# 140509\_40.md - AI-Powered Data Quality and Cleaning Platform

## README

**Summary:** Build an intelligent data quality platform that automatically detects, diagnoses, and corrects data quality issues using machine learning techniques.

**Problem Statement:** Data quality issues significantly impact AI model performance and business decisions. Your task is to create an AI-powered platform that automatically identifies data quality problems (missing values, outliers, inconsistencies, duplicates), diagnoses root causes, and provides intelligent correction suggestions. The system should learn from data patterns, adapt to domain-specific requirements, and provide transparent quality improvement processes.

**Steps:** - Design automated data profiling and quality assessment algorithms - Implement ML-based anomaly detection and outlier identification - Create intelligent data imputation and correction mechanisms - Build data lineage tracking and quality issue root cause analysis - Develop domain-specific data validation rules and quality metrics - Include data quality monitoring and alerting for production pipelines

**Suggested Data Requirements:** - Datasets with known quality issues and correction examples - Domain-specific data quality rules and validation criteria - Historical data quality improvement cases and outcomes - Data lineage information and processing pipeline metadata

**Themes:** AI for Data & Data for AI, Using AI for Data cleaning

## PRD (Product Requirements Document)

### Product Vision

Create an AI-powered data quality platform that automatically detects, diagnoses, and resolves data quality issues at scale, ensuring high-quality data for analytics and machine learning applications.

### Target Users

* **Primary:** Data Engineers, Data Scientists, Analytics Teams
* **Secondary:** Data Stewards, Business Analysts, ML Engineers
* **Tertiary:** Data Governance Teams, Quality Assurance, Compliance Officers

### Core Value Propositions

1. **Automated Quality Detection:** ML-powered identification of quality issues across all data types
2. **Intelligent Remediation:** Smart correction suggestions with confidence scoring
3. **Root Cause Analysis:** Deep diagnostics to identify systemic data quality problems
4. **Continuous Monitoring:** Real-time quality monitoring for production data pipelines
5. **Domain Adaptation:** Customizable rules and quality metrics for specific domains

### Key Features

1. **Comprehensive Quality Assessment:** Multi-dimensional quality profiling and scoring
2. **ML-Based Anomaly Detection:** Advanced outlier and inconsistency detection
3. **Intelligent Data Imputation:** Context-aware missing value imputation
4. **Duplicate Detection and Resolution:** Fuzzy matching and record linkage
5. **Data Lineage and Root Cause Analysis:** Complete traceability and issue diagnosis
6. **Production Pipeline Integration:** Seamless integration with data processing workflows

### Success Metrics

* Quality detection accuracy: >95% precision in identifying data quality issues
* Remediation effectiveness: >90% improvement in data quality scores after correction
* Processing speed: Handle 1TB datasets within 2 hours
* False positive rate: <5% for quality issue alerts
* User adoption: 80% of data teams using platform within 12 months

## FRD (Functional Requirements Document)

### Core Functional Requirements

#### F1: Comprehensive Data Quality Assessment

* **F1.1:** Multi-dimensional quality profiling (completeness, validity, consistency, accuracy)
* **F1.2:** Statistical anomaly detection using unsupervised learning
* **F1.3:** Pattern-based validation rule discovery and application
* **F1.4:** Cross-column and cross-table consistency checking
* **F1.5:** Temporal quality analysis for time-series data

#### F2: Intelligent Anomaly Detection

* **F2.1:** Isolation Forest and Local Outlier Factor for numerical anomalies
* **F2.2:** Text anomaly detection using NLP and embedding techniques
* **F2.3:** Categorical outlier detection using frequency and entropy analysis
* **F2.4:** Multivariate anomaly detection considering variable relationships
* **F2.5:** Contextual anomaly detection based on business rules and domain knowledge

#### F3: Advanced Data Imputation and Correction

* **F3.1:** Multiple imputation techniques (mean, median, mode, KNN, MICE)
* **F3.2:** ML-based imputation using Random Forest, XGBoost, and neural networks
* **F3.3:** Time-series aware imputation with seasonality and trend consideration
* **F3.4:** Contextual imputation using related columns and external data sources
* **F3.5:** Confidence scoring and uncertainty quantification for corrections

