs.

  - Then only send representative samples to the LLM for classification into themes.
- **Why we didn't do this here**

- Our dataset has **relatively few lapse contracts** compared to the total population.
- Over-engineering text clustering/encoding before the LLM would risk distorting the balance between **lapse** and **non-lapse** examples.
- To keep signals consistent, we directly map each contract's description to a theme using the LLM.

[1]:
```
# Standard library
import os
import re
import time
import logging
from typing import List, Tuple

# Third-party Library
import pandas as pd

# LLM API
from groq import Groq
```

[2]:
```
# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)s | %(message)s"
)
```

[3]:
```
df = pd.read_csv('fea_eng_basic.csv')
df
```

[3]:
```
      federal_action_obligation  total_dollars_obligated  \
0                      23116.94                 23116.94
1                      84000.00                284319.29
2                         66.00                    66.00
3                       1216.40                  1216.40
4                        259.70                   259.70
…                          …                        …
2627                   22973.12                119973.12
2628                    1008.64                  1008.64
2629                     374.70                   374.70
2630                     289.34                   289.34
2631                      26.62                    26.62

      current_total_value_of_award  potential_total_value_of_award  \
0                         23116.94                        23116.94
1                        284319.29                       599885.84
2                            66.00                           66.00
3                          1216.40                         1216.40
```

```
4                           259.70                          259.70
…                             …                               …
2627                     119973.12                       119973.12
2628                       1008.64                         1008.64
2629                        374.70                          374.70
2630                        289.34                          289.34
2631                         26.62                           26.62

      action_date_fiscal_year  funding_agency_code  award_type  \
0                        2022                   13           1
1                        2022                   36           1
2                        2022                   15           1
3                        2022                   97           0
4                        2022                   97           0
…                         …                    …          …
2627                     2023                   97           1
2628                     2022                   15           0
2629                     2022                   15           0
2630                     2022                   97           0
2631                     2022                   15           0

      type_of_contract_pricing  \
0                            0
1                            0
2                            0
3                            1
4                            1
…                         …
2627                         0
2628                         0
2629                         0
2630                         1
2631                         0

                          transaction_description  extent_competed  …  \
0     WEATHER OBSERVING STATION INCLUDING INSTALLATI…              1  …
1     RADIOPHARMACEUTICALS FOR THE MONTANA VA HEALTH…              1  …
2                    OPEN MARKET PHARMACEUTICALS ORDER               3  …
3                    4556151540!BLANKET DISP COMFORT 1               0  …
4                          4556017656!CLEANING MONITOR               0  …
…                                                   …             …  …
2627                                ELECTRONIC CONTROL               4  …
2628  FY22 MCKESSON CONTROLLED CONTRACTED  PHARMACEU…              0  …
2629  PHARMACY ORDER FOR INMATES INCARCERATED AT FCI…              0  …
2630                    4557236030!FILE K 25MM #15 6S               0  …
2631              JULY MCKESSON CONTROLLED SUBSTANCES               0  …
```

```
           sale       revt        ib         lt        ceq      oancf  \
0     39211.000  39211.000  7725.000  54146.000  40793.000  9312.000
1        40.697     40.697   -54.454    125.427     52.413   -48.746
2       583.187    583.187     7.729    400.966    667.099   -66.537
3     12401.021  12401.021   631.232   3804.587   3425.126   709.580
4     12401.021  12401.021   631.232   3804.587   3425.126   709.580
…           …          …        …         …          …         …
2627     26.074     26.074     0.481      9.036     20.020     0.250
2628    583.187    583.187     7.729    400.966    667.099   -66.537
2629    583.187    583.187     7.729    400.966    667.099   -66.537
2630  12401.021  12401.021   631.232   3804.587   3425.126   709.580
2631    583.187    583.187     7.729    400.966    667.099   -66.537

           xrd       cogs  psc_3digit_freq  lapse_flag
0     1406.000  18171.000         120352.0           1
1       85.641    115.855         635345.0           1
2       29.307    118.470         635345.0           0
3        0.000   8534.570         120352.0           0
4        0.000   8534.570         120352.0           0
…          …          …              …          …
2627     1.828     16.821            459.0           1
2628    29.307    118.470         635345.0           0
2629    29.307    118.470         635345.0           0
2630     0.000   8534.570         463837.0           0
2631    29.307    118.470         635345.0           0

[2632 rows x 34 columns]
```

Check few transaction_descriptions

[4]:
```python
# Filter lapsed contracts
lapsed_df = df[df["lapse_flag"] == 1]

# Randomly sample 50 transaction descriptions
sample_desc = lapsed_df["transaction_description"].dropna().sample(50, 
 ↪random_state=42)

# Print them
for i, desc in enumerate(sample_desc, 1):
    print(f"{i}. {desc}")
```

```
1. WIRELESS LAN INFRASTRUCTURE LCR
2. REMANUFACTURE OF FLOW CONTROL VALVE
3. MIGRATED ID08190050  DYNAMIC AND EVOLVING FEDERAL ENTERPRISE NETWORK DEFENSE
GROUP E DEFEND E
4. NETSPOT RADIOPHARMACEUTICAL DELIVERY OPTION YEAR 3
5. RADIUM-223 DICHLORIDE (XOFIGO)
6. PRODUCT SUPPORT PLAN TO PROVIDE EXTENDED WARRANTY ON HARDWARE/SOFTWARE AND
```

