Data Management Plan (DMP)  
Project: Scenario C — GitHub Repository Analytics  
Objective: Analyze open-source project trends and activity to compare repository health and activity for a chosen programming language or topic.

1. Scope & Purpose  
This project collects public repository metadata and basic activity statistics (stars, forks, issues, last push, commit activity, contributor counts) from the GitHub REST API for the top 20–50 repositories matching a selected language or topic. The goal is to produce a reproducible dataset and summary analyses that show trends in activity, contribution, and quality for coursework deliverables.

2. Data Sources & Types  
Primary source: GitHub REST API (https://api.github.com).  
Data types collected (examples):  
- Repository metadata: full\_name, html\_url, description, language, created\_at, pushed\_at.  
- Activity metrics: stargazers\_count, forks\_count, open\_issues\_count, watchers\_count.  
- Derived stats: last-year commits (from /stats/commit\_activity), contributors (from /stats/contributors), contributor counts.  
- Operational logs: collection timestamps, request counts, rate-limit snapshots, agent strategy changes.  
All data are public, non-sensitive repository metadata. No private or user credential data will be collected or stored.

3. Collection Method & Tools  
Agent: A Python class GitHubDataAgent (delivered) that:  
- Loads configuration from config.json or .env (preferred) for the GitHub token and collection parameters.  
- Discovers top repositories via /search/repositories then requests per-repo endpoints.  
- Implements respectful collection: exponential backoff, jittered delays, rate-limit monitoring and adaptive delay multiplier.  
- Retries /stats/\* endpoints which may return HTTP 202 (processing).  
- Logs events to data\_collection.log and periodically flushes results to disk.  
Authentication: Personal Access Token (PAT) provided locally (never stored in repo). The agent reads GITHUB\_TOKEN from .env or config.json (local file). Example config file provided with placeholder values.

4. File Formats & Naming Conventions  
Primary outputs (saved to results/):  
- github\_repo\_summary.json — full records (JSON array, UTF-8, pretty printed).  
- github\_repo\_summary.csv — flattened table for spreadsheet use (UTF-8, comma separated).  
- collection\_stats.json — request counts, successes, failures, quality scores.  
- dataset\_metadata.json — automated metadata (collection date, agent version, config summary).  
- quality\_report.json and quality\_report.md — machine + human readable QA report.  
- collection\_summary.txt — short human summary.  
- data\_collection.log — rolling collection log.  
Filenames include timestamps in the metadata; datasets use consistent names as above for reproducibility.

5. Metadata & Documentation  
The agent automatically generates:  
- dataset\_metadata.json containing collection date/time, agent version, collector identity (local username), total\_records, data\_sources, processing history and a generated data dictionary (variable names, sample types and short descriptions).  
- quality\_report.md (human-readable) describing completeness, distributional statistics, anomalies and recommendations.  
- A README.md (project root) describing required environment, how to run the agent, and how to reproduce the collection.  
These documents make the dataset reusable and traceable.

6. Quality Assurance & Validation  
QA steps implemented in the agent:  
- Validation rules: required fields (full\_name, html\_url, stargazers\_count) must be present for a record to be stored.  
- Quality metrics: completeness, timeliness (recent push within 1 year), consistency (language distribution), and accuracy placeholder (external cross-checks not implemented for assignment).  
- Anomaly detection: simple numeric outlier detection (mean ± 3 stdev) for numeric metrics.  
- Logging & checkpoints: agent flushes to disk every 10 records; logs contain delay and rate-limit events to show respectful behavior.  
Results include collection\_stats.json and quality\_report.json that summarize test outcomes and quality scores; include these in deliverable.

7. Storage, Backups & Preservation  
Working storage: local project folder (results/) during development.  
Backup strategy: final deliverables (JSON/CSV/metadata/report/log) should be copied to a persistent location (e.g., personal cloud storage or institutional drive) and included in the course submission ZIP. For longer term preservation, export CSV and JSON to a research data repository if required. Add .gitignore entries for config.json and .env to avoid committing secrets.  
Retention: keep raw outputs and metadata for the course term (recommended ≥ 1 year) or per institutional retention policy.

8. Access, Sharing & Licensing  
Data collected are public GitHub repository metadata; sharing is permitted under GitHub terms. The dataset will be shared in the course submission; sensitive tokens will never be included.  
Provide a short license (e.g., CC0 or CC-BY) in the project README for the assembled dataset. Cite GitHub as the original data source.

9. Security & Privacy  
API tokens: never hardcoded. Use .env or OS environment variables; .env and config.json are added to .gitignore.  
No PII: We only collect public repo metadata — no private emails, personal tokens or other PII. If any PII appears in a repo’s description, treat it as part of public data; do not attempt to extract or store additional personal data.  
Network: use HTTPS connections to GitHub API; follow GitHub rate limits and policies.

10. Roles & Responsibilities  
Collector / Developer: (Your Name) — set up token locally, run the agent, verify outputs, and submit datasets and logs.  
Reviewer / Instructor: evaluates deliverables (code, config, results, metadata, log screenshots) and verifies respectful collection.

11. Reproducibility & Deliverables  
Deliverables to submit:  
- Source: github\_agent.py, run\_agent.py, README.md, config.example.json, .env.example.  
- Data: results/github\_repo\_summary.json, results/github\_repo\_summary.csv, results/collection\_stats.json.  
- Documentation: results/dataset\_metadata.json, results/quality\_report.json, results/quality\_report.md, results/collection\_summary.txt.  
- Evidence: data\_collection.log and screenshots showing agent run and respectful delays.  
Include a short paragraph in submission describing how to reproduce (activate venv, set GITHUB\_TOKEN in .env, python run\_agent.py --config config.json).

12. Risk Management & Contingencies  
If rate limits are hit, the agent increases delay multiplier and waits for reset. If token invalid, the agent logs 401 and stops; regenerate token. If /stats endpoints return 202, the agent retries with backoff. Increase base\_delay and reduce topN when necessary.

Conclusion  
This DMP ensures the collection is respectful, reproducible, documented, and auditable. The agent’s automatic metadata, QA reports, and logging satisfy the deliverables and make the dataset ready for analysis and submission.