#### F4: Duplicate Detection and Entity Resolution

* **F4.1:** Fuzzy string matching using phonetic and edit distance algorithms
* **F4.2:** ML-based record linkage with similarity learning
* **F4.3:** Blocking and indexing techniques for scalable duplicate detection
* **F4.4:** Active learning for improving matching accuracy over time
* **F4.5:** Hierarchical clustering for entity resolution and deduplication

#### F5: Data Lineage and Root Cause Analysis

* **F5.1:** End-to-end data lineage tracking from source to consumption
* **F5.2:** Impact analysis for quality issues across downstream systems
* **F5.3:** Root cause identification using causal inference techniques
* **F5.4:** Quality issue propagation analysis and containment strategies
* **F5.5:** Historical quality trend analysis and predictive quality modeling

#### F6: Production Integration and Monitoring

* **F6.1:** Real-time data quality monitoring with configurable thresholds
* **F6.2:** Integration with popular data processing frameworks (Spark, Airflow, Kafka)
* **F6.3:** Automated quality gates and pipeline validation
* **F6.4:** Quality SLA monitoring and breach alerting
* **F6.5:** Quality dashboard and reporting for stakeholders

## NFRD (Non-Functional Requirements Document)

### Performance Requirements

* **NFR-P1:** Quality assessment speed: Process 1M records per minute
* **NFR-P2:** Real-time monitoring latency: <30 seconds for quality alerts
* **NFR-P3:** Imputation processing time: <1 hour for datasets up to 10M records
* **NFR-P4:** Duplicate detection performance: Handle 100M record comparisons in <4 hours
* **NFR-P5:** Dashboard response time: <3 seconds for quality metric queries

### Accuracy Requirements

* **NFR-A1:** Anomaly detection precision: >95% with <5% false positive rate
* **NFR-A2:** Imputation accuracy: >90% for numerical values, >85% for categorical
* **NFR-A3:** Duplicate detection recall: >95% for true duplicates
* **NFR-A4:** Quality score consistency: ±2% variance across repeated assessments
* **NFR-A5:** Root cause identification accuracy: >80% for traceable quality issues

### Scalability Requirements

* **NFR-S1:** Handle datasets up to 1TB in size with distributed processing
* **NFR-S2:** Support 1000+ concurrent quality assessment jobs
* **NFR-S3:** Scale to monitor 10,000+ production data pipelines
* **NFR-S4:** Multi-tenant architecture supporting 500+ organizations
* **NFR-S5:** Horizontal scaling across cloud and on-premise infrastructure

### Integration Requirements

* **NFR-I1:** API-first architecture with comprehensive REST and GraphQL APIs
* **NFR-I2:** Native integration with major data platforms (Snowflake, Databricks, BigQuery)
* **NFR-I3:** Support for 20+ data formats and protocols
* **NFR-I4:** Real-time streaming integration with Kafka, Kinesis, Pub/Sub
* **NFR-I5:** MLOps integration with MLflow, Kubeflow, SageMaker

## AD (Architecture Diagram)

graph TB  
 subgraph "User Interfaces"  
 WEB\_UI[Web Dashboard]  
 API[REST/GraphQL APIs]  
 CLI[CLI Tools]  
 NOTEBOOKS[Jupyter Integration]  
 end  
   
 subgraph "API Gateway & Security"  
 GATEWAY[API Gateway]  
 AUTH[Authentication]  
 RATE\_LIMIT[Rate Limiter]  
 AUDIT[Audit Logger]  
 end  
   
 subgraph "Core Quality Services"  
 PROFILER[Data Profiler]  
 QUALITY\_ASSESSOR[Quality Assessor]  
 ANOMALY\_DETECTOR[Anomaly Detector]  
 IMPUTATION\_ENGINE[Imputation Engine]  
 DEDUP\_ENGINE[Deduplication Engine]  
 LINEAGE\_TRACKER[Lineage Tracker]  
 end  
   
 subgraph "ML & Analytics Engines"  
 OUTLIER\_DETECTION[Outlier Detection ML]  
 PATTERN\_LEARNING[Pattern Learning]  
 IMPUTATION\_ML[ML Imputation Models]  
 SIMILARITY\_ENGINE[Similarity Engine]  
 CAUSAL\_INFERENCE[Causal Analysis]  
 PREDICTIVE\_QUALITY[Quality Prediction]  
 end  
   