SOFTWARE UPDATES. INCLUDES SITE-VISIT OF UP TO 3 DAYS FOR ON-SITE SUPPORT IF NECESSARY. SERVICE TO BE PROVIDED FOR 12 MONTHS
7. EO14042 ADDING COVID-19 CLAUSE AS REQUIRED BY EXECUTIVE ORDER
8. REAGENTS/CONSUMABLES/CONTROLS FOR COBAS 4800 TESTING SYSTEM
9. DISPOSAL, TRANSPORTATION, AND RECYCLING OF FERROUS AND NON-FERROUS SCRAP METALS
10. CHEMISTRY/ IMMUNOCHEMISTRY EQUIPMENT, CPRR REAGENTS, OPTION YEAR 1
11. MCKESSON OPEN MARKET MEDICATIONS
12. AYDIN DISPLAYS AND OPTICONN DVI EXTENDERS
13. REGULAR MEDS: FY22 (NOVEMBER 3, 2021)
14. REAGENTS
15. MIGRATED ID08190050 DYNAMIC AND EVOLVING FEDERAL ENTERPRISE NETWORK DEFENSE GROUP E DEFEND E
16. VIDAS 3 REAGENTS AND SUPPLIES
17. REGULAR MEDS: FY22 (DECEMBER 3, 2021)
18. FUNDING INCREASE PO# 546C20054
19. CIRCUIT CARD ASSEMB
20. NANOSTRING GEOMX
21. UPDATE SOW TO REV C
22. INO NITRIC OXIDE - MODIFICATION TO UPDATE POC AND WAWF ACCEPTOR.
23. AB SCIEX LLC, QTRAP 6500+ LC/MS/MS
24. POLAR ORGANIC CHEMICAL INTEGRATIVE SAMPLERS
25. VISN 20 POC WHOLE BLOOD GLUCOSE TESTING ANALYZERS BPA
26. REAGENTS
27. DYNAMIC AND EVOLVING FEDERAL ENTERPRISE NETWORK DEFENSE GROUP E DEFEND E
28. CONTRACTOR SHALL PROVIDE "OPEN MARKET" PHARMACEUTICAL SUPPLIES.
29. 39 MULTIFUNCTIONAL DEVICES (LEASE) - ADD DFARS CLAUSE 252.223-7999. EO14042.
30. AYDIN DISPLAYS AND OPTICONN DVI EXTENDERS
31. MICROBIOLOGY TESTS, LOT
32. GXP EXPLORER ENT.
33. THE PURPOSE OF THIS MODIFICATION CHANGE THE CONTRACTING OFFICER AND CLOSEOUT CONTRACT.
34. MCKESSON GENERAL MEDICATIONS 10/01/2023-11/17/2023
35. ADDED FUNDS THAT WERE ERRONEOUSLY REMOVED.
36. BLOOD CULTURE BOTTLES: MODIFICATION TO EXERCISE OPTION QUANTITY ONE
37. AUTOMATED URINALYSIS
38. ROUTINE AND NARCOTIC MEDICINES 36W79720D0001
39. ILLUMINA INC (AMBIS #1843684)
40. CPT FOR ONE (1) TOSOH G8 HBA1C HPLC ANALYZER
41. CHEMISTRY IMMUNOASSAY ANALYZER FOR BIG SPRING VA
42. ULTRIO ELITE TEST KIT
43. CELLULAR LOAD TESTER SUPPORT - EO 14042
44. FILMARRAY DIAGNOSTICS TESTING
45. COST PER TEST - IMMUNOCHEMISTRY ANALYZER
46. EMBEDDED GLOBAL POSITIONING SYSTEM/INERTIAL NAVIGATION SYSTEM (EGI) - MODERNIZED (EGI-M) ENGINEERING, MANUFACTURING AND DEVELOPMENT (EMD) PHASE
47. PON MAINTENANCE CYBERSECURE BASE YEAR
48. MICROBIOLOGY IDENTIFICATION AND SUSECPTIBILITY

```
49. MIGRATED ID08190050  DYNAMIC AND EVOLVING FEDERAL ENTERPRISE NETWORK DEFENSE
GROUP E DEFEND E
50. GN TEST KIT/AST-GN95 PN 421982
```

0.0.3 Developing the Themes

We developed the following set of themes by **randomly sampling and reviewing many `transaction_description` values**, then iteratively refining categories.
This process ensured coverage across the most common spending patterns while keeping the number of categories manageable.

Having a consistent theme mapping helps:
- Reduce noise from messy free-text descriptions.
- Provide interpretable contract groupings for downstream models.
- Make LLM calls more reliable by constraining the classification set.

