Listening Beyond the Lab: Patient Voices on Cancer Care and Journey

LT1

Acosta, Maynard Anthony
Borromeo, Chloe
Garcia, Fernando
Tan, Allan

Introduction

Tony Boy's Data Company – Project Overview

Oncolens is a hypothetical project commissioned to Tony Boy's Data Company by a pharmaceutical client seeking deeper insights into patient behaviors and perceptions surrounding their products. For this initiative, the focus is on olaparib, a PARP inhibitor prescribed for the treatment of ovarian, breast, pancreatic, and prostate cancers.

Tony Boy's Data Company is committed to delivering high-quality, data-driven outputs at affordable rates. By providing actionable healthcare insights, the company supports clients in enhancing patient journeys and making more informed, patient-centered decisions.

Use Case: Social Listening for Target Drug Products - Olaparib

Challenge
Cancer patients and their caregivers frequently discuss treatments, side effects, costs, and emotional experiences on platforms like **Reddit**. These discussions contain valuable but unstructured insights that can help a pharmaceutical company:

- Understand how patients **perceive olaparib**
- Track **sentiment trends** across specific cancer communities (e.g., ovarian vs breast cancer subreddits).

Proposed Solution
Tony Boy's Data Company will develop a **data architecture and analytics platform** that systematically scrapes and analyzes Reddit discussions from carefully chosen subreddits. Using natural language processing (NLP), sentiment analysis, and topic modeling, the platform will transform raw posts into actionable insights.

The insights will support:

- **Medical Affairs** – understanding patient and physician sentiment.
- **Commercial Teams** – benchmarking against competitor medications.
- **Pharmacovigilance** – monitoring for signals of adverse drug experiences.

Design Considerations for Oncolens Social Listening System

1. Data Sources & Coverage

- **Platform Selection:** Start with Reddit (targeted subreddits like r/cancer, r/breastcancer, r/medicine), but design for extensibility to include Twitter/X, Inspire, or other patient forums.
- **Data Completeness:** Use both Reddit API (official, limited history) and for future improvements, Pushshift archives (historical depth).
- **Content Types:** Handle posts, comments, and metadata (scores, upvotes, time posted).
- **Justification:** Reddit hosts active cancer communities where patients, caregivers, and medical professionals discuss real-world treatment experiences. Compared to clinical trial data, these conversations provide unsolicited, candid insights on olaparib and broader oncology themes.
- **Design Implication:** Focusing on Reddit ensures the system starts with high-signal, oncology-specific discussions before expanding to other platforms (Twitter, Inspire, etc.).

2. Data Ingestion & Pipeline Design

- **Batch:** Begin with scheduled batch scraping (hourly/ daily) to manage volume.
- **Pipeline Orchestration:** Use Airflow for scheduling, retries, and monitoring ingestion.
- **Storage Format:** Raw JSON into object storage S3, converted to parquet for query efficiency.
- **Justification:** Ingesting and processing Reddit data requires automation, retries, and scheduling. Clinical and commercial teams expect regular updates (hourly/weekly), and ad hoc scripts are unreliable.
- **Design Implication:** Airflow provides end-to-end pipeline control (scraping → cleaning → NLP → storage), ensuring operational stability and monitoring of failures.

3. Data Processing & Cleaning

- **Normalization:** Standardize schema (as described in Schema Section)
- **Noise Reduction:** Detect and remove spam, bots, duplicate posts.
- **Ethical Handling:** Reddit does not have any personal identifiers in most cases

4. NLP & Analytics Layer

- **Sentiment Analysis:** Use domain-adapted NLP models - RoBERTa fine-tuned.
- **Rule Based Tagging:** Tagging posts and comments based on a list of pre-selected key words
- **Justification:** Patient discussions contain nuanced sentiment (“worked well but nausea was unbearable”).
- **Design Implication:**
    - **Sentiment analysis** quantifies patient perception trends over time.

5. Data Storage & Querying

- **Data Lake Zones:**
    - *Bronze:* Raw JSON logs.
    - *Silver:* Cleaned, structured tables.
    - *Gold:* Analytics-ready aggregates

    - *Marts:* Parquet ready for serving to the API
    - *Temps:* Serves as the work/ dev folder
- **Warehouse Choice:** Redshift for Streamlit dashboard.
- **Justification:** Cancer-related discourse is unstructured, noisy, and full of irrelevant content. A **multi-zone lake** enforces a clean pipeline:
    - *Bronze* preserves raw Reddit JSON for traceability.
    - *Silver* standardizes posts and comments into tabular form for analysis.
    - *Gold* stores analytics-ready aggregates (sentiment, topics, drug mentions).
    - *Mart* provides the files that are ready for use
    - *Temp* serves as the working folder
- **Design Implication:** This separation provides auditability and reproducibility, which is crucial in a regulated pharma environment where results may need to be justified to compliance teams.

6. Visualization & Reporting

- **Dashboards:** Streamlit
- **Key Views:**
    - Hourly trends in subreddit subscribers and active users
    - Daily sentiment analysis of subreddit comments
    - Daily sentiment analysis of subreddit posts
    - Daily sentiment tracking for pre-selected keywords
- **Export Options:** Automated report generation (CSV / PNG for plots)
- **Justification:** Analysts, researchers, and executives require fast, structured querying and dashboards. A warehouse complements the lake by providing SQL-accessible, governed datasets optimized for Streamlit
- **Design Implication:** Allows integration with Streamlit, making insights accessible to non-technical stakeholders while keeping the underlying raw data intact.

7. Security, Governance, and Compliance

- **PHAP, Data Privacy Act, DOH and FDA Considerations:** While Reddit is public, treat data with ethical care — anonymize users, avoid storing PII.
- **Access Control**
    - Role-based access
        - Data Steward: read access on all zones
        - Data Engineer: read and write access on all zones (excluding sensitive zone)
        - Data Scientist: read and write access on temp / work zone and mart
        - Business Analyst: read access on gold / mart access
- **Auditability:** Full logs of scraping and processing for reproducibility in Airflow.
- **Justification:** Even though Reddit data is public, pharma companies face ethical obligations (as regulated by PHAP, DOH, and FDA). Mishandling patient-like data could pose reputational risks.
- **Design Implication:**
    - Role-based access control ensures that the correct teams are able to read and write in the appropriate zones

8. Scalability & Extensibility

- **Storage:** S3 possesses seamless scalability

- **Future Data Sources:** Extend pipeline to include PubMed abstracts, FDA adverse event reports, and Twitter.
- **Justification:** Data volumes from Reddit are moderate initially, but expanding to other platforms (Twitter, forums, PubMed) could grow quickly. Pharma research often requires longitudinal analysis (years of discussions).
- **Design Implication:**
    - Object storage S3 ensures cheap, elastic storage for unbounded history.