 subgraph "Processing Infrastructure"  
 SPARK\_CLUSTER[Apache Spark Cluster]  
 TASK\_SCHEDULER[Task Scheduler]  
 DISTRIBUTED\_COMPUTE[Distributed Computing]  
 GPU\_ACCELERATOR[GPU Acceleration]  
 STREAMING\_PROCESSOR[Stream Processor]  
 end  
   
 subgraph "Data Storage"  
 METADATA\_DB[PostgreSQL - Metadata]  
 QUALITY\_METRICS[InfluxDB - Quality Metrics]  
 LINEAGE\_GRAPH[Neo4j - Data Lineage]  
 RULES\_STORE[MongoDB - Quality Rules]  
 CACHE\_LAYER[Redis - Caching]  
 OBJECT\_STORAGE[S3 - Data & Models]  
 end  
   
 subgraph "External Integrations"  
 DATA\_SOURCES[Data Sources]  
 DATA\_PLATFORMS[Data Platforms]  
 PIPELINE\_TOOLS[Pipeline Tools]  
 MONITORING[Monitoring Systems]  
 NOTIFICATIONS[Alert Systems]  
 end  
   
 WEB\_UI --> GATEWAY  
 API --> GATEWAY  
 CLI --> GATEWAY  
 NOTEBOOKS --> GATEWAY  
   
 GATEWAY --> AUTH  
 GATEWAY --> RATE\_LIMIT  
 GATEWAY --> AUDIT  
   
 GATEWAY --> PROFILER  
 GATEWAY --> QUALITY\_ASSESSOR  
 GATEWAY --> ANOMALY\_DETECTOR  
 GATEWAY --> IMPUTATION\_ENGINE  
 GATEWAY --> DEDUP\_ENGINE  
 GATEWAY --> LINEAGE\_TRACKER  
   
 ANOMALY\_DETECTOR --> OUTLIER\_DETECTION  
 QUALITY\_ASSESSOR --> PATTERN\_LEARNING  
 IMPUTATION\_ENGINE --> IMPUTATION\_ML  
 DEDUP\_ENGINE --> SIMILARITY\_ENGINE  
 LINEAGE\_TRACKER --> CAUSAL\_INFERENCE  
 PROFILER --> PREDICTIVE\_QUALITY  
   
 PROFILER --> SPARK\_CLUSTER  
 QUALITY\_ASSESSOR --> TASK\_SCHEDULER  
 ANOMALY\_DETECTOR --> DISTRIBUTED\_COMPUTE  
 IMPUTATION\_ENGINE --> GPU\_ACCELERATOR  
 LINEAGE\_TRACKER --> STREAMING\_PROCESSOR  
   
 PROFILER --> METADATA\_DB  
 QUALITY\_ASSESSOR --> QUALITY\_METRICS  
 LINEAGE\_TRACKER --> LINEAGE\_GRAPH  
 ANOMALY\_DETECTOR --> RULES\_STORE  
 IMPUTATION\_ENGINE --> CACHE\_LAYER  
 DEDUP\_ENGINE --> OBJECT\_STORAGE  
   
 PROFILER --> DATA\_SOURCES  
 LINEAGE\_TRACKER --> DATA\_PLATFORMS  
 QUALITY\_ASSESSOR --> PIPELINE\_TOOLS  
 ANOMALY\_DETECTOR --> MONITORING  
 QUALITY\_ASSESSOR --> NOTIFICATIONS

## HLD (High Level Design)

### Core Data Quality Architecture

class DataQualityPlatform:  
 def \_\_init\_\_(self):  
 self.profiler = DataProfiler()  
 self.quality\_assessor = QualityAssessor()  
 self.anomaly\_detector = AnomalyDetector()  
 self.imputation\_engine = ImputationEngine()  
 self.deduplication\_engine = DeduplicationEngine()  
 self.lineage\_tracker = LineageTracker()  
 self.root\_cause\_analyzer = RootCauseAnalyzer()  
   
 async def comprehensive\_quality\_assessment(self, dataset, assessment\_config):  
 # Step 1: Profile the dataset  
 data\_profile = await self.profiler.profile\_dataset(dataset)  
   
 # Step 2: Assess data quality across multiple dimensions  
 quality\_assessment = await self.quality\_assessor.assess\_quality(  
 dataset, data\_profile, assessment\_config  
 )  
   
 # Step 3: Detect anomalies and outliers  
 anomalies = await self.anomaly\_detector.detect\_anomalies(  
 dataset, data\_profile  
 )  
   