Theme Categories

a. Medical & Healthcare Supplies/Services

b. IT, Software & Hardware

c. Facilities, Utilities & Maintenance

d. Professional & Management Services

e. Financial & Contract Modifications

f. Training, Education & Outreach

g. Logistics & Support Services

h. Closeout & Administrative Actions

i. Defense, Aerospace & Mechanical Equipment

j. Other

0.0.4 Use of LLM for Feature Mapping

We will leverage a Large Language Model (LLM) to classify contract descriptions into predefined themes.

Process Overview:
1. **Batching the Data**
- Large datasets are split into manageable batches (e.g., 50 descriptions per batch).

2. **Prompt Construction**
    - Each batch is formatted into a structured prompt with transaction descriptions and the theme list.
    - The model is instructed to assign **exactly one theme per item**.
3. **Model Inference**
    - The `meta-llama/llama-4-scout-17b-16e-instruct` model is used for classification.
    - The output is in the strict format: `<number>) <theme_letter>`.
4. **Parsing & Updating**
    - The raw output is parsed into a dictionary mapping row numbers → theme.
    - The dataframe is updated with theme classifications for each transaction.
5. **Iterative Processing**
    - The process repeats across all batches, with delays between requests to respect API rate limits.
6. **Final Output**
    - The dataframe contains a new `Theme` column, ensuring each contract is mapped consistently.

LLM Model Discussion:
- `openai/gpt-oss-120b` → Did not provide correct classification format.
- `openai/gpt-oss-20b` → Also failed to return responses in the expected structured style.
- `qwen/qwen3-32b` → Misunderstood the prompt and gave verbose answers such as:
"Theme a is Medical & Healthcare Supplies/Services. That would include things like medications, medical devices, healthcare services, etc. So items with drugs, medical equipment, or services related to healthcare should go here."
- `llama-4-scout-17b-16e-instruct` was the **only model** that consistently followed the prompt and returned clean, structured responses as required.

[5]:
```
# Theme map
THEME_MAP = {
    "a": "Medical & Healthcare Supplies/Services",
    "b": "IT, Software & Hardware",
    "c": "Facilities, Utilities & Maintenance",
    "d": "Professional & Management Services",
    "e": "Financial & Contract Modifications",
    "f": "Training, Education & Outreach",
    "g": "Logistics & Support Services",
    "h": "Closeout & Administrative Actions",
    "i": "Defense, Aerospace & Mechanical Equipment",
    "j": "Other",
}

# tolerant parser: "1) a", "1) a - extra", "1) A" all OK
ROW_RE = re.compile(r"^\s*(\d+)\)\s*([a-j])\b", re.IGNORECASE)
```

[11]:

```python
class GroqThemeClassifier:
    """
    Classify contract descriptions into predefined themes using Groq LLM API.
    """

    def __init__(
        self,
        api_key: str = None,
        model: str = "meta-llama/llama-4-scout-17b-16e-instruct",
        rpm_delay_sec: float = 2.0,
    ):
        """
        Parameters
        ----------
        api_key : str, optional
            Groq API key. If None, will read from env variable GROQ_API_KEY.
        model : str
            Groq model to use for classification.
        rpm_delay_sec : float
            Delay (seconds) between batches to respect rate limits.
        """
        self.api_key = api_key or os.getenv("GROQ_API_KEY")
        self.client = Groq(api_key=self.api_key)
        self.model = model
        self.rpm_delay_sec = rpm_delay_sec

        self.logger = logging.getLogger(self.__class__.__name__)

    @staticmethod
    def parse_mapping(text: str) -> dict:
        """Parse model output into {row_number: theme} mapping."""
        mapping = {}
        for line in str(text).splitlines():
            m = ROW_RE.match(line.strip())
            if m:
                num = int(m.group(1))
                letter = m.group(2).lower()
                mapping[num] = THEME_MAP[letter]
        return mapping

    @staticmethod
    def build_prompt(numbered_items: List[Tuple[int, str]]) -> str:
        """Build classification prompt for Groq."""
        header = [
            "Classify EACH transaction into EXACTLY ONE theme letter (a-j).",
            "",
            "Themes:",
```

```
            *[f"{k}. {v}" for k, v in THEME_MAP.items()],
            "",
            "Return ONLY lines in the exact format: <number>) <letter>",
            "Do NOT include any other text before or after.",
            "",
            "Items:",
        ]
        lines = [f"{i}) {txt}" for i, txt in numbered_items]
        return "\n".join(header + lines)

    def classify_texts(self, texts: List[str], max_tokens: int = 200) -> str:
        """Send a batch of texts to Groq and return raw model output."""
        prompt = self.build_prompt([(i + 1, t) for i, t in enumerate(texts)])
        resp = self.client.chat.completions.create(
            model=self.model,
            messages=[
                {"role": "system", "content": "You are a precise classifier."},
                {"role": "user", "content": prompt},
            ],
            temperature=0,
            max_tokens=max_tokens,
        )
        return resp.choices[0].message.content

    def classify_in_batches(
        self,
        df: pd.DataFrame,
        text_col: str = "transaction_description",
        theme_col: str = "Theme",
        batch_size: int = 50,
        max_batches: int = None,
        show_prompts: bool = False,
        show_model_output: bool = True,
    ) -> pd.DataFrame:
        """
        Classify descriptions into themes in batches.

        Parameters
        ----------
        df : pd.DataFrame
            Input dataframe with a text column.
        text_col : str
            Column containing contract descriptions.
        theme_col : str
            Column to write theme classifications into.
        batch_size : int
            Number of rows per batch.
```

```
        max_batches : int, optional
            If set, only process up to this many batches.
        show_prompts : bool
            Whether to log full prompts for debugging.
        show_model_output : bool
            Whether to log raw model output.

        Returns
        -------
        pd.DataFrame
            Dataframe with new theme assignments.
        """
        if theme_col not in df.columns:
            df[theme_col] = pd.Series(index=df.index, dtype="object")

        idxs = df.index[df[text_col].notna()].tolist()
        total = len(idxs)
        if total == 0:
            self.logger.warning("No rows to classify.")
            return df

        num_batches = (total + batch_size - 1) // batch_size
        if max_batches is not None:
            num_batches = min(num_batches, max_batches)

        self.logger.info("Planned: %s batch(es) of up to %s rows", num_batches,␣
↪batch_size)

        for b in range(num_batches):
            start, end = b * batch_size, min((b + 1) * batch_size, total)
            batch_index = idxs[start:end]

            numbered_items = [(i, str(df.at[idx, text_col]).strip())
                              for i, idx in enumerate(batch_index, start=1)]

            prompt = self.build_prompt(numbered_items)
            if show_prompts:
                self.logger.debug("Prompt (batch %s/%s):\n%s", b + 1,␣
↪num_batches, prompt)

            # Call Groq API
            raw_output = self.classify_texts(
                [t for _, t in numbered_items],
                max_tokens=max(600, 10 * len(numbered_items)),
            )
            if show_model_output:
```

```
                self.logger.debug("Raw model output (batch %s/%s):\n%s", b + 1,␣
↪num_batches, raw_output)

            # Parse and update
            mapping = self.parse_mapping(raw_output)
            if not mapping:
                self.logger.warning("No lines parsed for batch %s. Skipping␣
↪updates.", b + 1)
            else:
                for local_num, idx in enumerate(batch_index, start=1):
                    df.at[idx, theme_col] = mapping.get(local_num,␣
↪THEME_MAP["j"])

            # Rate limiting
            if b < num_batches - 1 and self.rpm_delay_sec:
                time.sleep(self.rpm_delay_sec)

        self.logger.info("Classification complete. Updated column '%s'.",␣
↪theme_col)
        return df
```