9. Operational Monitoring

- **Data Quality Checks:** Included in the ELT process for deduplication
- **Pipeline Health:** Airflow dashboards for latency and error rates.
- **Justification:** A pharma-grade system must **prove reliability**; missed data or silent pipeline failures can lead to biased insights.
- **Design Implication:** Airflow provides **adequate observability**: ingestion health and processing latency. This gives stakeholders confidence in the **integrity of the insights**.

Data Architecture

1. Objectives

- **Scrape Reddit posts** from cancer-related subreddits (e.g., r/cancer, r/breastcancer, r/ovariancancer, r/askdocs, r/medicine, r/pharmacy, etc.).
- **Track sentiment** around olaparib (brand: Lynparza)
- **Identify themes** in patient experiences (side effects, efficacy, affordability).
- **Provide dashboards** for medical team, marketers, and executives.

2. Pipeline (orchestrated by Apache Airflow)

1. Extract post + comment metadata from Reddit API
2. Store raw JSON in **Amazon S3 (bronze zone)**
3. Log ingestion in **Postgres RDS** for monitoring and retries.
4. Convert into parquet, normalize data types, deduplication and partition into entity and datetime **(silver zone)**
5. Run sentiment analysis and tag post and comments -> output stored in **S3 gold zone**
6. Resume Redshift cluster from snapshot
7. Load query ready data in **Redshift data warehouse**
8. Unloading aggregates into S3 marts
9. Create new snapshot -> pause cluster
10. Store subreddit metrics directly into **Amazon DynamoDB** for faster availability
11. Serving API gateway to **Flask**
12. Consume by Streamlit Dashboard

3. ELT Job Frequency

- Daily frequency: post, comments and tags

4. Architecture Justification

- **Object store (S3) as system of record:**
  - Having different folders with specific access permissions ensures cleanliness and efficiency in the pipeline
- **Postgre RDS**
  - Logging ingestion in Postgres RDS creates a reliable, queryable control plane for monitoring, dedupe, and retries—backed by ACID guarantees and enterprise-grade recovery—without burdening the analytics warehouse.
- **Redshift as serving DW:**
  - SQL-friendly, snapshot/pause pattern keeps **costs down and reproducibility high** (freeze a point-in-time with snapshot → analyze → pause).
- **DynamoDB for fast-changing counters** (subreddit subscribers/actives):
  - Millisecond reads for the "Key Views" time series that update hourly, without waking a cluster.
- **Airflow orchestration:**
  - Single pane for retries, SLAs, and lineage (ingest → clean → NLP → load). Business users can trust freshness and engineers get granular failure visibility.
- **API Gateway + Flask**:
  - A stable contract between data and apps; Streamlit (and any future tool) hits the same API, avoiding tight coupling to warehouse schemas.

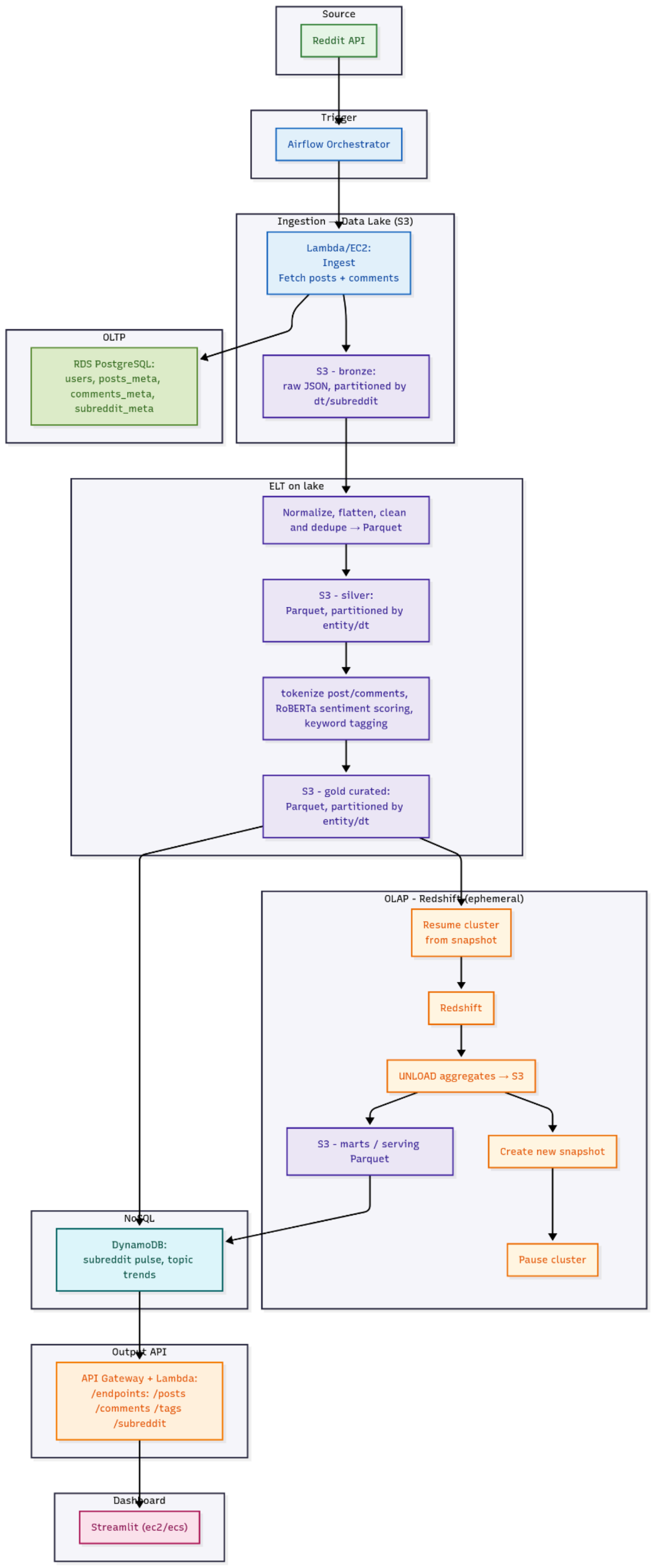
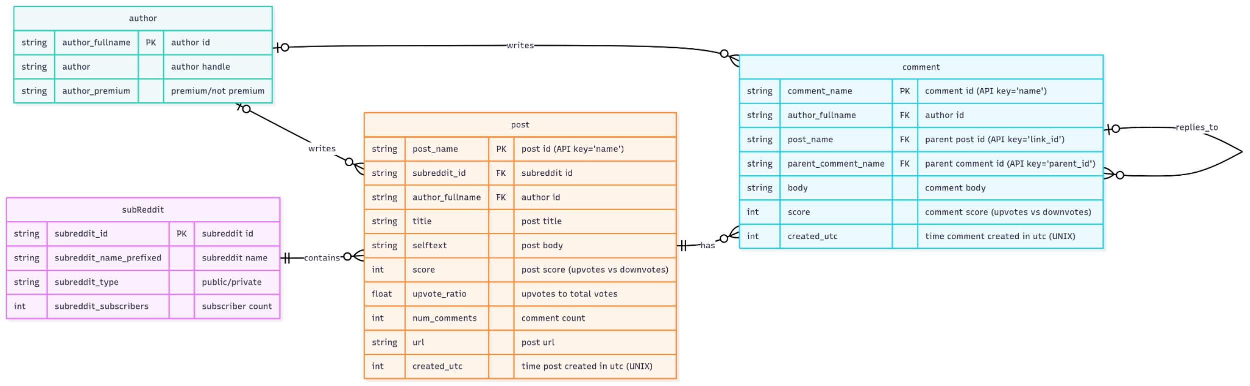