 # Step 4: Identify duplicates  
 duplicates = await self.deduplication\_engine.find\_duplicates(dataset)  
   
 # Step 5: Analyze missing values  
 missing\_analysis = await self.analyze\_missing\_patterns(dataset)  
   
 # Step 6: Root cause analysis  
 root\_causes = await self.root\_cause\_analyzer.analyze\_issues(  
 quality\_assessment, anomalies, duplicates, missing\_analysis  
 )  
   
 # Step 7: Generate remediation recommendations  
 recommendations = await self.generate\_remediation\_plan(  
 quality\_assessment, anomalies, duplicates, missing\_analysis, root\_causes  
 )  
   
 return ComprehensiveQualityReport(  
 data\_profile=data\_profile,  
 quality\_assessment=quality\_assessment,  
 anomalies=anomalies,  
 duplicates=duplicates,  
 missing\_analysis=missing\_analysis,  
 root\_causes=root\_causes,  
 recommendations=recommendations  
 )  
  
class QualityAssessor:  
 def \_\_init\_\_(self):  
 self.completeness\_checker = CompletenessChecker()  
 self.validity\_checker = ValidityChecker()  
 self.consistency\_checker = ConsistencyChecker()  
 self.accuracy\_checker = AccuracyChecker()  
 self.uniqueness\_checker = UniquenessChecker()  
   
 async def assess\_quality(self, dataset, data\_profile, config):  
 quality\_dimensions = {}  
   
 # Completeness assessment  
 completeness\_score = await self.completeness\_checker.assess(dataset)  
 quality\_dimensions['completeness'] = completeness\_score  
   
 # Validity assessment  
 validity\_score = await self.validity\_checker.assess(dataset, data\_profile)  
 quality\_dimensions['validity'] = validity\_score  
   
 # Consistency assessment  
 consistency\_score = await self.consistency\_checker.assess(dataset, config.consistency\_rules)  
 quality\_dimensions['consistency'] = consistency\_score  
   
 # Accuracy assessment (if reference data available)  
 if config.reference\_data:  
 accuracy\_score = await self.accuracy\_checker.assess(dataset, config.reference\_data)  
 quality\_dimensions['accuracy'] = accuracy\_score  
   
 # Uniqueness assessment  
 uniqueness\_score = await self.uniqueness\_checker.assess(dataset)  
 quality\_dimensions['uniqueness'] = uniqueness\_score  
   
 # Calculate overall quality score  
 overall\_score = self.calculate\_weighted\_quality\_score(  
 quality\_dimensions, config.dimension\_weights  
 )  
   
 return QualityAssessmentResult(  
 overall\_score=overall\_score,  
 dimension\_scores=quality\_dimensions,  
 detailed\_results=self.generate\_detailed\_quality\_report(quality\_dimensions)  
 )  
  
class AnomalyDetector:  
 def \_\_init\_\_(self):  
 self.numerical\_detectors = {  
 'isolation\_forest': IsolationForest(),  
 'local\_outlier\_factor': LocalOutlierFactor(),  
 'one\_class\_svm': OneClassSVM()  
 }  
 self.categorical\_detector = CategoricalAnomalyDetector()  
 self.text\_detector = TextAnomalyDetector()  
 self.multivariate\_detector = MultivariateAnomalyDetector()  
   
 async def detect\_anomalies(self, dataset, data\_profile):  
 anomaly\_results = {}  
   
 for column in dataset.columns:  
 column\_profile = data\_profile.column\_profiles[column]  
   
 if column\_profile.data\_type == 'numerical':  
 anomalies = await self.detect\_numerical\_anomalies(  
 dataset[column], column\_profile  
 )  
 elif column\_profile.data\_type == 'categorical':  
 anomalies = await self.detect\_categorical\_anomalies(  
 dataset[column], column\_profile  
 )  
 elif column\_profile.data\_type == 'text':  
 anomalies = await self.detect\_text\_anomalies(  
 dataset[column], column\_profile  
 )  
   
 anomaly\_results[column] = anomalies  
   
 # Multivariate anomaly detection  
 multivariate\_anomalies = await self.multivariate\_detector.detect(dataset)  
 anomaly\_results['multivariate'] = multivariate\_anomalies  
   
 return AnomalyDetectionResult(  
 column\_anomalies=anomaly\_results,  
 total\_anomalies=sum(len(anomalies.outlier\_indices) for anomalies in anomaly\_results.values()),  
 anomaly\_summary=self.summarize\_anomalies(anomaly\_results)  
 )  
   
 async def detect\_numerical\_anomalies(self, column\_data, column\_profile):  
 """Ensemble approach for numerical anomaly detection"""  
 anomaly\_scores = {}  
 outlier\_votes = np.zeros(len(column\_data))  
   
