01_data_scraping_reddit

October 12, 2025

1 Step 1 — Data Scraping (Reddit)

Goal: Collect raw text data from Reddit to use in later analysis.

What this notebook does

- Connects to Reddit (via API/Pushshift or praw) with read-only access.
- Pulls posts and/or comments for selected subreddits, keywords, or time windows.
- Keeps useful fields (id, author, created_utc, score, title, selftext/body, subreddit).
- Light validation: drops obvious duplicates/empties and tags each record with a source.

Inputs: Query terms and subreddit list (set in the params cell).
Outputs: Saves a unified raw dataset to the dataset folder.
Notes: Keep API keys out of the notebook (use environment variables or a `.env` file).

1.1 Data Collection

The following are examples of possible data sources.

1.1.1 Articles

- Oklahoma schools rank 50th in the nation in latest education quality study
- Public School Rankings by State 2025
- Worst School Districts by State 2025
- 2025 Best School Districts in Massachusetts
- 2025 Best School Districts in California

Palo Alto, CA

- Henry M. Gunn High School
  - Website
  - Niche.com
  - Yelp.com
- Palo Alto Highschool
  - Website
  - Niche.com
  - Yelp.com
- Hope Technology School - Private
  - Website
  - Yelp.com
- Palo Alto Middle College High School

  - Website
  - Yelp.com

Oklahoma City, OK

- Boulevard Academy
  - Website
- Memorial High School
  - Website
  - Niche.com
  - Yelp.com
- North High School
- Santa Fe High School

1.2 Import libraries

[ ]:
```
#
try:
    IS_PIPELINE_RUN
except NameError:
    IS_PIPELINE_RUN = False

try:
    IS_PIPELINE_TEST
except NameError:
    IS_PIPELINE_TEST = False
```

[ ]:
```
import pandas as pd
import praw
import re
import sys
import time

from pathlib import Path
from datetime import datetime
```

[3]:
```
# set dataset folder location
dataset_folder = Path("../datasets")
```

1.3 Connect to Reddit API

Note: Default credentials are stored in the `praw.ini` file. This file must be created for the Reddit API to work. Use the `praw.ini.template` as a reference to create the `praw.ini` file.

[4]:
```
# default credentials stored in `praw.ini` file
try:

    reddit = praw.Reddit("default")
```

```
    print(f"Authenticated as: {reddit.user.me()}")
except Exception as e:
    print("[ERROR] Error initializing Reddit instance.")
    print("Check that the `praw.ini` is configured correctly. ")
    print(f"Details: {e}")
    sys.exit(1)

# Check for connection
# for post in subreddit.hot(limit=5):
#     print(post.title, post.score, post.id, post.url)
```

```
Authenticated as: None
```