Source
Reddit API
Trigger
Airflow Orchestrator
Ingestion → Data Lake (S3)
Lambda/EC2: Ingest Fetch posts + comments
OLTP
RDS PostgreSQL: users, posts_meta, comments_meta, subreddit_meta
S3 - bronze: raw JSON, partitioned by dt/subreddit
ELT on lake
Normalize, flatten, clean and dedupe → Parquet
S3 - silver: Parquet, partitioned by entity/dt
tokenize post/comments, RoBERTa sentiment scoring, keyword tagging
S3 - gold curated: Parquet, partitioned by entity/dt
OLAP - Redshift (ephemeral)
Resume cluster from snapshot
Redshift
UNLOAD aggregates → S3
S3 - marts / serving Parquet
Create new snapshot
Pause cluster
NoSQL
DynamoDB: subreddit pulse, topic trends
Output API
API Gateway + Lambda: /endpoints: /posts /comments /tags /subreddit
Dashboard
Streamlit (ec2/ecs)

Figure 1. Data Architecture Diagram.

Schemas



author			
string	author_fullname	PK	author id
string	author		author handle
string	author_premium		premium/not premium

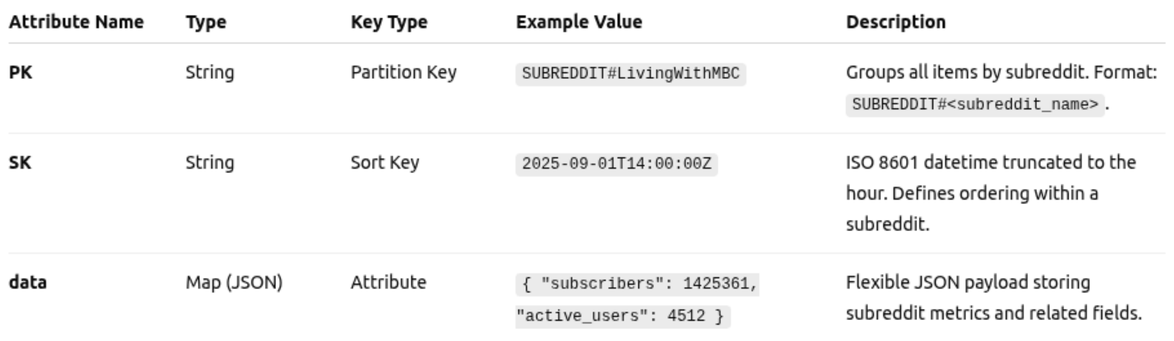
subReddit			
string	subreddit_id	PK	subreddit id
string	subreddit_name_prefixed		subreddit name
string	subreddit_type		public/private
int	subreddit_subscribers		subscriber count

post			
string	post_name	PK	post id (API key='name')
string	subreddit_id	FK	subreddit id
string	author_fullname	FK	author id
string	title		post title
string	selftext		post body
int	score		post score (upvotes vs downvotes)
float	upvote_ratio		upvotes to total votes
int	num_comments		comment count
string	url		post url
int	created_utc		time post created in utc (UNIX)

comment			
string	comment_name	PK	comment id (API key='name')
string	author_fullname	FK	author id
string	post_name	FK	parent post id (API key='link_id')
string	parent_comment_name	FK	parent comment id (API key='parent_id')
string	body		comment body
int	score		comment score (upvotes vs downvotes)
int	created_utc		time comment created in utc (UNIX)

Figure 2. OLTP ERD. The schema consists of 4 tables - author, subReddit, post and comment.

