Popper Framework: Data Integrity Agent Development Report

- **Name**: Pushkar Kurhekar
- **Unique ID**: 106
- **Project:** Popper Framework
- **Project Leader:** Cletus Lopes
- **Date**: August 15, 2025 - September 7, 2025
- **GitHub Link**: https://github.com/Humanitariansai/Popper/tree/main/data_integrity_agent
- **Video Link**: data-integrity-agent-video.mp4

Week 1: Foundation and Core Development (August 15-21, 2025)

Learning and Research Phase

Understanding AI Agents
- Studied existing online guides and resources on AI agent architectures
- Researched different approaches to autonomous data processing
- Analyzed the distinction between basic automation and agent-like behavior

Codebase Exploration
- Reviewed the existing FactCheckingAgent implementation
- Studied the project structure and coding patterns
- Understanding the Popper Framework's philosophy and technical requirements
- Familiarized with the existing agent interfaces and design principles

Tasks Completed

Basic Agent Implementation (August 21)
- Commit: 05a9c4f - "Basic data integrity code implementation"
- Core Functionality Delivered:
- Missing value detection with percentage calculations
- Outlier detection using IQR (Interquartile Range) method
- Statistical distribution analysis (mean, std, min, max, skewness)

- Data summary generation (row/column counts, data type analysis)

LLM Integration (August 22)

- Commit: 11de32e - "Add LLM enhanced agent"
- Files Added:
- llm_enhanced_agent.py (314 lines)
- llm_example.py (114 lines)

Features Implemented:

- Google Gemini-1.5-flash integration for validation strategy selection
- LLM-driven decision making for validation approach based on data characteristics
- Strategy interpretation and insight generation
- JSON parsing with markdown handling for LLM communication

Other Changes:

- Dependency management improvements
- JSON serialization fixes for numpy/pandas compatibility

Technical Challenges Overcome

- JSON Serialization Issues: Implemented _convert_to_serializable() method to handle numpy int64/float64 types that aren't natively JSON serializable
- LLM Response Parsing: Created robust _extract_json_from_response() to handle markdown-wrapped JSON responses from Gemini
- Fallback Strategy: Built graceful degradation from LLM to basic validation when API calls fail

Weekly Impact

- Core Architecture: Established the foundation for modular, extensible validation system
- LLM Integration: Integrated AI-driven validation strategy selection

Week 2: Advanced Features and Modularity (August 22 - September 1, 2025)

Tasks Completed

Visualization System (August 26)
- Commit: 6a44d87 - "Add visualization module with Plotly charts and HTML reports"
- File Created: visualization.py (505 lines)

Visualization Features:
- Interactive Plotly charts for missing values, outliers, distributions
- Comprehensive HTML reports with gauge charts for data quality scores
- Correlation matrices and statistical summaries
- Professional-grade data integrity dashboards

Advanced Type Detection (August 27)
- Commit: 3e50b86 - "Add data type detection and schema validation"
- File Created: type_detector.py (472 lines)

Type Detection Capabilities:
- Automatic inference of 13 data types (integer, float, email, phone, URL, currency, etc.)
- Confidence scoring for type predictions
- Pattern recognition for structured data formats
- Schema validation against expected types

Configuration Management (August 25)
Commits:
- 0c83366 - "Add configuration system with YAML support"
- 6959723 - "Add CLI interface for Data Integrity Agent"
- 8c3ba24 - "Add version tracking for Data Integrity Agent"

Files Created:
- config.py (238 lines)
- config.yaml (67 lines)

- cli.py (400 lines)
- version.py (19 lines)

Features Delivered:
- YAML-based configuration with validation
- CLI interface with colored output and progress bars
- Support for multiple file formats (CSV, Excel, JSON, Parquet, HDF5)
- Batch processing capabilities
- Version tracking and management

Added Testing (August 28)
- Commit: 656bb87 - "Add test code for new features"
- File Created: test_new_features.py (248 lines)

New Features

- Modular Architecture: Each component (validation, visualization, configuration) became independently usable
- Format Agnostic: Support for 6 different data file formats
- Progressive Enhancement: System works with or without LLM, with or without visualization
- Enterprise Patterns: Configuration management, version tracking, comprehensive logging

Weekly Impact

- Lines of Code: ~1,400 additional lines
- System Development: Enhanced from prototype to more complete architecture
- User Experience: CLI interface with colored output and progress tracking

Week 3: Automation and Intelligence (September 2-7, 2025)