[16]:

```
# Logging at `DEBUG` level is enabled so raw model outputs can be monitored in␣
↪test runs
logging.getLogger("GroqThemeClassifier").setLevel(logging.DEBUG)
```

[12]:

```
# Test API Call
clf = GroqThemeClassifier()

# Test API Call
df_out = clf.classify_in_batches(
        df,
        text_col="transaction_description",
        theme_col="Theme",
        batch_size=3,         # 3 rows per batch
        max_batches=2,        # only call 2 batches
        show_prompts=True,    # log full prompts for debugging
        show_model_output=True  # log raw LLM responses
    )
```

```
2025-08-21 18:17:15,455 | INFO | Planned: 2 batch(es) of up to 3 rows
2025-08-21 18:17:15,456 | DEBUG | Prompt (batch 1/2):
Classify EACH transaction into EXACTLY ONE theme letter (a-j).

Themes:
a. Medical & Healthcare Supplies/Services
b. IT, Software & Hardware
c. Facilities, Utilities & Maintenance
```

```
d. Professional & Management Services
e. Financial & Contract Modifications
f. Training, Education & Outreach
g. Logistics & Support Services
h. Closeout & Administrative Actions
i. Defense, Aerospace & Mechanical Equipment
j. Other

Return ONLY lines in the exact format: <number>) <letter>
Do NOT include any other text before or after.

Items:
1) WEATHER OBSERVING STATION INCLUDING INSTALLATION, TRAINING AND WARRANTY
2) RADIOPHARMACEUTICALS FOR THE MONTANA VA HEALTH CARE SYSTEM
3) OPEN MARKET PHARMACEUTICALS ORDER
2025-08-21 18:17:16,176 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:17:16,178 | DEBUG | Raw model output (batch 1/2):
1) i
2) a
3) a
2025-08-21 18:17:18,184 | DEBUG | Prompt (batch 2/2):
Classify EACH transaction into EXACTLY ONE theme letter (a-j).

Themes:
a. Medical & Healthcare Supplies/Services
b. IT, Software & Hardware
c. Facilities, Utilities & Maintenance
d. Professional & Management Services
e. Financial & Contract Modifications
f. Training, Education & Outreach
g. Logistics & Support Services
h. Closeout & Administrative Actions
i. Defense, Aerospace & Mechanical Equipment
j. Other

Return ONLY lines in the exact format: <number>) <letter>
Do NOT include any other text before or after.

Items:
1) 4556151540!BLANKET DISP COMFORT 1
2) 4556017656!CLEANING MONITOR
3) MONTHLY LEASE CATEGORY I, II, III MFDS - ADD DFARS CLAUSE 252.223-7999. EO
14042.
2025-08-21 18:17:18,795 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:17:18,798 | DEBUG | Raw model output (batch 2/2):
1) j
```

```
2) c
3) e
2025-08-21 18:17:18,799 | INFO | Classification complete. Updated column
'Theme'.
```

Note: - The GroqThemeClassifier is now working successfully on test batches. - We will now run the classification process on the **entire dataset**.

