Hitlog Processing

-Pulkit Dhingra

Overview

- This project analyzes hitlog data (user page views and registration events) to determine which articles are most influential in driving user registrations. The system tracks user journeys through a website and counts how many unique users viewed each article before registering.

Key Features:

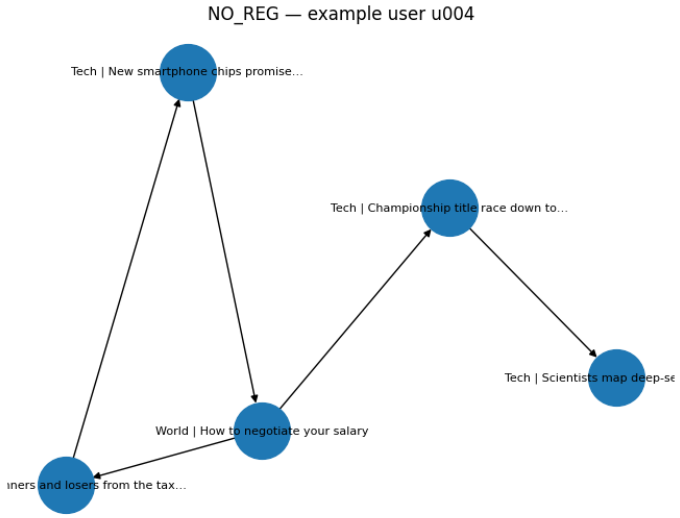
- Two distinct algorithmic approaches (counter-based and graph-based)
- Comprehensive test suite
- CLI tool for production use
- Jupyter notebooks for exploratory data analysis
- Automated code quality checks (ruff linting and formatting)

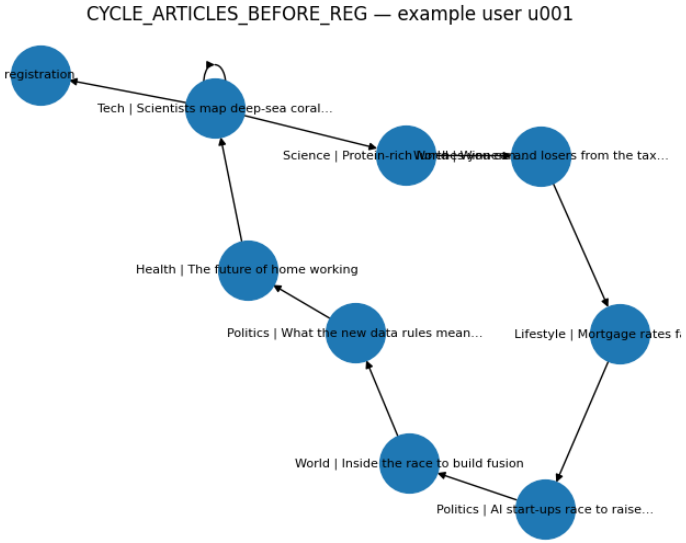
About the Data

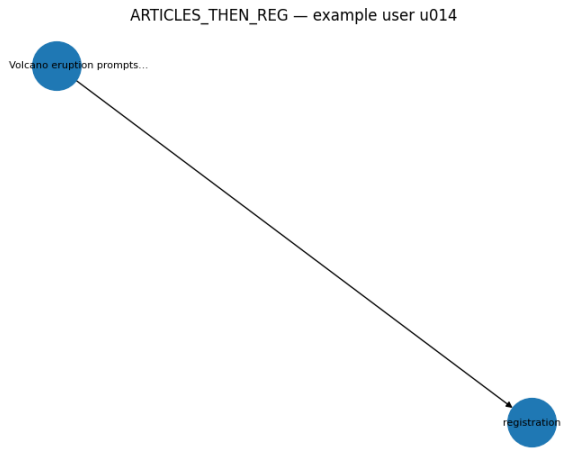
The Hitlog_<date>.csv contains the following columns

Column	Type	Description
Page_name	string	Page-Title
Page_url	string	URL path of page
User_id	string	User Id
Timestamp	datetime	Time user visited the article

Use case Coverage







Algorithm - URL Counter Based Approach

- Process each user's events in chronological order (sorted by timestamp)
- Maintain a set of unique articles seen in the current "journey"
- When a /register event is encountered:
  - Increment the count for each article in the journey set by 1
  - Clear the journey set for the next registration cycle
- Articles viewed after registration or without subsequent registration are not counted

Benefits

1. Easy to understand and implement
2. Efficient memory usage

Algorithm - Graph-Based Approach

- Build a directed graph where:
  - Nodes represent pages (articles, registration page, etc.)
  - Edges connect consecutive page views for each user
- Traverse through the graph for each user and when a /register event is encountered:
  - Increment the count for each article in the journey set by 1
  - Clear the journey set for the next registration cycle
- Store metadata (article name, weight) in Node objects

Benefits

1. Follows conventional data structure approach.
2. Can be upgraded for weighted page ranking solution.

Drawbacks

1. High Memory overhead for the current problem
2. More complex

Algorithms (Add-On)

Building on the Graph based solution, I've implemented a **Weighed Directed Graph** based approach to rank the articles.

- **Approach**

  - Builds on previous idea of Graph-traversal for each user.
  - In place of frozen weight (1) the algorithm implements dynamic weights where
    - Each article gains a higher weight when it directly precedes a registration event.
    - The influence of earlier articles **decays gradually** as the traversal moves away from the registration point.
    - If an article appears multiple times in a user's journey, its cumulative weight **increases**, reflecting repeated engagement.
  - The final ranking aggregates these weights across all users to identify the most influential articles.

Results

- The output is stored as Comma Separated Value File ".csv" with ranking to all the articles.

Column	Type	Description
Page_name	string	Title
Page_url	string	URL for the article
Total	integer	Number of unique users who registered after viewing

Pipeline

Apart from the conventional notebooks, the code follows a modular structure designed for future enhancements.

The overall pipeline uses the **uv** package manager for environment management and **ruff** as the main linting tool.

Several repetitive commands are simplified through the **Makefile**.

The structure clearly separates the notebook-based solution, which is easy to understand, from the modular codebase that follows industry standards.

The code is covered with test cases for validating the accuracy.

Usage

Code can be executed using CLI

python -m telegraph_ranker.cli \ --input <input file directory>\ --output <output file directory>\ --approach <approach – timestamp/graph>

Arguments:

--input: Path to input hitlog CSV file (required)

--output: Path for output CSV file (required)

--approach: Ranking algorithm to use: timestamp (default) or graph