Tasks Completed

Automated Data Repair (September 2-3)

Commits:

- 10a8159 - "Add skeleton for DataRepairAgent action-taking capabilities"
- 7d67826 - "Basic implementation for data repair agent"
- File Enhanced: action_agent.py (evolved from 22 to 46 lines)

Autonomous Capabilities:

- Automatic missing value imputation (mean for numeric, mode for categorical)
- Outlier handling using IQR-based capping
- Repair report generation with timestamps and action tracking
- Integration with validation pipeline for before/after analysis

System Integration (September 3-5)

Commits:

- af5c580 - "Remove example code, implementing new demo script"
- 53f35bb - "Remove domain param and fix json parse issue"
- 299be95 - "Initial runner code implementation"
- File Created: demo.py (160 lines)

Integration Features:

- End-to-end demonstration of all system capabilities
- Before/after data repair comparison
- Comprehensive workflow orchestration
- Production-ready demo with error handling

Video Recording (September 5-6)

- Recorded a 9 minute video going over the implementation so far and discussing future steps.

Documentation & Handover (September 6-7)

- README expanded with full implementation details and explanations.
- Conducted a meeting with project leader and team member to explain progress so far and how it can be taken forward.
- Performed final code clean up and documentation changes wherever required.

Documentation Updates:

- Architecture documentation
- Usage examples for all components
- Troubleshooting guide for common issues
- Future development scope

Weekly Impact

- Lines of Code: ~400 additional lines
- Major documentation updates
- Recorded video for knowledge transfer and smooth handover process

---

Technical Architecture Summary

Core Components Developed

BasicDataIntegrityAgent (data_integrity_agent.py)

- Foundation validation logic
- Missing values, outliers, distributions analysis
- JSON serialization handling

LLMEnhancedDataIntegrityAgent (llm_enhanced_agent.py)

- Google Gemini integration
- Intelligent validation strategy selection
- Advanced format and range validation
- Insight generation and quality scoring

DataRepairAgent (action_agent.py)

- Automated data repair capabilities
- Missing value imputation
- Outlier handling
- Repair reporting

DataIntegrityVisualizer (visualization.py)

- Interactive Plotly visualizations
- HTML report generation
- Professional dashboard creation

DataTypeDetector (type_detector.py)

- Advanced type inference
- Pattern recognition
- Schema validation

Configuration System (config.py, config.yaml)

- YAML-based settings management
- Validation parameter tuning
- Enterprise configuration patterns

CLI Interface (cli.py)

- Professional command-line interface
- Batch processing
- Multiple output formats
- Progress tracking

Validation Capabilities

- Missing Values: Detection, analysis, and automated repair
- Outliers: IQR-based detection and intelligent handling
- Data Types: Automatic inference with confidence scoring
- Formats: Email, phone, URL, date validation
- Ranges: Age, score, and custom range validation
- Consistency: Logical contradiction detection
- Schema: Expected vs. actual type validation

Future Directions

Lessons for AI Development

Human-AI Collaboration: Most effective when AI provides insights and humans retain control

Fallback Strategies: Essential for reliable systems in uncertain environments

Incremental Complexity: Build simple first, add intelligence gradually

Documentation-Driven Development: Understanding emerges through explanation

Personal Growth

- Systems Thinking: Learned to design modular, extensible architectures
- AI Integration: Gained practical experience with LLM APIs and prompt engineering
- Product Development: Experienced full lifecycle from prototype to production-ready system
- Open Source: Contributed to a meaningful educational project with philosophical foundations

Technical Skills Developed & Refined

- Python development with pandas, numpy
- Google Generative AI API integration
- CLI development with argparse and colorama
- YAML configuration management
- Interactive visualization with Plotly
- JSON serialization and error handling
- Git workflow and version management

Conclusion

Though this was a relatively short 2.5-week experience, it provided an excellent learning opportunity as I had not worked on developing AI agents before. The project allowed me to explore the intersection of traditional data validation techniques with modern LLM capabilities while contributing to the Popper Framework's mission of computational skepticism.

I gained valuable hands-on experience with AI integration patterns, modular system design, and the practical challenges of building educational tools that maintain both technical rigor and transparency. Working within the framework's "building to learn" philosophy was particularly valuable, as it encouraged thoughtful design decisions over quick solutions.

I want to thank Humanitarians AI for providing this opportunity to contribute to meaningful open-source work while developing new technical skills.