[15]:

```python
# Run on entire dataset
classifier = GroqThemeClassifier()

# Final dataset
df_final = clf.classify_in_batches(
        df,
        text_col="transaction_description",
        theme_col="Theme",
        batch_size=50,
        max_batches=None,
        show_prompts=False,
        show_model_output=False
    )
```

```
2025-08-21 18:22:07,802 | INFO | Planned: 53 batch(es) of up to 50 rows
2025-08-21 18:22:09,916 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:12,756 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:15,556 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:18,428 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:21,221 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:24,118 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:27,136 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:29,996 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:32,866 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:35,790 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:38,714 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:41,777 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:44,617 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
```

```
2025-08-21 18:22:47,509 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:50,375 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:53,218 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:55,990 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:22:59,334 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:02,368 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:05,337 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:08,138 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:10,958 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:13,823 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:16,605 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:19,482 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:22,356 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:25,152 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:27,956 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:30,826 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:33,663 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:36,599 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:39,395 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:42,166 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:44,951 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:47,820 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:50,689 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:53,546 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
```

```
2025-08-21 18:23:56,379 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:23:59,168 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:01,958 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:04,818 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:07,997 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:10,862 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:13,660 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:16,476 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:19,357 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:22,490 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:25,281 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:28,170 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:31,030 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:33,889 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:36,853 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:39,474 | INFO | HTTP Request: POST
https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
2025-08-21 18:24:39,478 | INFO | Classification complete. Updated column
'Theme'.
```

[17]:
```
df_final
```

[17]:
```
      federal_action_obligation  total_dollars_obligated  \
0                      23116.94                 23116.94
1                      84000.00                284319.29
2                         66.00                    66.00
3                       1216.40                  1216.40
4                        259.70                   259.70
…                             …                        …
2627                   22973.12                119973.12
2628                    1008.64                  1008.64
2629                     374.70                   374.70
```

```
2630                    289.34                   289.34
2631                     26.62                    26.62

      current_total_value_of_award  potential_total_value_of_award  \
0                         23116.94                        23116.94
1                        284319.29                       599885.84
2                            66.00                           66.00
3                          1216.40                         1216.40
4                           259.70                          259.70
…                                …                               …
2627                     119973.12                       119973.12
2628                       1008.64                         1008.64
2629                        374.70                          374.70
2630                        289.34                          289.34
2631                         26.62                           26.62

      action_date_fiscal_year  funding_agency_code  award_type  \
0                        2022                   13           1
1                        2022                   36           1
2                        2022                   15           1
3                        2022                   97           0
4                        2022                   97           0
…                         …                    …           …
2627                     2023                   97           1
2628                     2022                   15           0
2629                     2022                   15           0
2630                     2022                   97           0
2631                     2022                   15           0

      type_of_contract_pricing  \
0                            0
1                            0
2                            0
3                            1
4                            1
…                          …
2627                         0
2628                         0
2629                         0
2630                         1
2631                         0

                               transaction_description  extent_competed  …  \
0     WEATHER OBSERVING STATION INCLUDING INSTALLATI…                1  …
1     RADIOPHARMACEUTICALS FOR THE MONTANA VA HEALTH…                1  …
2                    OPEN MARKET PHARMACEUTICALS ORDER                3  …
3                    4556151540!BLANKET DISP COMFORT 1                0  …
```

```
4                      4556017656!CLEANING MONITOR             0  …
…                                                 …            …  …
2627                               ELECTRONIC CONTROL            4  …
2628  FY22 MCKESSON CONTROLLED CONTRACTED  PHARMACEU…            0  …
2629  PHARMACY ORDER FOR INMATES INCARCERATED AT FCI…            0  …
2630                    4557236030!FILE K 25MM #15 6S            0  …
2631              JULY MCKESSON CONTROLLED SUBSTANCES            0  …

           revt        ib         lt         ceq     oancf       xrd  \
0     39211.000  7725.000  54146.000  40793.000  9312.000  1406.000
1        40.697   -54.454    125.427     52.413   -48.746    85.641
2       583.187     7.729    400.966    667.099   -66.537    29.307
3     12401.021   631.232   3804.587   3425.126   709.580     0.000
4     12401.021   631.232   3804.587   3425.126   709.580     0.000
…           …         …         …          …         …         …
2627     26.074     0.481      9.036     20.020     0.250     1.828
2628    583.187     7.729    400.966    667.099   -66.537    29.307
2629    583.187     7.729    400.966    667.099   -66.537    29.307
2630  12401.021   631.232   3804.587   3425.126   709.580     0.000
2631    583.187     7.729    400.966    667.099   -66.537    29.307

           cogs  psc_3digit_freq  lapse_flag  \
0     18171.000         120352.0           1
1       115.855         635345.0           1
2       118.470         635345.0           0
3      8534.570         120352.0           0
4      8534.570         120352.0           0
…           …                …           …
2627     16.821            459.0           1
2628    118.470         635345.0           0
2629    118.470         635345.0           0
2630   8534.570         463837.0           0
2631    118.470         635345.0           0

                                          Theme
0     Defense, Aerospace & Mechanical Equipment
1        Medical & Healthcare Supplies/Services
2        Medical & Healthcare Supplies/Services
3           Facilities, Utilities & Maintenance
4           Facilities, Utilities & Maintenance
…                                           …
2627                   IT, Software & Hardware
2628         Financial & Contract Modifications
2629       Medical & Healthcare Supplies/Services
2630  Defense, Aerospace & Mechanical Equipment
2631       Medical & Healthcare Supplies/Services
```

