4DT903 Project Proposal

Automated Data Analysis Pipeline Generator from Dataset Metadata

Samuel Berg (sb224sc)

October 2025

Contents

1 Objective

1.1 Domain and Problem

This project addresses the challenge of automatically generating executable Jupyter notebook data analysis pipelines from dataset metadata and analysis requirements. [1]

Data scientists and analysts often perform repetitive exploratory data analysis (EDA) tasks when working with new datasets: loading data, checking data quality, generating descriptive statistics, creating visualizations, and applying standard preprocessing steps. This manual process involves:

- Writing boilerplate code for data loading, cleaning, and initial exploration.
- Repeatedly implementing similar visualization patterns (distributions, correlations, time series).
- Manually selecting appropriate statistical tests and machine learning models based on data characteristics.
- Creating inconsistent analysis workflows across different projects.
- Difficulty reproducing analyses when datasets are updated or extended.
- Lack of standardized documentation and reporting formats.

The project will use Model-Driven Engineering (MDE) to automatically transform dataset metadata and analysis specifications into complete, executable Jupyter notebooks with data loading, quality checks, exploratory analysis, visualizations, statistical tests, and preliminary modeling, ensuring reproducibility and best practices.

2 Models

2.1 Metamodels and Domains

The project involves three distinct domains, each requiring its own metamodel:

1. Dataset Metadata Metamodel (Source Domain)

   This metamodel captures the structure and characteristics of datasets:

   - Dataset: Root element with name, source, format (CSV, Excel, JSON, SQL, Parquet).
   - Column: Individual data field:
     - Name: Column identifier.
     - DataType: Categorical, numerical (continuous/discrete), temporal, text, boolean.
     - Constraints: Unique, not-null, primary key, foreign key.
     - Statistics: Min, max, mean, median, mode, missing percentage.
     - Distribution: Normal, skewed, uniform, bimodal.
   - Relationship: Connections between columns:
     - ForeignKey: References to other tables/datasets.
     - Correlation: Statistical relationships between numerical columns.
     - Hierarchy: Parent-child relationships (e.g. Country $\rightarrow$ State $\rightarrow$ City).
   - TemporalInfo: Time related properties:
     - TimeColumn: Timestamp or date field.
     - Granularity: Second, minute, hour, day, month, year.
     - Seasonality: Detected patterns.
   - DataQuality: Issues and anomalies:
     - MissingValues: Percentage and pattern (MCAR, MAR, MNAR).
     - Outliers: Statistical outliers with detection method.
     - Duplicates: Duplicate row information.
     - Inconsistencies: Format issues, value ranges.

2. Analysis Specification Metamodel (Source Domain – Secondary)

   This metamodel defines what analysis to perform:

   - AnalysisGoal: Overall objective (exploration, prediction, clustering, comparison).
   - Question: Specific research questions to answer.
   - TargetVariable: Column of primary interest (for supervised learning).
   - FeatureSet: Columns to include in analysis.

- AnalysisTask: Specific analyses to perform:
  - DescriptiveAnalysis: Summary statistics, distributions.
  - VisualAnalysis: Plots and charts to generate.
  - StatisticalTest: Hypothesis tests (t-test, chi-square, ANOVA, correlation).
  - ModelingTask: ML algorithms to apply (regression, classification, clustering).
- Constraint: Requirements and limitations:
  - ComputationalBudget: Time/memory constraints.
  - InterpretabilityRequirement: Need for explainable models.
  - Preprocessing: Required data transformations.

3. Notebook Structure Metamodel (Intermediate Domain)

   This platform-independent metamodel represents Jupyter notebook organization:

   - Notebook: Container with title, author, description.
   - Section: Logical grouping with heading and markdown description:
     - ImportSection: Library imports and configuration.
     - DataLoadingSection: Data ingestion code.
     - DataQualitySection: Quality checks and profiling.
     - ExploratorySection: EDA with statistics and visualizations.
     - PreprocessingSection: Data cleaning and transformation.
     - AnalysisSection: Statistical tests or modeling.
     - ResultsSection: Findings and interpretation.
     - ConclusionSection: Summary and recommendations.
   - Cell: Individual notebook cell:
     - CodeCell: Executable Python code.
     - MarkdownCell: Documentation and explanations.
     - OutputCell: Expected output (figure, table, text).
   - CodeBlock: Python code structures:
     - ImportStatement: Library imports.
     - Function: Reusable code blocks.
     - DataOperation: Pandas/NumPy operations.
     - Visualization: Matplotlib/Seaborn/Plotly code.
     - Model: Scikit-learn/TensorFlow model definitions.
   - Documentation: Markdown content:
     - Explanation: Interpretive text.
     - Warning: Data quality notes.
     - Insight: Key findings.