 # Clean data (remove nulls for analysis)  
 clean\_data = column\_data.dropna().values.reshape(-1, 1)  
 clean\_indices = column\_data.dropna().index  
   
 for detector\_name, detector in self.numerical\_detectors.items():  
 # Fit detector and predict anomalies  
 outliers = detector.fit\_predict(clean\_data)  
   
 # Map back to original indices  
 for i, idx in enumerate(clean\_indices):  
 if outliers[i] == -1: # Anomaly detected  
 outlier\_votes[idx] += 1  
   
 # Get anomaly scores if available  
 if hasattr(detector, 'decision\_function'):  
 scores = detector.decision\_function(clean\_data)  
 anomaly\_scores[detector\_name] = dict(zip(clean\_indices, scores))  
   
 # Ensemble decision: majority vote  
 outlier\_threshold = len(self.numerical\_detectors) / 2  
 outlier\_indices = np.where(outlier\_votes >= outlier\_threshold)[0].tolist()  
   
 return NumericalAnomalyResult(  
 outlier\_indices=outlier\_indices,  
 anomaly\_scores=anomaly\_scores,  
 ensemble\_votes=outlier\_votes.tolist(),  
 detection\_summary=f"Found {len(outlier\_indices)} outliers using ensemble approach"  
 )  
  
class ImputationEngine:  
 def \_\_init\_\_(self):  
 self.imputers = {  
 'mean': MeanImputer(),  
 'median': MedianImputer(),  
 'mode': ModeImputer(),  
 'knn': KNNImputer(),  
 'mice': MICEImputer(),  
 'ml\_based': MLBasedImputer(),  
 'time\_series': TimeSeriesImputer()  
 }  
 self.imputation\_selector = ImputationMethodSelector()  
   
 async def impute\_missing\_values(self, dataset, imputation\_config):  
 imputation\_results = {}  
   
 for column in dataset.columns:  
 if dataset[column].isnull().sum() > 0:  
 # Select best imputation method for this column  
 best\_method = await self.imputation\_selector.select\_method(  
 dataset, column, imputation\_config  
 )  
   
 # Perform imputation  
 imputed\_values, confidence\_scores = await self.perform\_imputation(  
 dataset, column, best\_method, imputation\_config  
 )  
   
 imputation\_results[column] = ImputationResult(  
 method\_used=best\_method,  
 imputed\_values=imputed\_values,  
 confidence\_scores=confidence\_scores,  
 missing\_count=dataset[column].isnull().sum()  
 )  
   
 return ImputationResults(  
 column\_results=imputation\_results,  
 overall\_improvement=self.calculate\_imputation\_improvement(dataset, imputation\_results)  
 )  
   
 async def perform\_imputation(self, dataset, column, method, config):  
 """Perform imputation with confidence scoring"""  
 imputer = self.imputers[method]  
   
 # Prepare data for imputation  
 missing\_mask = dataset[column].isnull()  
   
 if method in ['knn', 'mice', 'ml\_based']:  
 # Use other columns as features  
 feature\_columns = [col for col in dataset.columns if col != column]  
 X = dataset[feature\_columns].fillna(dataset[feature\_columns].mean()) # Simple preprocessing  
   
 # Fit imputer  
 imputed\_values = imputer.fit\_transform(X, dataset[column])  
   
 # Calculate confidence scores based on cross-validation  
 confidence\_scores = await self.calculate\_imputation\_confidence(  
 X, dataset[column], imputer, missing\_mask  
 )  
 else:  
 # Simple imputation methods  
 imputed\_values = imputer.fit\_transform(dataset[column])  
 confidence\_scores = [0.7] \* missing\_mask.sum() # Fixed confidence for simple methods  
   
 return imputed\_values[missing\_mask], confidence\_scores  
  
class DeduplicationEngine:  
 def \_\_init\_\_(self):  
 self.similarity\_calculator = SimilarityCalculator()  
 self.blocking\_engine = BlockingEngine()  
 self.matching\_engine = MatchingEngine()  
 self.clustering\_engine = ClusteringEngine()  
   
 async def find\_duplicates(self, dataset, dedup\_config=None):  
 # Step 1: Generate blocking keys to reduce comparison space  
 blocks = await self.blocking\_engine.create\_blocks(dataset, dedup\_config)  
   
