Developer's Guide: *UX Research Analysis Tool*

Overview

The *UX Research Analysis Tool* is designed to help UX researchers analyze user discussions from various online sources (e.g., Reddit) to identify common issues, sentiment trends, and thematic patterns. While the provided examples and keywords focus on security concerns in the gaming community, researchers can easily adapt the tool to investigate any domain—by changing input data, keywords, or data sources.

Core Objectives:

- Collect discussions from online platforms (e.g., Reddit) using PRAW.
- Clean, structure, and analyze text data with pandas and NLTK's VADER for sentiment analysis.
- Visualize key insights with charts, word clouds, and other plots.
- Potentially generate PDF reports to share findings with stakeholders.

Licensing and Legal Notes

The project is licensed under the MIT License. This permissive license grants wide freedom to use, modify, and distribute the software, provided that the copyright notice and license terms are retained.

File Structure

A suggested, modular structure for scalability and clarity is as follows. Your actual repository structure may vary, and you can refactor `main.py` into dedicated modules as the project grows:

```
Research-Tool-for-UX-Researchers/
├─ main.py
├─ keys.py                  # Reddit API credentials (excluded by
.gitignore)
```

```
├─ requirements.txt
├─ README.md              # User-facing instructions, usage scenarios
├─ LICENSE
├─ .gitignore
├─ Project Specification.pdf
│
├─ src/
│  ├─ data_acquisition/
│  │  └─ scraper.py       # Future: scraping logic for Reddit or other
platforms
│  ├─ analysis/
│  │  ├─ cleaning.py      # Future: data cleaning/preprocessing
│  │  ├─ sentiment_analysis.py # Future: sentiment logic (VADER, etc.)
│  │  └─ topic_extraction.py  # Future: topic modeling, if desired
│  ├─ visualizations/
│  │  ├─ charts.py        # Future: bar charts, word clouds, etc.
│  │  └─ utils.py         # Future: helper functions for visualization
│  ├─ reporting/
│  │  └─ pdf_report.py    # Future: generate PDF reports
│  └─ ui/
│     └─ app.py           # Future: web UI (Flask/Streamlit)
│
└─ tests/
   └─ test_basic.py        # Future: unit tests
```

Note: Initially, most logic (scraping, sentiment, visualization) may reside in `main.py`. Over time, migrate code into `src/` directories for better maintainability and testability.

Installation and Setup

Prerequisites:

- Python 3.x

Dependencies:

The `requirements.txt` file lists all necessary libraries.

```
praw==7.6.0
nltk==3.8.1
pandas==2.0.3
plotly==5.15.0
wordcloud==1.8.2.2
matplotlib==3.7.1
seaborn==0.12.2
```

Install Dependencies:

```
pip install -r requirements.txt
```

Reddit API Credentials:

- Acquire a `client_id`, `client_secret`, and `user_agent` from Reddit by registering an app at https://www.reddit.com/prefs/apps.

Insert these credentials into `keys.py`:

```
# keys.py (excluded from version control)
client_id = "<your_client_id>"
client_secret = "<your_client_secret>"
user_agent = "<your_user_agent>"
```

Ensure `keys.py` is listed in `.gitignore` so you don’t commit sensitive information.

Running the Project

```
python main.py
```

`main.py` will:

- Connect to Reddit via PRAW, using credentials from `keys.py`.
- Scrape posts matching certain keywords (e.g., related to gaming security).
- Perform sentiment analysis, generate visualizations (histograms, time-series, word clouds, heatmaps).

If a web UI is implemented (e.g., Streamlit), navigate to `http://localhost:8501` to interact with filters, sliders, and other controls.

Data Flow and Internal Processes

1. **Data Acquisition**:
    - Scrape Reddit using PRAW or load CSV/Excel datasets uploaded by users.
    - Filter data by date, upvotes, or keywords as needed.
2. **Data Processing**:
    - Use pandas to clean and structure data.
    - Apply NLTK's VADER to assign sentiment scores (positive, negative, neutral).
    - (Future) Implement advanced NLP (topic modeling, entity extraction).
3. **Visualization**:
    - Present findings via bar charts (frequency of concerns), word clouds (common terms), time-series graphs (keyword trends), and heatmaps (sentiment by keyword).
    - Refactor visualization code into `src/visualizations/charts.py` for modularity.
4. **Reporting**:
    - Eventually, integrate `reporting/pdf_report.py` to export findings as PDF reports.
    - Provide templates, customizable sections, and embedding of charts.

Adapting the Tool to Any Topic

While the initial example focuses on security concerns in gaming communities, you can tailor the tool to any UX research question:

- **Change Keywords/Subreddits**:
  In `main.py` or `scraper.py`, replace security-related terms with other keywords relevant to your domain (e.g., "usability," "feature requests," "payment issues"). Switch `r/gaming` to any other subreddit or data source.
- **Alternate Data Inputs**:
  Instead of scraping, upload a CSV file with user comments from forums, surveys, or internal feedback channels. Ensure data matches expected schema (Post/Comment, User, Date, Topic) as per `README.md` guidelines.
- **Refining Analysis**:
  Enhance sentiment analysis by integrating more sophisticated NLP libraries, machine learning classifiers, or topic modeling algorithms.

Integration with the User-Facing README

The README.md provides instructions for end-users, detailing how to install dependencies, run the project, upload data, and interpret visualizations. As a developer:

- **Data Schema Enforcement**:
  Ensure code validates that uploaded CSV/Excel files match the schema (Post/Comment, User, Date, Topic). Add error handling to guide users if data is malformed.
- **Interactive Filters & Sliders**:
  The README describes using sliders/filters to adjust sentiment categories or topics. Implement these UI elements in `src/ui/app.py` (if using a web dashboard) and connect them to back-end filtering logic.
- **Exporting Insights**:
  The README mentions exporting visualizations and insights into PDF reports. As a developer, integrate the reporting code and ensure exported reports reflect the selected filters, timelines, or topics.

Testing and Maintenance

Unit Tests:
Add unit tests in `tests/` for key functionalities (scraping, sentiment analysis, data transformations). Use `pytest` to run tests:

bash
Copy code
```
pytest tests/
```

-
- **Continuous Integration**:
  Set up GitHub Actions or another CI tool to run tests on every pull request. This maintains code quality and prevents regressions.
- **Code Style**:
  Use `black` or `flake8` to maintain consistent formatting and style standards.
- **Documentation Updates**:
  Keep this Developer's Guide and the README in sync. Document any new features, dependencies, or structural changes promptly.

Future Enhancements

- **Modular Codebase**:
  Move logic out of `main.py` into `src/` directories for better organization and clarity.
- **Advanced Analytics**:
  Introduce topic modeling (LDA), entity extraction, or ML-based sentiment analysis for deeper insights.
- **Performance & Scalability**:
  Consider a database backend for large datasets, parallelize sentiment analysis, or integrate caching strategies.
- **UI/UX Improvements**:
  Build a fully interactive dashboard with filtering options, tooltips, and dynamic updates to charts as users adjust parameters.