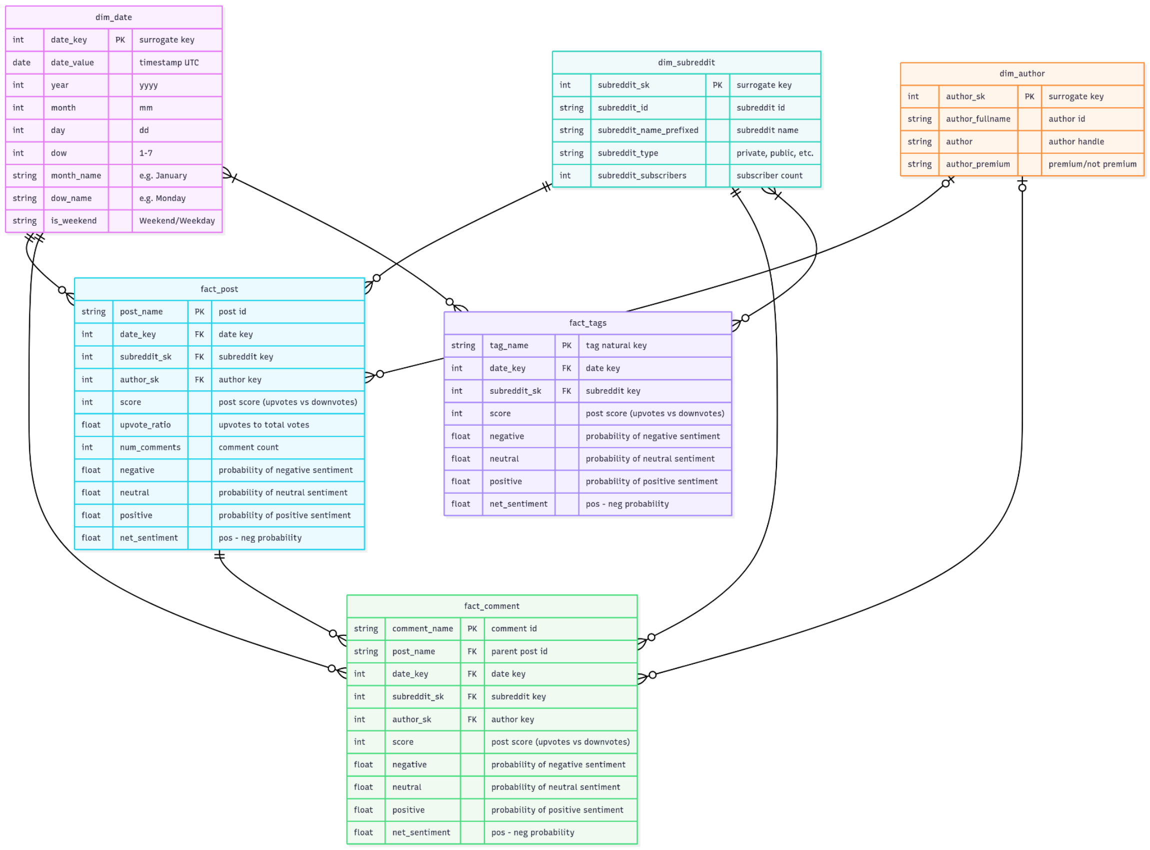
The OLTP schema is designed to capture Reddit data in its most normalized, operational form, ensuring efficient ingestion, deduplication, and traceability back to the original API payloads. Separating entities like `author`, `subReddit`, `post`, and `comment` minimizes redundancy and enforces relational integrity, allowing reliable joins and lineage tracking. By modeling relationships explicitly (e.g., posts written by authors, comments linked to both posts and parent comments), this schema preserves the threaded structure of discussions and provides a clean, auditable foundation for downstream transformations. It balances flexibility (storing text bodies, metadata, scores) with constraints that guarantee data quality, making it ideal as a staging/transactional layer before analytical processing.



Attribute Name	Type	Key Type	Example Value	Description
PK	String	Partition Key	`SUBREDDIT#LivingWithMBC`	Groups all items by subreddit. Format: `SUBREDDIT#<subreddit_name>`.
SK	String	Sort Key	`2025-09-01T14:00:00Z`	ISO 8601 datetime truncated to the hour. Defines ordering within a subreddit.
data	Map (JSON)	Attribute	`{ "subscribers": 1425361, "active_users": 4512 }`	Flexible JSON payload storing subreddit metrics and related fields.

Figure 3. DynamoDB Schema. The table provides sample values for the following Keys and Attributes

The DynamoDB schema complements the OLTP/OLAP layers by focusing on high-velocity, time-series counters—subscriber counts and active users updated hourly. Using a partition key (subreddit name) and sort key (timestamp in ISO 8601) enables efficient time-ordered queries for a given community, while a flexible JSON `data` attribute supports extensibility (e.g., adding engagement rate or growth deltas). This NoSQL design is motivated by the need for millisecond read/write performance and cost efficiency for metrics that change frequently and drive real-time dashboard tiles. Unlike Redshift or S3, DynamoDB handles streaming-style updates without cluster overhead, making it the right tool for fast-changing operational metrics while leaving historical and heavy analytics to the warehouse and data lake.



dim_date

int	date_key	PK	surrogate key
date	date_value		timestamp UTC
int	year		yyyy
int	month		mm
int	day		dd
int	dow		1-7
string	month_name		e.g. January
string	dow_name		e.g. Monday
string	is_weekend		Weekend/Weekday

dim_subreddit

int	subreddit_sk	PK	surrogate key
string	subreddit_id		subreddit id
string	subreddit_name_prefixed		subreddit name
string	subreddit_type		private, public, etc.
int	subreddit_subscribers		subscriber count

dim_author

int	author_sk	PK	surrogate key
string	author_fullname		author id
string	author		author handle
string	author_premium		premium/not premium

fact_post

string	post_name	PK	post id
int	date_key	FK	date key
int	subreddit_sk	FK	subreddit key
int	author_sk	FK	author key
int	score		post score (upvotes vs downvotes)
float	upvote_ratio		upvotes to total votes
int	num_comments		comment count
float	negative		probability of negative sentiment
float	neutral		probability of neutral sentiment
float	positive		probability of positive sentiment
float	net_sentiment		pos - neg probability

fact_tags

string	tag_name	PK	tag natural key
int	date_key	FK	date key
int	subreddit_sk	FK	subreddit key
int	score		post score (upvotes vs downvotes)
float	negative		probability of negative sentiment
float	neutral		probability of neutral sentiment
float	positive		probability of positive sentiment
float	net_sentiment		pos - neg probability