1.4 Reddit Query Functions

[ ]:
```
def word_count(s: str) -> int:
    return len(re.findall(r"\w+", s or ""))


def comments_to_corpus(submission, min_score=6, min_words=21):
    """
    Extract and filter comments from a Reddit submission.

    Iterates through all comments in a submission and keeps only those that meet
    both a minimum score threshold and a minimum word count. Each qualifying
    comment is returned in two parallel forms:

    - "nested": [["comment1"], ["comment2"], ...]
    - "flat":   ["comment1", "comment2", ...]

    Parameters
    ----------
    submission : praw.models.Submission
        Reddit submission object from which to collect comments.
    min_score : int, optional, default=6
        Minimum upvote score required for a comment to be included.
    min_words : int, optional, default=21
        Minimum number of words required for a comment to be included.

    Returns
    -------
    dict
        Dictionary with two keys:
        - "nested": list of list of str
        - "flat":   list of str

    Notes
```

```
    -----
    - Comments with `[deleted]` or `[removed]` text are skipped.
    - Comments with no body text or missing score are excluded.
    - `submission.comments.replace_more(limit=0)` is used to ensure that
      all comments are fully loaded before filtering.
    """
    submission.comments.replace_more(limit=0)
    nested, flat = [], []

    for c in submission.comments.list():
        body = c.body
        if not isinstance(body, str):
            continue
        body = body.strip()

        if body.lower() in ("[deleted]", "[removed]"):
            continue
        if c.score is None:
            continue

        if c.score >= min_score and word_count(body) >= min_words:
            nested.append([body])
            flat.append(body)

    return {"nested": nested, "flat": flat}


def build_df_for_query(
    subreddit_name: str,
    query: str,
    limit=20,
    sort="relevance",
    time_filter="all",
    min_score=6,
    min_words=21,
):
    """
    Query Reddit submissions and build a DataFrame of post titles and filtered␣
 ↪comments.

    This function searches a specified subreddit for submissions matching a␣
 ↪query
    and collects comments from each submission that meet the given thresholds
    (minimum score and minimum word count). The comments are returned both as a
    nested form (list of single-element lists) and as a flat form (list of␣
 ↪strings).
    The result is a DataFrame where each row corresponds to one submission.
```

```
    Parameters
    ----------
    subreddit_name : str
        Name of the subreddit to search (without the "r/").
    query : str
        Search query string to use within the subreddit.
    limit : int, optional, default=20
        Maximum number of submissions to retrieve.
    sort : {"relevance", "hot", "top", "new", "comments"}, optional,␣
↪default="relevance"
        Sorting method for the search results.
    time_filter : {"all", "day", "hour", "month", "week", "year"}, optional,␣
↪default="all"
        Restrict search results to a specific time window.
    min_score : int, optional, default=6
        Minimum upvote score required for a comment to be included.
    min_words : int, optional, default=21
        Minimum number of words required for a comment to be included.

    Returns
    -------
    pandas.DataFrame
        DataFrame with one row per submission and the following columns:

        - ``source`` : str
            Constant value "reddit".
        - ``query`` : str
            The original search query string.
        - ``topic`` : str
            Title of the submission.
        - ``corpus`` : list of list of str
            Nested list of comments (each comment wrapped in a list).
        - ``flat_corpus`` : list of str
            Flat list of comments.
    """
    rows = []
    sr = reddit.subreddit(subreddit_name)

    for subm in sr.search(query, sort=sort, time_filter=time_filter,␣
↪limit=limit):
        try:
            comments = comments_to_corpus(
                subm, min_score=min_score, min_words=min_words
            )
```

```
            if comments["nested"]:  # only keep submissions with qualifying␣
 ↪comments
                rows.append(
                    {
                        "source": "reddit",
                        "query": query,
                        "topic": (subm.title or "").strip(),
                        "comments_nested": comments["nested"],
                        "comments_flat": comments["flat"],
                    }
                )

        except Exception as e:
            print(f"[warn] {subm.id}: {e}")
            time.sleep(0.2)  # be polite to the API

    return pd.DataFrame(rows)


def query_builder(district, options):
    """
    Build a Reddit search query string based on district name and topic options.

    The function constructs a search query suitable for Reddit's API by␣
 ↪combining
    the district name (quoted for exact matching) with user-selected topic␣
 ↪keywords
    joined using logical OR operators inside parentheses.

    Parameters
    ----------
    district : str
        Name of the district or area to search for (e.g., "Palo Alto").
    options : list of str
        List of topic keywords to include in the query
        (e.g., ["schools", "district", "education"]).

    Returns
    -------
    str
        Formatted Reddit search query string.
        Example:
        '"Palo Alto" (schools OR district OR education)'

    Notes
    -----
    - If `district` is empty, only the topic portion is returned.
```

```
    - If `options` is empty, only the district portion is returned.
    - Leading and trailing whitespace in `district` is automatically stripped.

    Examples
    --------
    >>> query_builder("Palo Alto", ["schools", "teachers"])
    '"Palo Alto" (schools OR teachers)'

    >>> query_builder("San Diego", [])
    '"San Diego"'
    """
    district_part = f'"{district.strip()}"' if district else ""

    # join topic options with OR inside ()
    if options:
        options_part = "(" + " OR ".join(options) + ")"
    else:
        options_part = ""

    query = f"{district_part} {options_part}".strip()

    return query


def total_word_count(comments):
    """
    Compute the total number of words across a list of comment strings.

    Parameters
    ----------
    comments : list of str
        List of comments from which to count words.

    Returns
    -------
    int
        Total number of words across all comments.
        Returns 0 if `comments` is empty or None.

    Notes
    -----
    - Each comment is split on whitespace to estimate word count.
    - Non-string entries in the list should be cleaned before calling this␣
 ↪function.

    Examples
    --------
```

```
    >>> total_word_count(["This is one comment.", "This is another."])
    7

    >>> total_word_count([])
    0
    """
    total = 0
    if not comments:
        return 0

    for comment in comments:
        words = comment.split()
        total += len(words)

    return total


def save_pickle_file(dataframe, filename, dataset_folder):
    # check if running master pipeline
    try:
        flag = IS_PIPELINE_RUN
    except NameError:
        flag = False

    # Handle dataset_folder
    if not dataset_folder:
        dataset_folder = Path.cwd()
    else:
        dataset_folder = Path(dataset_folder)

    # remove spaces in filename
    filename = filename.replace(" ", "_")

    # ensure folder exists
    dataset_folder.mkdir(parents=True, exist_ok=True)

    if flag:
        full_path = dataset_folder / f"{filename}_pipeline_reddit.pkl"
    else:
        # create timestamp
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        # full path
        full_path = dataset_folder / f"{filename}_{timestamp}_reddit.pkl"

    dataframe.to_pickle(full_path)
    print(f"Saved as {filename}. ")
```

```
    return full_path


def query_and_save(list_of_districts):
    for district in list_of_districts:
        query = query_builder(district=district, options=query_options)
        df = build_df_for_query(
            subreddit_name=subreddit_var,
            query=query,
            limit=LIMIT,
            sort="relevance",
            time_filter="all",
            min_words=MIN_WORD,
            min_score=MIN_SCORE,
        )
        df["num_comments"] = df["comments_flat"].apply(len)
        df["total_words"] = df["comments_flat"].apply(total_word_count)
        # df.head(5)
        file_path = save_pickle_file(
            dataframe=df, filename=district, dataset_folder=dataset_folder
        )
        print(file_path)
```

1.4.1 Variables for Reddit Query

[ ]:
```
MIN_WORD = 10  # default = 6
MIN_SCORE = 5  # default = 21
# LIMIT = 5  # default = 20
if IS_PIPELINE_TEST:
    LIMIT = 10
else:
    LIMIT = 150

subreddit_var = "all"
query_options = [
    "school",
    "schools",
    "district",
    "education",
    "homework",
    "teacher",
    "teachers",
    "student",
    "students",
]
```

1.5 Query and Save Dataframes

[16]:
```
districts = ["Palo Alto", "Oklahoma City"]

query_and_save(districts)
```

```
Saved as Palo Alto.
../datasets/Palo Alto_20251007_235943_reddit.pkl
Saved as Oklahoma City.
../datasets/Oklahoma City_20251008_000300_reddit.pkl
```