 # Step 2: Calculate similarities within blocks  
 similarity\_pairs = []  
   
 for block\_key, block\_records in blocks.items():  
 if len(block\_records) > 1:  
 block\_similarities = await self.calculate\_block\_similarities(  
 dataset, block\_records, dedup\_config  
 )  
 similarity\_pairs.extend(block\_similarities)  
   
 # Step 3: Apply matching threshold  
 potential\_matches = [  
 pair for pair in similarity\_pairs   
 if pair.similarity\_score >= dedup\_config.similarity\_threshold  
 ]  
   
 # Step 4: Cluster similar records  
 duplicate\_clusters = await self.clustering\_engine.cluster\_duplicates(  
 potential\_matches, dedup\_config  
 )  
   
 # Step 5: Generate deduplication recommendations  
 dedup\_recommendations = await self.generate\_dedup\_recommendations(  
 duplicate\_clusters, dataset  
 )  
   
 return DeduplicationResult(  
 duplicate\_clusters=duplicate\_clusters,  
 total\_duplicates=sum(len(cluster.record\_ids) - 1 for cluster in duplicate\_clusters),  
 recommendations=dedup\_recommendations,  
 similarity\_distribution=self.analyze\_similarity\_distribution(similarity\_pairs)  
 )

## LLD (Low Level Design)

### Advanced Quality Assessment Algorithms

```python class MLBasedQualityAssessor: def **init**(self): self.pattern\_detector = PatternDetectionModel() self.quality\_predictor = QualityPredictionModel() self.anomaly\_explainer = AnomalyExplainer()

async def assess\_data\_quality\_ml(self, dataset, historical\_patterns):  
 """Use ML to assess data quality based on learned patterns"""  
   
 # Extract features for quality assessment  
 quality\_features = self.extract\_quality\_features(dataset)  
   
 # Detect known patterns and deviations  
 pattern\_analysis = await self.pattern\_detector.analyze\_patterns(  
 quality\_features, historical\_patterns  
 )  
   
 # Predict quality scores using trained model  
 predicted\_quality = await self.quality\_predictor.predict\_quality(  
 quality\_features, pattern\_analysis  
 )  
   
 # Generate explanations for quality issues  
 quality\_explanations = await self.anomaly\_explainer.explain\_quality\_issues(  
 dataset, quality\_features, predicted\_quality  
 )  
   
 return MLQualityAssessment(  
 predicted\_scores=predicted\_quality,  
 pattern\_analysis=pattern\_analysis,  
 explanations=quality\_explanations,  
 confidence\_intervals=self.calculate\_prediction\_confidence(predicted\_quality)  
 )  
  
def extract\_quality\_features(self, dataset):  
 """Extract comprehensive features for quality assessment"""  
 features = {}  
   
 for column in dataset.columns:  
 col\_features = {}  
   
 # Basic statistics  
 col\_features['null\_rate'] = dataset[column].isnull().mean()  
 col\_features['unique\_rate'] = dataset[column].nunique() / len(dataset)  
   
 if dataset[column].dtype in ['int64', 'float64']:  
 # Numerical features  
 col\_features['mean'] = dataset[column].mean()  
 col\_features['std'] = dataset[column].std()  
 col\_features['skewness'] = dataset[column].skew()  
 col\_features['kurtosis'] = dataset[column].kurtosis()  
 col\_features['outlier\_rate'] = self.calculate\_outlier\_rate(dataset[column])  
   
 elif dataset[column].dtype == 'object':  
 # Categorical/text features  
 col\_features['mode\_frequency'] = dataset[column].value\_counts().iloc[0] / len(dataset)  
 col\_features['entropy'] = self.calculate\_entropy(dataset[column])  
 col\_features['avg\_length'] = dataset[column].astype(str).str.len().mean()  
   
 features[column] = col\_features  
   
 # Cross-column features  
 features['correlation\_matrix'] = dataset.select\_dtypes(include=[np.number]).corr().values.flatten()  
 features['duplicate\_rate'] = dataset.duplicated().mean()  
   
 return features

class AdvancedImputationEngine: def **init**(self): self.neural\_imputer = NeuralNetworkImputer() self.collaborative\_imputer = CollaborativeFilteringImputer() self.context\_aware\_imputer = ContextAwareImputer()