fact_comment

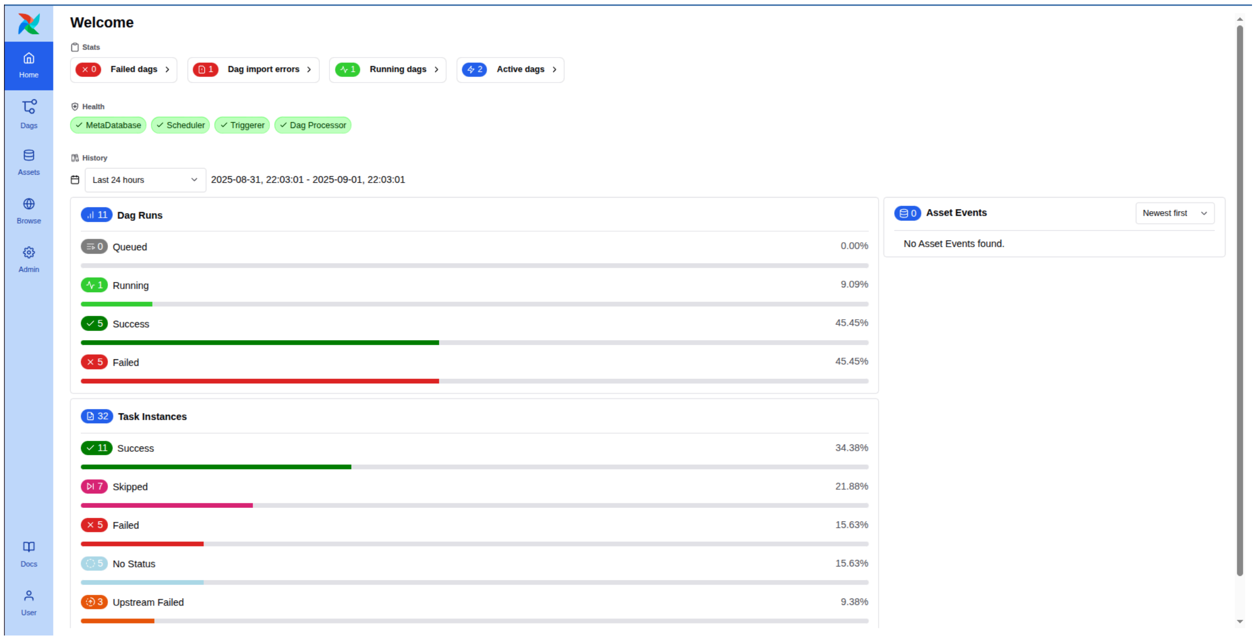
string	comment_name	PK	comment id
string	post_name	FK	parent post id
int	date_key	FK	date key
int	subreddit_sk	FK	subreddit key
int	author_sk	FK	author key
int	score		post score (upvotes vs downvotes)
float	negative		probability of negative sentiment
float	neutral		probability of neutral sentiment
float	positive		probability of positive sentiment
float	net_sentiment		pos - neg probability

Figure 4. OLAP ERD. The schema consists of 3 fact tables - fact_post, fact_tags and fact_comment and 3 dimension tables - dim_date, dim_subreddit and dim_author.

The OLAP star schema restructures raw Reddit interactions into analytics-ready aggregate**s**, optimizing for fast queries and reproducibility. Dimension tables (`dim_date`, `dim_subreddit`, `dim_author`) centralize context like time, community attributes, and user roles, ensuring consistency across analyses. Fact tables (`fact_post`, `fact_comment`, `fact_tags`) capture the measurable events at a defined grain (one row per post, comment, or tagged keyword per day), enriched with sentiment probabilities and engagement metrics. This separation allows analysts to drill down by time, subreddit, or author cohort while keeping heavy NLP features pre-computed. By enforcing clear grains and surrogate keys, the OLAP design prevents double-counting and provides a stable foundation for dashboards and KPIs like daily sentiment trends and keyword tracking

Screenshots

Airflow Screenshots



Welcome

Stats

0 Failed dags | 1 Dag import errors | 1 Running dags | 2 Active dags

Health

✓ MetaDatabase ✓ Scheduler ✓ Triggerer ✓ Dag Processor

History

Last 24 hours 2025-08-31, 22:03:01 - 2025-09-01, 22:03:01

11 Dag Runs

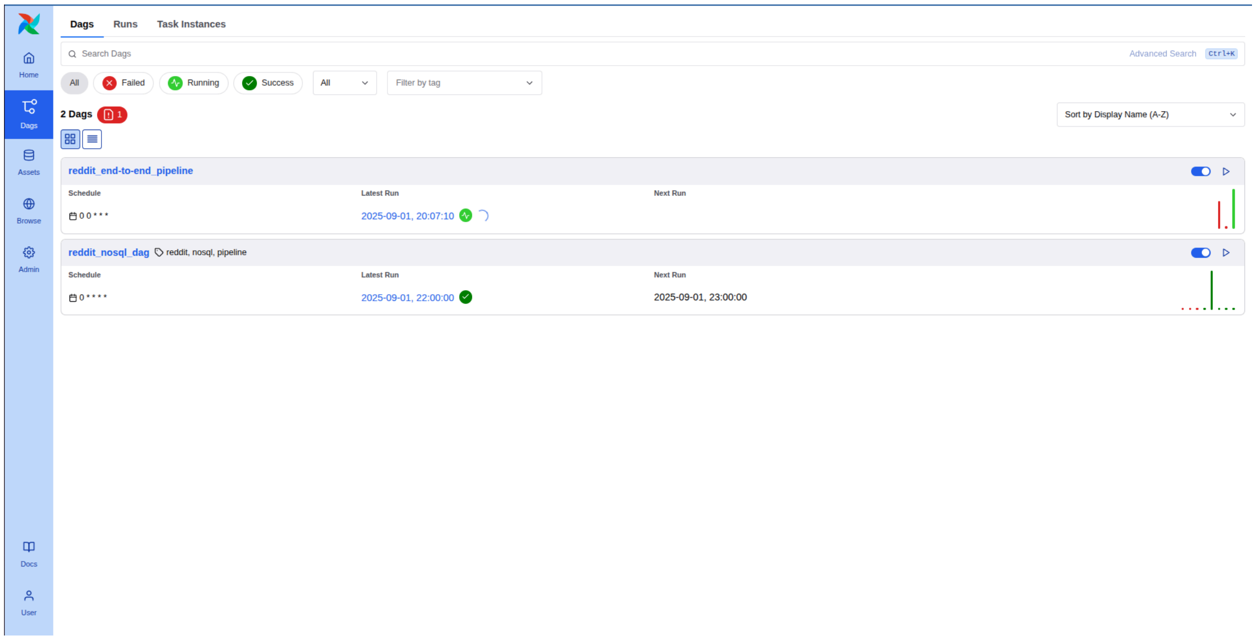
0 Queued	0.00%
1 Running	9.09%
5 Success	45.45%
5 Failed	45.45%

0 Asset Events Newest first

No Asset Events found.