4. Python/Jupyter Implementation Metamodel (Target Domain)

   This metamodel represents executable Jupyter notebook elements:

   - NotebookFile: .ipynb JSON structure.
   - CellMetadata: Execution order, cell type, tags.
   - PythonCode: Executable Python with proper syntax.
   - Library: Specific library usage:
     - Pandas: DataFrame operations (`read_csv`, `groupby`, `merge`, `pivot`).
     - NumPy: Array operations and mathematical functions.
     - Matplotlib/Seaborn/Plotly: Visualization code.
     - Scikit-learn: Preprocessing, models, metrics.
     - Scipy: Statistical test.
     - Statsmodels: Advanced statistical modeling.
   - PlotConfiguration: Figure size, style, colors, labels, legends.
   - DataFrameOperation: Specific pandas operations with parameters.
   - MarkdownFormatting: Headers, lists, code blocks, LaTeX equations, tables.

2.2 Metamodel Relations

The metamodels are connected through the transformation pipeline:

- Dataset Metadata + Analysis Specification → Notebook Structure: M2M transformation that infers appropriate analysis workflow based on data characteristics and goals.
- Notebook Structure → Python/Jupyter Implementation: M2M transformation adapting abstract analysis steps to specific library calls and notebook structure.
- Python/Jupyter Implementation → .ipynb File: M2T transformation generating executable Jupyter notebook.

2.3 Tool Integration

Model Creation:

Dataset Metadata Generation:

- Automatically extract metadata from existing datasets using pandas-profiling or custom profiler.
- Import schema definitions from database metadata (SQL DDL, MongoDB schemas).

- Parse data dictionaries (Excel/CSV files with column descriptions).
- Support Great Expectations validation suites as metadata source.
- Accept manual metadata definition through EMF-based form editor.
- Store metadata in XMI format or JSON schema.

Analysis Specification:

- Create through custom Eclipse editor with wizard-style interface.
- Import from template library for common analysis patterns (customer segmentation, time series forecasting, A/B testing).
- Convert from natural language descriptions using simple NLP parsing.
- Integration with MLflow or DVC for experiment tracking metadata.

Model Consumption:

- Generated Jupyter notebooks (.ipynb) are consumed by:
  - JupyterLab / Jupyter Notebook environments.
  - Google Colab (with minor adaptations).
  - VS Code with Jupyter extension.
  - Databricks notebooks (with Spark adaptations).
- Notebooks are immediately executable with proper environment setup.
- Can be version controlled in Git repositories.
- Convertible to Python scripts (.py) for production pipelines.

Model Updates:

- When dataset changes (new columns, data refreshes), metadata can be regenerated.
- Incremental regeneration updates only affected analysis sections.
- Custom code cells marked with special tags are preserved during regeneration.
- Diff tool shows changes between generated versions.
- Support for parameterized notebooks using papermill for automatic re-execution with different parameters.

3 Transformations

3.1 Transformation Pipeline

1. M2M Transformation: Dataset Metadata + Analysis Specification to Notebook Structure (QVTo)

   This transformation implements intelligent analysis workflow generation:

   Structure Generation:

   - Create notebook sections based on analysis goals:
     - Always include: Imports, Data Loading, Data Quality Check.
     - Add preprocessing section if data quality issues detected.
     - Include EDA section with visualizations appropriate for data types.
     - Add statistical testing section if comparison/hypothesis testing specified.
     - Include modeling section if prediction/classification goal specified.
     - Generate results interpretation and visualization.

   Data Type-Driven Analysis:

   - Numerical columns:
     - Generate histogram, box plot, QQ plot for distribution analysis.
     - Calculate descriptive statistics (mean, median, std, quartiles).
     - Check for outliers using IQR or z-score methods.
     - Create correlation heatmaps if multiple numerical columns exist.
   - Categorical columns:
     - Generate bar charts showing value distributions.
     - Calculate frequency tables and percentages.
     - Create cross-tabulations with target variable.
     - Suggest chi-square tests for independence.
   - Temporal columns:
     - Generate time series plots.
     - Calculate trend and seasonality components.
     - Create autocorrelation plots.
     - Suggest appropriate time series models (ARIMA, Prophet).
   - Text columns:
     - Generate word clouds and frequency distributions.
     - Calculate basic text statistics (length, unique values).
     - Suggest sentiment analysis or topic modeling if appropriate.