[2632 rows x 35 columns]

[18]:
```
# Check if any Theme is NaN
df_final['Theme'].isnull().sum()
```

[18]: np.int64(0)

[20]:
```
# Map each unique theme label to a numeric code.
unique_sorted = sorted(set(df_final['Theme'].dropna()))
theme_to_code_alpha = {t: i for i, t in enumerate(unique_sorted)}
df_final['ThemeCodeAlpha'] = df_final['Theme'].map(theme_to_code_alpha)
```

[23]:
```
# No need of string column now
df_final = df_final.drop(columns=["transaction_description", "Theme"],␣
 ↪errors="ignore")
```

[24]:
```
# Check if all columns are numeric
all_numeric = df_final.apply(pd.api.types.is_numeric_dtype).all()
print("All columns numeric:", all_numeric)
```

```
All columns numeric: True
```

[25]:
```
list(df_final.columns)
```

[25]:
```
['federal_action_obligation',
 'total_dollars_obligated',
 'current_total_value_of_award',
 'potential_total_value_of_award',
 'action_date_fiscal_year',
 'funding_agency_code',
 'award_type',
 'type_of_contract_pricing',
 'extent_competed',
 'government_furnished_property',
 'undefinitized_action',
 'performance_based_service_acquisition',
 'veteran_owned_business',
 'woman_owned_business',
 'minority_owned_business',
 'contracting_officers_determination_of_business_size',
 'foreign_owned',
 'for_profit_organization',
 'nonprofit_organization',
 'the_ability_one_program',
 'small_disadvantaged_business',
 'sic4',
 'at',
 'sale',
```

```
 'revt',
 'ib',
 'lt',
 'ceq',
 'oancf',
 'xrd',
 'cogs',
 'psc_3digit_freq',
 'lapse_flag',
 'ThemeCodeAlpha']
```

[26]:
```
df_final.to_csv('final_feature_eng.csv', index=False)
#df_final = pd.read_csv('final_feature_eng.csv')
```

0.0.5 Comparing Performance

We first trained simple models **before including `ThemeCodeAlpha`** as a feature. And we had:

Random Forest
- Accuracy: **0.8221**
- ROC-AUC: **0.824**
- Precision-Recall AUC: **0.771**

XGBoost
- Accuracy: **0.8322**
- ROC-AUC: **0.883**
- Precision-Recall AUC: **0.883**

[43]:
```
# Import Libraries
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import roc_curve, roc_auc_score, precision_recall_curve,␣
 ↪auc

import matplotlib.pyplot as plt

# Ignore warning
import warnings
warnings.filterwarnings("ignore")
```

[44]:
```
# Train = 2022 + 2023
train_df = df_final[df_final["action_date_fiscal_year"].isin([2022, 2023])].
 ↪copy()
```

```python
# Test = 2024
test_df = df_final[df_final["action_date_fiscal_year"] == 2024].copy()

# Remove date column
train_df = train_df.drop(columns="action_date_fiscal_year", errors="ignore")
test_df = test_df.drop(columns="action_date_fiscal_year", errors="ignore")
```

[45]:
```python
# --- Separate features and target ---
X_train = train_df.drop(columns=["lapse_flag"])
y_train = train_df["lapse_flag"]

X_test  = test_df.drop(columns=["lapse_flag"])
y_test  = test_df["lapse_flag"]
```

[46]:
```python
# 1. Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = (lr.predict(X_test) > 0.5).astype(int)
print("Linear Regression Accuracy:", accuracy_score(y_test, y_pred_lr))

# 2. Logistic Regression
logr = LogisticRegression(max_iter=1000, random_state=42)
logr.fit(X_train, y_train)
y_pred_logr = logr.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logr))

# 3. Support Vector Machine (SVM)
svm = SVC(kernel="rbf", probability=True, random_state=42)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

# 4. Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

# 5. XGBoost
xgb = XGBClassifier(
    use_label_encoder=False, eval_metric="logloss", random_state=42
)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb))

# 6. LightGBM
```