32 Task Instances

11 Success	34.38%
7 Skipped	21.88%
5 Failed	15.63%
5 No Status	15.63%
3 Upstream Failed	9.38%



2 Dags 1

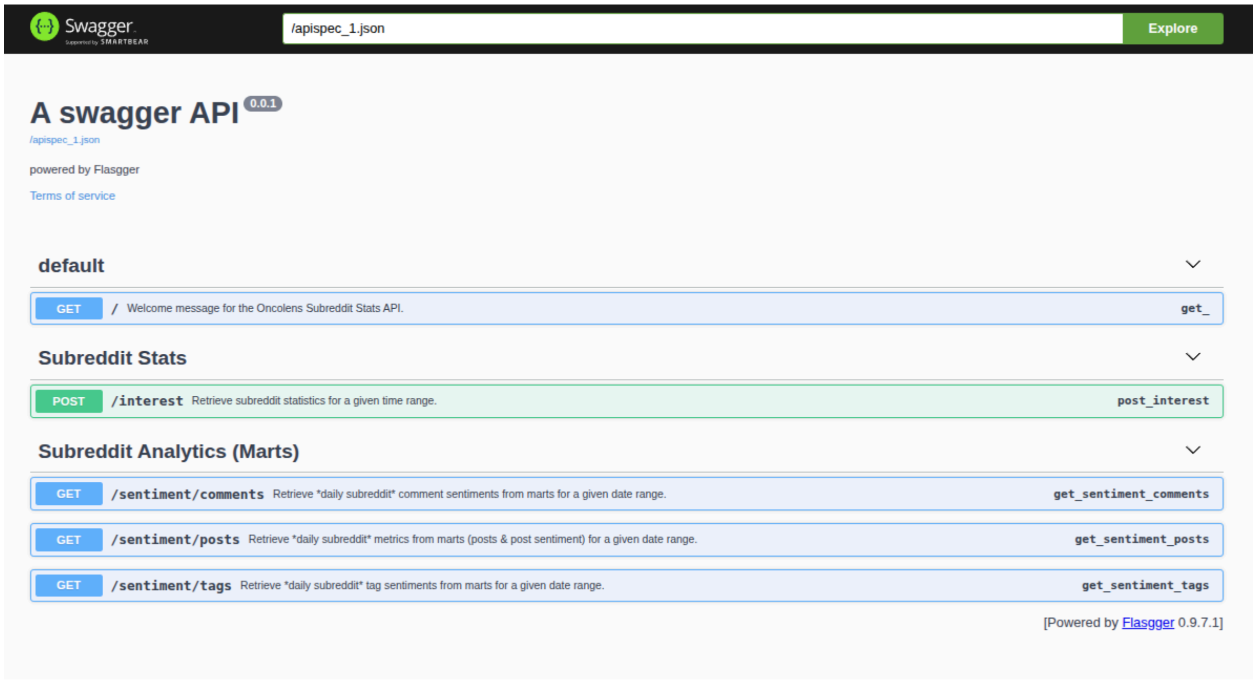
reddit_end-to-end_pipeline

Schedule	Latest Run	Next Run
0 0 * * *	2025-09-01, 20:07:10	

reddit_nosql_dag reddit, nosql, pipeline

Schedule	Latest Run	Next Run
0 * * * *	2025-09-01, 22:00:00	2025-09-01, 23:00:00

API Doc



Swagger supported by SMARTBEAR

/apispec_1.json

Explore

A swagger API 0.0.1

/apispec_1.json

powered by Flasgger

Terms of service

default

GET / Welcome message for the Oncolens Subreddit Stats API. get_

Subreddit Stats

POST /interest Retrieve subreddit statistics for a given time range. post_interest

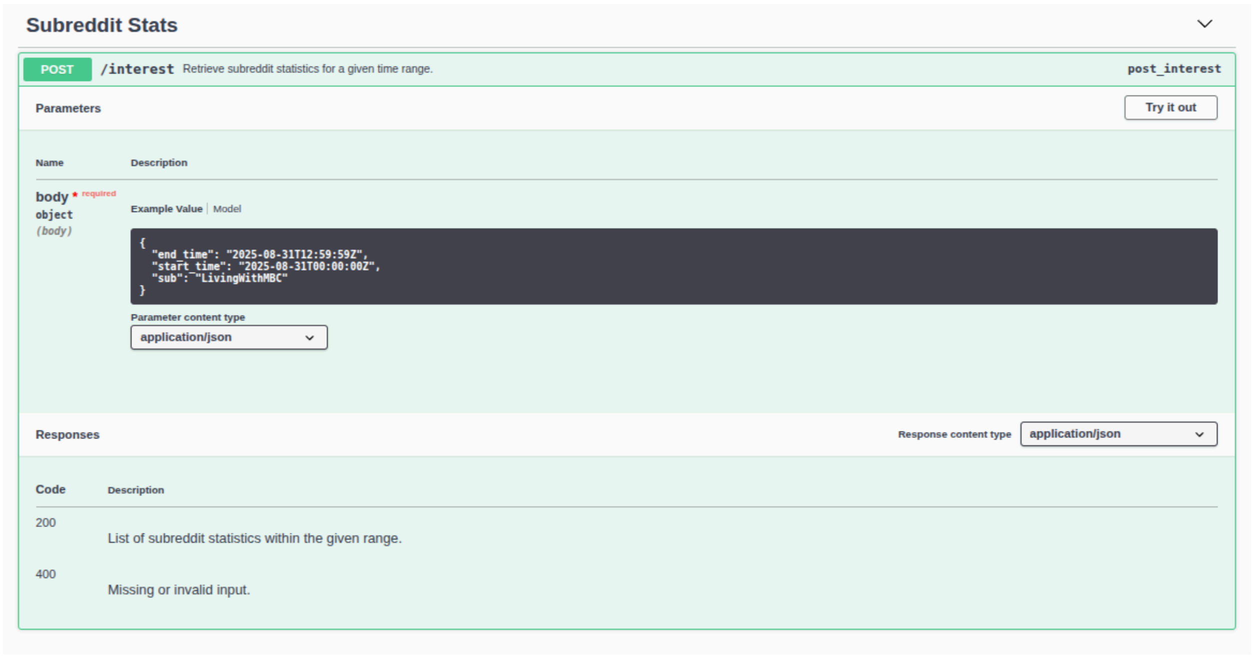
Subreddit Analytics (Marts)

GET /sentiment/comments Retrieve *daily subreddit* comment sentiments from marts for a given date range. get_sentiment_comments