Relationship-Based Analysis:

- For correlated columns, generate scatter plots and correlation coefficients.
- For foreign key relationships, suggest merge operations and relationship visualizations.
- For hierarchical data, create aggregation analyses at different levels.

Quality-Driven Preprocessing:

- If missing values $> 5\%$, generate multiple imputation strategies to compare.
- If outliers detected, create both outlier-inclusive and outlier-excluded analyses.
- If duplicates found, generate deduplication code with explanation.
- If class imbalance detected (categorical target), suggest resampling techniques.

Goal-Driven Modeling:

- Prediction goal with numerical target:
  - Split data into train/test sets.
  - Generate linear regression, random forest, gradient boosting models.
  - Create feature importance visualizations.
  - Calculate regression metrics (RMSE, MAE, R^2).
- Classification goal:
  - Generate logistic regression, decision tree, random forest models.
  - Create confusion matrices and ROC curves.
  - Calculate classification metrics (accuracy, precision, recall, F1).
  - Include cross-validation code.
- Clustering goal:
  - Generate K-means, hierarchical clustering, DBSCAN.
  - Create elbow plots for optimal cluster selection.
  - Visualize clusters with PCA/t-SNE dimensionality reduction.
  - Calculate silhouette scores.

Documentation Generation:

- Create markdown cells explaining each analysis step.
- Add interpretive comments for statistical results.
- Generate data quality warnings and recommendations.

- Include best practice notes (e.g. "Consider feature engineering", "Check for multicollinearity").

2. M2M Transformation: Notebook Structure to Python/Jupyter Implementation (QVTo)

   This transformation adapts abstract analysis to concrete Python code:

   Library Selection and Optimization:

   - Choose visualization library based on complexity:
     - Matplotlib for simple plots.
     - Seaborn for statistical visualizations.
     - Plotly for interactive dashboards.
   - Select appropriate pandas operations for efficiency (vectorized operations over loops).
   - Choose between scikit-learn and statsmodels based on analysis needs.

   Code Pattern Implementation:

   - Data Loading:

     ```
     # Generate appropriate read function based on format
     # CSV → pd.read_csv() with dtype inference
     # Excel → pd.read_excel() with sheet handling
     # SQL → pd.read_sql() with connection management
     # JSON → pd.read_json() with normalization
     ```

   - Visualization:
     - Apply consistent styling (figure size, color schemes, fonts).
     - Add proper titles, axis labels, legends.
     - Include grid lines and annotations where helpful.
     - Make plots publication-ready.
   - Statistical Tests:
     - Select appropriate test based on data characteristics.
     - Check assumptions before applying tests.
     - Include effect size calculations, not just p-values.
     - Generate clear interpretation of results.
   - Machine Learning Pipeline:
     - Create proper train/test splits with stratification if needed.
     - Include feature scaling where appropriate.
     - Implement cross-validation.
     - Add hyperparameter tuning (GridSearchCV) for important models.

– Include model persistence code (pickle/joblib).

Error Handling:

- Add try-except blocks for data loading.
- Include data validation checks.
- Generate informative error messages.
- Add assertions for data shape and type expectations.

Performance Optimization:

- Use vectorized operations instead of loops.
- Suggest chunking for large datasets.
- Include memory profiling hints.
- Add progress bars for long-running operations (tqdm).

3. M2T Transformation: Python/Jupyter Implementation to .ipynb File (Acceleo)

   This transformation generates executable Jupyter notebooks:

   Notebook Structure Generation:

```
{
  "cells": [
    {
      "cell_type": "markdown",
      "metadata": {},
      "source": ["# Analysis Title\n",
        "## Overview\n", "Description..."]
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {},
      "source": ["import pandas as pd\n",
        "import numpy as np\n", ...],
      "outputs": []
    },
    ...
  ],
  "metadata": {
    "kernelspec": {...},
    "language_info": {...}
  },
  "nbformat": 4,
  "nbformat_minor": 4
}
```

Code Quality:

- Generate PEP 8 compliant Python code.
- Include type hints where beneficial.
- Add docstrings for custom functions.
- Use meaningful variable names (df, not d; fig, ax for matplotlib).
- Apply consistent formatting (indentation, spacing).

Documentation Quality:

- Generate markdown with proper heading hierarchy.
- Include LaTeX equations for statistical formulas: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.
- Create formatted tables for results.
- Add inline code formatting for variable names.
- Include hyperlinks to documentation.

Reproducibility Features:

- Set random seeds for reproducible results.
- Include `requirements.txt` or `environment.yml`.
- Add data source information and download instructions.
- Include dataset versioning information.
- Generate timestamp in notebook header.