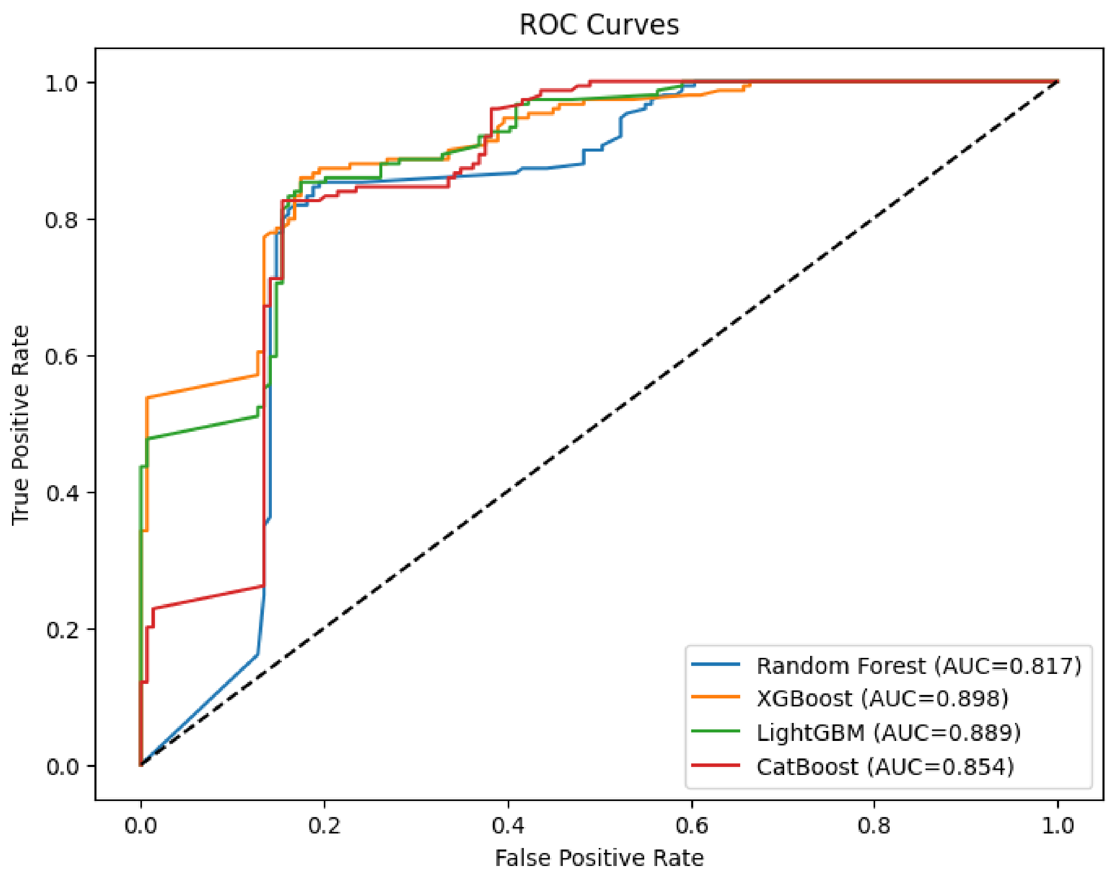
```
lgbm = LGBMClassifier(n_estimators=200, random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
print("LightGBM Accuracy:", accuracy_score(y_test, y_pred_lgbm))

# 7. CatBoost
# (silent=True suppresses training logs)
cat = CatBoostClassifier(iterations=200, random_state=42, verbose=False)
cat.fit(X_train, y_train)
y_pred_cat = cat.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, y_pred_cat))
```

```
Linear Regression Accuracy: 0.7013422818791947
Logistic Regression Accuracy: 0.6845637583892618
SVM Accuracy: 0.5100671140939598
Random Forest Accuracy: 0.825503355704698
XGBoost Accuracy: 0.8322147651006712
[LightGBM] [Info] Number of positive: 1167, number of negative: 1167
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000488 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2389
[LightGBM] [Info] Number of data points in the train set: 2334, number of used
features: 28
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
LightGBM Accuracy: 0.8355704697986577
CatBoost Accuracy: 0.8322147651006712
```