GET /sentiment/posts Retrieve *daily subreddit* metrics from marts (posts & post sentiment) for a given date range. get_sentiment_posts

GET /sentiment/tags Retrieve *daily subreddit* tag sentiments from marts for a given date range. get_sentiment_tags

[Powered by Flasgger 0.9.7.1]



Subreddit Stats

POST /interest Retrieve subreddit statistics for a given time range. post_interest

Parameters

Try it out

Name	Description
body * required object *(body)*	Example Value | Model

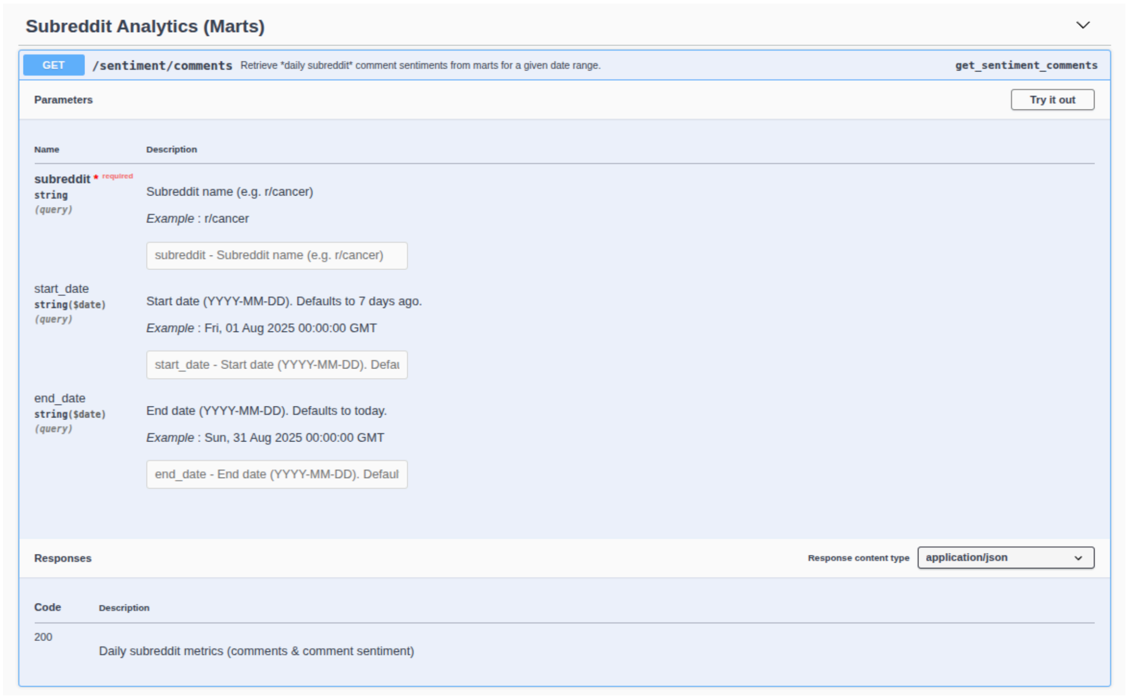
```
{
  "end_time": "2025-08-31T12:59:59Z",
  "start_time": "2025-08-31T00:00:00Z",
  "sub": "LivingWithMBC"
}
```

Parameter content type: application/json

Responses

Response content type: application/json

Code	Description
200	List of subreddit statistics within the given range.
400	Missing or invalid input.



Subreddit Analytics (Marts)

GET `/sentiment/comments` Retrieve *daily subreddit* comment sentiments from marts for a given date range. `get_sentiment_comments`

Parameters

Try it out

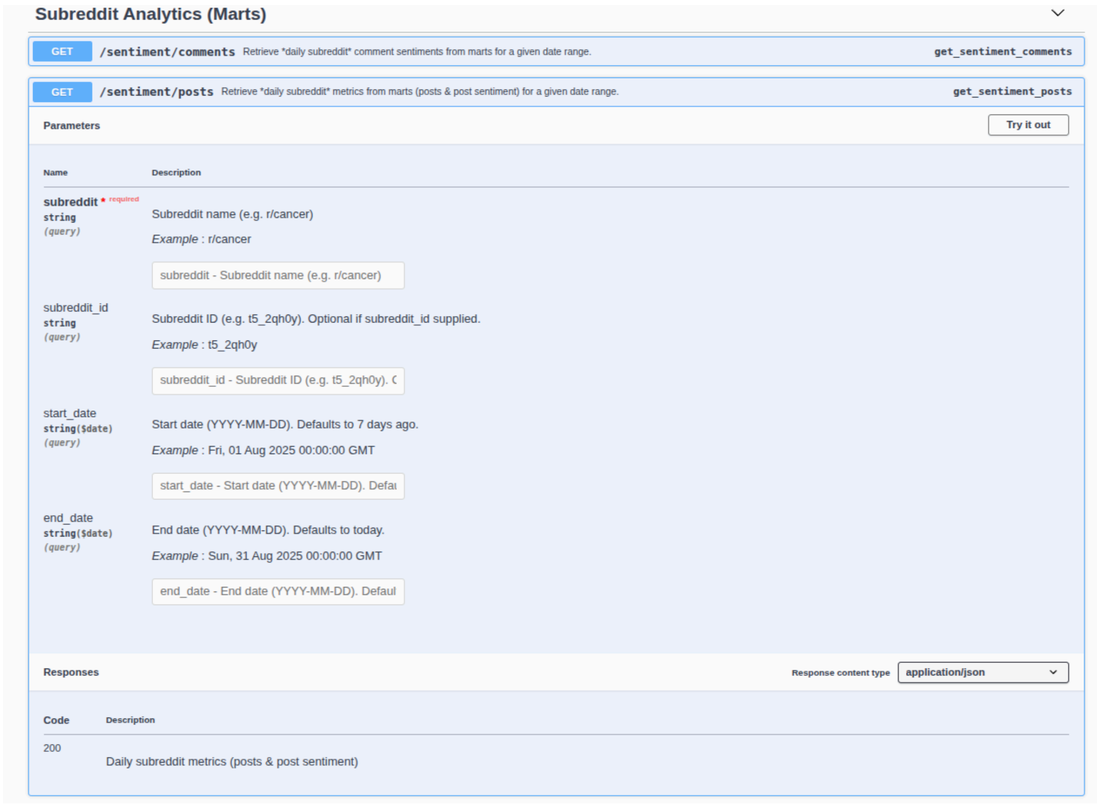
Name	Description
subreddit * required `string` *(query)*	Subreddit name (e.g. r/cancer) *Example* : r/cancer subreddit - Subreddit name (e.g. r/cancer)
start_date `string($date)` *(query)*	Start date (YYYY-MM-DD). Defaults to 7 days ago. *Example* : Fri, 01 Aug 2025 00:00:00 GMT start_date - Start date (YYYY-MM-DD). Defau

end_date
`string($date)`
(query)
	End date (YYYY-MM-DD). Defaults to today.
	Example : Sun, 31 Aug 2025 00:00:00 GMT
	end_date - End date (YYYY-MM-DD). Defaul