Interactive Elements:

- Add ipywidgets for parameter exploration where beneficial.
- Include table of contents for navigation.
- Add collapsible sections for lengthy outputs.
- Generate summary visualizations at the top for quick insights.

Output Management:

- Pre-populate expected outputs (optional, for demonstration).
- Add cell execution timing metadata.
- Include memory usage profiling for large datasets.
- Generate figure exports (PNG/SVG) for presentations.

3.2 Combining Transformations

The transformations will be orchestrated through a comprehensive workflow system:

Main Workflow:

1. Metadata Extraction Phase:
    - Load dataset and automatically profile it.
    - Extract statistical properties, data types, relationships.
    - Detect data quality issues.
    - Save metadata model.
2. Specification Phase:
    - User defines analysis goals through GUI wizard.
    - Select analysis type (EDA, prediction, comparison, clustering).
    - Specify target variables and features.
    - Set constraints (interpretability, computational budget).
    - Choose visualization preferences.
3. Notebook Generation Phase:
    - Execute Metadata + Specification → Notebook Structure transformation.
    - Preview notebook outline with section descriptions.
    - Allow user to enable/disable sections.
    - Execute Notebook Structure → Python Implementation transformation.
    - Execute Python Implementation → .ipynb file transformation.
4. Output Phase:
    - Generate executable .ipynb file.
    - Create requirements.txt with all dependencies.
    - Generate README with setup instructions.
    - Optionally execute notebook and save with outputs.
    - Package as downloadable ZIP or push to Git.

Advanced Features:

Template Library:

- Pre-built analysis templates for common scenarios:
    - Customer churn analysis.
    - A/B test statistical analysis.
    - Time series forecasting.

  - Image classification with CNN.
  - NLP sentiment analysis.
  - Market basket analysis
- Users can create custom templates and save them.

Multi-Dataset Analysis:

- Generate notebooks that analyze multiple related datasets.
- Create comparison analyses across datasets.
- Handle dataset merging and relationship visualization.

Parameterized Execution:

- Generate notebooks with parameters using papermill.
- Allow batch execution with different parameter sets.
- Create automated reporting pipelines.

Progressive Disclosure:

- Generate basic notebook first, then enhance with advanced features.
- Beginner mode: simpler code, more explanations.
- Advanced mode: optimized code, less commentary.
- Expert mode: compact code, minimal documentation.

Integration Capabilities:

- MLflow integration: Generate experiment tracking code.
- Great Expectations integration: Add data validation checkpoints.
- Airflow/Prefect integration: Generate workflow orchestration code.
- Dashboard generation: Create Streamlit/Dash apps from notebooks.
- Report generation: Convert to HTML/PDF with executive summaries.

Incremental Updates:

- When dataset is updated with new records:
  - Regenerate only data loading and profiling sections.
  - Preserve custom analysis code.
  - Re-execute statistical tests with new data.
  - Update visualizations automatically.

- When new columns are added:
  - Add analysis for new columns.
  - Update correlation analyses.
  - Suggest new features for existing models.

Quality Assurance:

- Validate generated code syntax before output.
- Include unit tests for custom functions.
- Add data validation assertions.
- Generate code quality report (complexity, documentation coverage).
- Include performance profiling suggestions.

Collaboration Features:

- Generate notebooks with clear section ownership.
- Include review checklists in markdown.
- Add version control-friendly cell IDs.
- Generate diff-friendly outputs (exclude execution counts and output in Git).

This MDE approach dramatically reduces the time required to start analyzing new datasets, ensures analysis best practices are followed consistently, improves reproducibility across projects, and allows data scientists to focus on domain-specific insights rather than boilerplate code. The generated notebooks serve as both executable analysis and comprehensive documentation, making knowledge transfer and collaboration significantly easier (see Figure 1).

References

[1] ”placeholder”, “”placeholder”.” ”placeholder”, ”placeholder”.

A Model and Transformation Overview

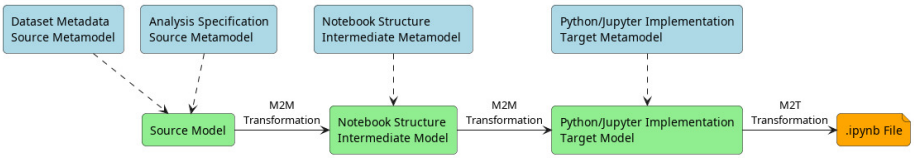

Figure 1: Overview of the Transformations to be implemented in this project