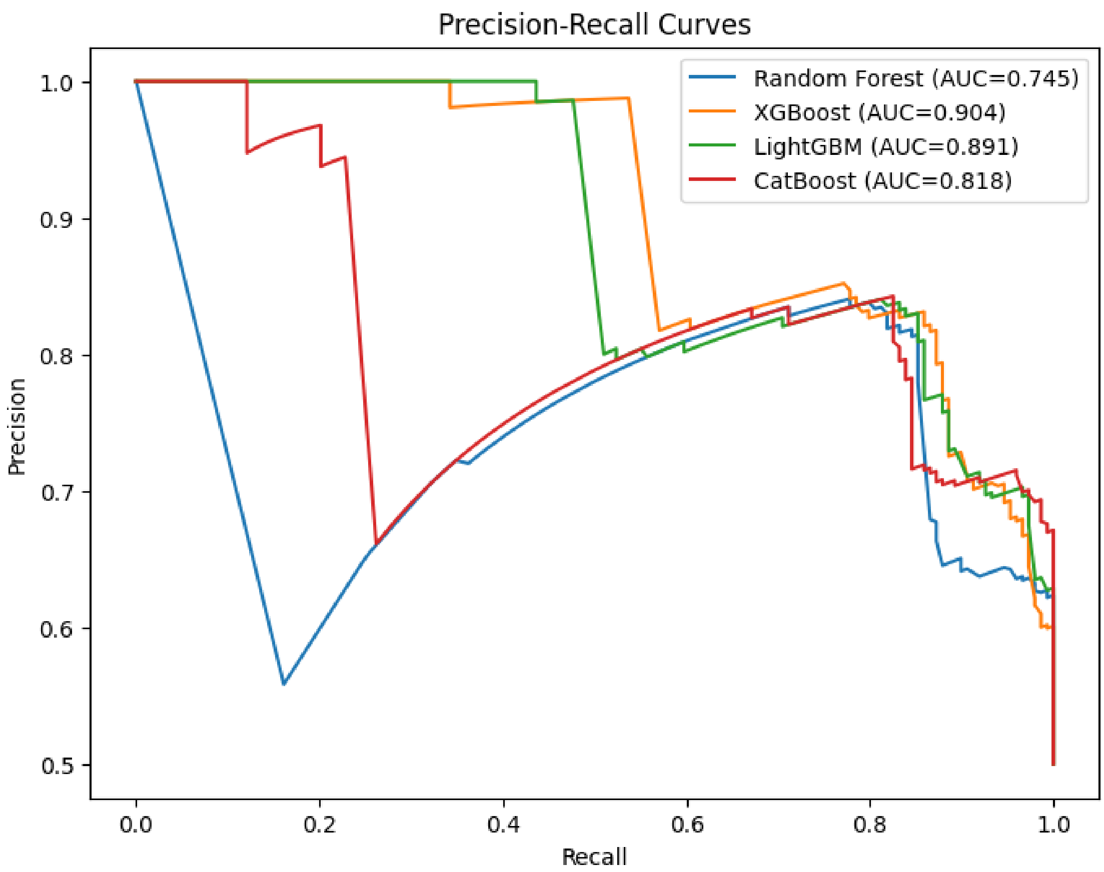
[47]:
```
models = {
    "Random Forest": rf,
    "XGBoost": xgb,
    "LightGBM": lgbm,
    "CatBoost": cat,
}

# --- ROC Curves ---
plt.figure(figsize=(8,6))
for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    elif hasattr(model, "decision_function"):
        y_proba = model.decision_function(X_test)
    else:
        y_proba = model.predict(X_test)

    # ROC
```

```python
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc_score = roc_auc_score(y_test, y_proba)
    print(f"{name} ROC AUC: {auc_score:.3f}")
    plt.plot(fpr, tpr, label=f"{name} (AUC={auc_score:.3f})")

plt.plot([0,1], [0,1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.legend()
plt.show()

# --- Precision-Recall Curves ---
plt.figure(figsize=(8,6))
for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    elif hasattr(model, "decision_function"):
        y_proba = model.decision_function(X_test)
    else:
        y_proba = model.predict(X_test)

    precision, recall, _ = precision_recall_curve(y_test, y_proba)
    pr_auc = auc(recall, precision)
    print(f"{name} Precision-Recall AUC: {pr_auc:.3f}")
    plt.plot(recall, precision, label=f"{name} (AUC={pr_auc:.3f})")

plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curves")
plt.legend()
plt.show()
```

```
Random Forest ROC AUC: 0.817
XGBoost ROC AUC: 0.898
LightGBM ROC AUC: 0.889
CatBoost ROC AUC: 0.854
```




```
Random Forest Precision-Recall AUC: 0.745
XGBoost Precision-Recall AUC: 0.904
LightGBM Precision-Recall AUC: 0.891
CatBoost Precision-Recall AUC: 0.818
```



0.1 Comparing Model Performance

After removing the introducing the **ThemeCodeAlpha** mapping from LLM classification, we compared performance with our earlier results.

0.2 Model Performance Comparison

Model	Metric	Previous (w/o ThemeCodeAlpha)	Current (with ThemeCodeAlpha)
Random Forest	Accuracy	0.8221	0.8255
	ROC-AUC	0.824	0.817
	Precision-Recall AUC	0.771	0.745
XGBoost	Accuracy	0.8322	0.8322
	ROC-AUC	0.883	0.898
	Precision-Recall AUC	0.883	0.904

0.2.1 Discussion

- Random Forest showed a **slight gain in accuracy**, but **decrease in ROC-AUC and PR-AUC**, indicating weaker probability calibration and ranking ability.

- XGBoost maintained **competitive accuracy** and improved on both **ROC-AUC** (0.898 vs 0.883) and **Precision-Recall AUC** (0.904 vs 0.883).

- This suggests that **ThemeCodeAlpha provides little meaningful additional signal**.

Next Step We will proceed with **fine-tuning XGBoost**, as it achieves the **highest AUC scores** while maintaining **competitive accuracy**, making it the best candidate for modeling contract lapse risk.