Responses

Response content type: application/json

Code	Description
200	Daily subreddit metrics (comments & comment sentiment)



Subreddit Analytics (Marts)

GET `/sentiment/comments` Retrieve *daily subreddit* comment sentiments from marts for a given date range. `get_sentiment_comments`

GET `/sentiment/posts` Retrieve *daily subreddit* metrics from marts (posts & post sentiment) for a given date range. `get_sentiment_posts`

Parameters

Try it out

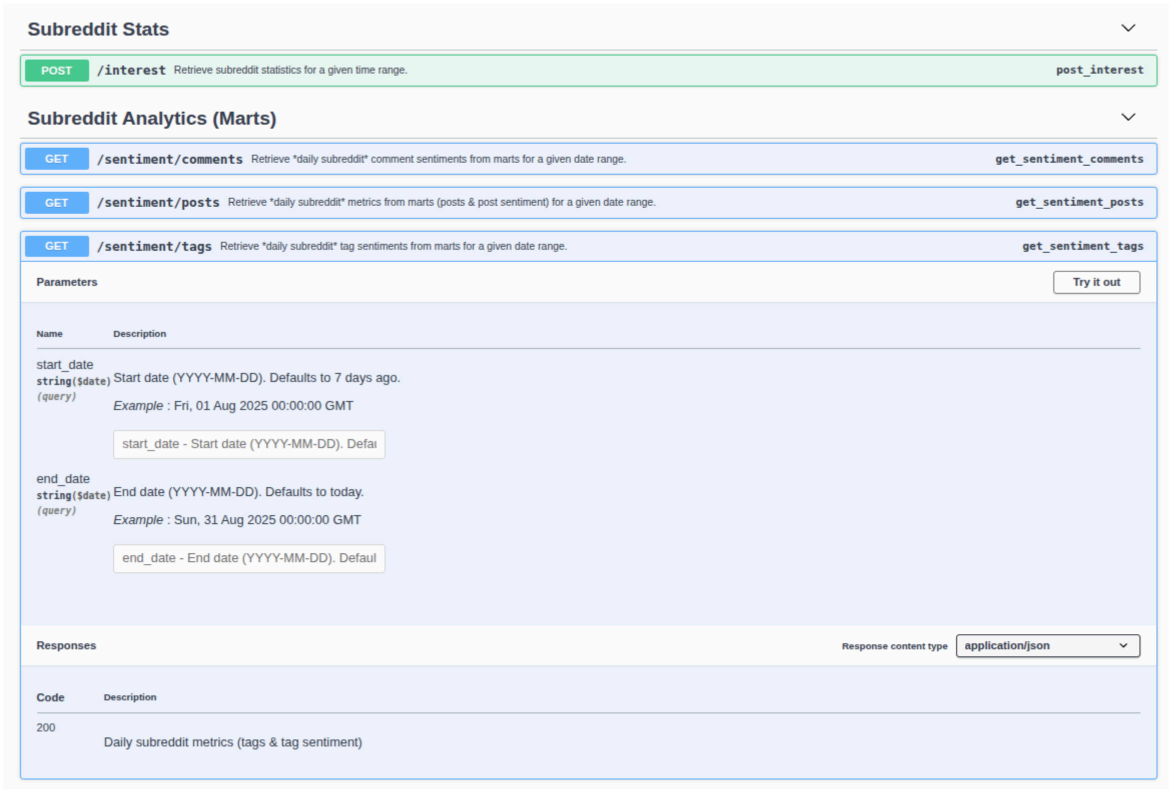
Name	Description
subreddit * required `string` *(query)*	Subreddit name (e.g. r/cancer) *Example* : r/cancer subreddit - Subreddit name (e.g. r/cancer)
subreddit_id `string` *(query)*	Subreddit ID (e.g. t5_2qh0y). Optional if subreddit_id supplied. *Example* : t5_2qh0y subreddit_id - Subreddit ID (e.g. t5_2qh0y). C
start_date `string($date)` *(query)*	Start date (YYYY-MM-DD). Defaults to 7 days ago. *Example* : Fri, 01 Aug 2025 00:00:00 GMT start_date - Start date (YYYY-MM-DD). Defau

end_date
`string($date)`
(query)
	End date (YYYY-MM-DD). Defaults to today.
	Example : Sun, 31 Aug 2025 00:00:00 GMT
	end_date - End date (YYYY-MM-DD). Defaul

Responses

Response content type: application/json

Code	Description
200	Daily subreddit metrics (posts & post sentiment)



Subreddit Stats

POST `/interest` Retrieve subreddit statistics for a given time range. `post_interest`

Subreddit Analytics (Marts)

GET `/sentiment/comments` Retrieve *daily subreddit* comment sentiments from marts for a given date range. `get_sentiment_comments`

GET `/sentiment/posts` Retrieve *daily subreddit* metrics from marts (posts & post sentiment) for a given date range. `get_sentiment_posts`

GET `/sentiment/tags` Retrieve *daily subreddit* tag sentiments from marts for a given date range. `get_sentiment_tags`

Parameters

Name	Description
start_date **string**($date) *(query)*	Start date (YYYY-MM-DD). Defaults to 7 days ago. *Example* : Fri, 01 Aug 2025 00:00:00 GMT start_date - Start date (YYYY-MM-DD). Defa
end_date **string**($date) *(query)*	End date (YYYY-MM-DD). Defaults to today. *Example* : Sun, 31 Aug 2025 00:00:00 GMT end_date - End date (YYYY-MM-DD). Defaul

Responses

Response content type: application/json

Code	Description
200	Daily subreddit metrics (tags & tag sentiment)