DSI321 Project

Overview

This project monitors and analyzes public discussions about Thammasat University using real-time web scraping and NLP. It extracts posts, generates keyword-based word clouds, and automates the entire workflow with Prefect.

Key capabilities include:

- Real-time scraping of social media posts and news articles mentioning TU
- NLP processing to extract and display significant terms
- Visualization using word clouds
- CI/CD integrations to ensure data quality and maintain code security

Tools Used

Tool	Purpose
lakeFS	Acts as a data versioning system, ensuring reproducibility and control over all changes in the dataset
Docker	Containerizes the application and its dependencies, enabling seamless deployment across different environments
Prefect	Orchestration tool to automate and schedule the scraping and processing pipelines
Streamlit	Used to create an interactive web-based dashboard for visualizing word clouds and key metrics

Hardware Requirements

- Docker-compatible environment
- Local or cloud system with: - At least 4 GB RAM - Internet access for X data - Port availability for Prefect UI (default: `localhost:4200`)

Project Structure

```
.
├── config                          # Configuration files for
│   ├── docker
│   │   ├── Dockerfile.cli          # Dockerfile for CLI usage
│   │   └── Dockerfile.worker       # Dockerfile for worker se
│   ├── logging
│   │   └── modern_log.py           # Custom logging configurc
│   └── path_config.py              # Path configuration for f
├── src                             # Source code directory
│   ├── backend                     # Backend logic for scrapi
│   │   ├── load
│   │   │   └── lakefs_loader.py    # Module for loading data
│   │   ├── pipeline
│   │   │   ├── incremental_scrape_flow.py   # Scraping flow f
│   │   │   └── initial_scrape_flow.py       # Scraping flow f
│   │   ├── scraping
│   │   │   ├── x_login.py          # Script to log in to X
│   │   │   └── x_scraping.py       # Script to scrape data fr
│   │   └── validation
│   │   │   └── validate.py         # Data validation logic
│   └── fronend                     # Frontend components (Not
│       └── streamlit.py            # Streamlit app for data c
├── test                            # Unit and integration tes
├── .env.example                    # Example of environment v
├── .gitignore                      # Git ignore rules
├── README.md                       # Project documentation
├── docker-compose.yml              # Docker Compose configurc
├── pyproject.toml                  # Python project configurc
├── requirements.txt                # Python package requireme
└── start.sh                        # Startup script for the p
```

Schema

This project enforces a strict schema and data validation protocol to ensure data consistency and integrity. Below is the schema and the validation results from the processed dataset (`data.parquet`):

DataFrame Schema

(`df_verlify.dtypes`):

Column	Data Type
category	string[python]
tag	string[python]
username	string[python]

tweetText	string[python]
timestamp	datetime64[ns, UTC]
scrapeTime	datetime64[ns]
tweet_link	string[python]
index	int64
year	int32
month	int32
day	int32

Data Type Comparison

`(df.dtypes == df_verlify.dtypes)`

Column	Match
category	True
tag	True
username	True
tweetText	True
timestamp	True
scrapeTime	True
tweet_link	True
index	True
year	True
month	True
day	True

Record Count Check

`(len(df_verlify) > 1000)`

> ✅ **Result: True**
> The dataset contains more than 1,000 records

Duplicate Records Check

```
(df_verlify.duplicated().sum())
```

> ✅ **Result: 0**
> No duplicate records found in the dataset

Null Values Check

```
(df_verlify.isnull().sum())
```

Column	Null Count
category	0
tag	0
username	0
tweetText	0
timestamp	0
scrapeTime	0
tweet_link	0
index	0
year	0
month	0
day	0

> ✅ **Result:** No null values in any columns

Dataset Quality

Contains at least 1,000 records	Pass✅
Covers a full 24-hour time range	Pass✅
At least 90% data completeness	Pass✅
No columns with data type 'object'	Pass✅
No duplicate records	Pass✅

Benefits

Educational Benefits

- Hands-on experience in real-time data pipeline development
- Practice with Docker, Prefect, and Streamlit in production settings
- Application of CI/CD and data validation using GitHub Actions

Practical Benefits

- Reusable template for social media monitoring and keyword analysis
- Supports real-time, incremental scraping flows
- Easy to scale and deploy in both local and cloud environments

Organizational Benefits

- Validated data ensures insights are reliable and reproducible
- Automation reduces the need for manual monitoring
- Can be adapted to other sentiment or public opinion use cases

Prepare

1. Create a virtual environment

```
python -m venv .venv
```

2. Activate the virtual environment

- Windows

```
source .venv/Scripts/activate
```

- macOS & Linux

```
source .venv/bin/activate
```

3. Run the startup script

```
bash start.sh
# or
./start.sh
```

Running Prefect

1. Start the Prefect server

```
docker compose --profile server up -d
```

2. Connect to the CLI container

```
docker compose run cli
```

3. Run the initial scraping flow (to collect all tweets for base data)

```
python src/backend/pipeline/initial_scrape_flow.py
```

4. Schedule scraping every 15 minutes (incremental updates)

```
python src/backend/pipeline/incremental_scrape_flow.py
```

- **View the Prefect flow UI** Open your browser and go to: http://localhost:42000