async def advanced\_imputation(self, dataset, column, context\_data=None):  
 """Advanced ML-based imputation with multiple strategies"""  
   
 # Strategy 1: Neural network imputation  
 nn\_imputation = await self.neural\_imputer.impute(dataset, column)  
   
 # Strategy 2: Collaborative filtering (for user-item like data)  
 if self.is\_collaborative\_applicable(dataset, column):  
 cf\_imputation = await self.collaborative\_imputer.impute(dataset, column)  
 else:  
 cf\_imputation = None  
   
 # Strategy 3: Context-aware imputation using external data  
 if context\_data:  
 context\_imputation = await self.context\_aware\_imputer.impute(  
 dataset, column, context\_data  
 )  
 else:  
 context\_imputation = None  
   
 # Ensemble the results  
 final\_imputation = self.ensemble\_imputations(  
 [nn\_imputation, cf\_imputation, context\_imputation]  
 )  
   
 return AdvancedImputationResult(  
 imputed\_values=final\_imputation.values,  
 confidence\_scores=final\_imputation.confidence,  
 method\_contributions=final\_imputation.method\_weights,  
 uncertainty\_estimates=final\_imputation.uncertainty  
 )

class RealTimeQualityMonitor: def **init**(self): self.streaming\_profiler = StreamingDataProfiler() self.quality\_tracker = QualityMetricTracker() self.alert\_manager = QualityAlertManager() self.drift\_detector = QualityDriftDetector()

async def monitor\_streaming\_quality(self, data\_stream, monitoring\_config):  
 """Monitor data quality in real-time streaming data"""  
   
 async for batch in data\_stream:  
 # Profile incoming batch  
 batch\_profile = await self.streaming\_profiler.profile\_batch(batch)  
   
 # Update quality metrics  
 current\_metrics = await self.quality\_tracker.update\_metrics(  
 batch\_profile, monitoring\_config.baseline\_metrics  
 )  
   
 # Detect quality drift  
 drift\_result = await self.drift\_detector.detect\_drift(  
 current\_metrics, monitoring\_config.drift\_thresholds  
 )  
   
 # Check for quality violations  
 violations = self.check\_quality\_violations(  
 current\_metrics, monitoring\_config.quality\_slas  
 )  
   
 # Trigger alerts if necessary  
 if violations or drift\_result.significant\_drift:  
 await self.alert\_manager.trigger\_quality\_alert(  
 violations, drift\_result, current\_metrics  
 )  
   
 # Store metrics for historical analysis  
 await self.store\_quality\_metrics(current\_metrics, batch.timestamp)  
   
 return StreamingQualityReport(  
 processed\_batches=self.quality\_tracker.total\_batches,  
 quality\_trends=self.quality\_tracker.get\_trend\_analysis(),  
 alert\_summary=self.alert\_manager.get\_alert\_summary()  
 )

# Database Schema

class DataQualitySchema: def **init**(self): self.tables = """ – Data quality assessments CREATE TABLE quality\_assessments ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), dataset\_id UUID NOT NULL, assessment\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, overall\_quality\_score DECIMAL(5,4) NOT NULL, completeness\_score DECIMAL(5,4), validity\_score DECIMAL(5,4), consistency\_score DECIMAL(5,4), accuracy\_score DECIMAL(5,4), uniqueness\_score DECIMAL(5,4), assessment\_config JSONB NOT NULL, detailed\_results JSONB NOT NULL, created\_by UUID NOT NULL );

-- Quality issues  
 CREATE TABLE quality\_issues (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 assessment\_id UUID REFERENCES quality\_assessments(id) ON DELETE CASCADE,  
 issue\_type VARCHAR(100) NOT NULL,  
 issue\_severity VARCHAR(20) NOT NULL,  
 affected\_columns TEXT[] NOT NULL,  
 affected\_rows INTEGER[],  
 issue\_description TEXT NOT NULL,  
 root\_cause\_analysis JSONB,  
 remediation\_suggestions JSONB NOT NULL,  
 issue\_status VARCHAR(50) DEFAULT 'open',  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 resolved\_at TIMESTAMP  
 );  
   
 -- Anomaly det# 140509\_40.md - AI-Powered Data Quality an