

# Drug Classification Prediction Pipeline - Team Contributions Report

---

**Project:** Database - Prediction Pipeline  
**Course:** Formative 1  
**Team:** Group 11  
**GitHub Repository:** <https://github.com/excelasaph/-Database---Prediction-Pipeline---Group-11>  
**Dataset Source:** [Drug Classification Dataset](#)

---

## Executive Summary

This report details the comprehensive implementation of a drug classification prediction pipeline that integrates both relational (PostgreSQL) and NoSQL (MongoDB) databases with machine learning capabilities. The project successfully fulfills all assignment requirements across four main tasks, with each team member contributing significantly to different aspects of the system.

### Project Overview

- **Database Design:** 5-table normalized schema with stored procedures and triggers
  - **API Development:** FastAPI with comprehensive CRUD operations and dual database support
  - **Machine Learning:** Neural network-based prediction pipeline with multiple model comparison
  - **Automation:** Script-based prediction system with error handling and validation
- 

## Team Member Contributions

### 1. Excel Asaph - Database Design & API Development

#### Task 1: Database Implementation (PostgreSQL & MongoDB)

##### Database Schema Design:

- Designed a 5-table normalized schema following 3NF
- Implemented primary and foreign key relationships
- Created data validation constraints for all fields
- Designed ERD diagram using pgadmin functionality

##### PostgreSQL Implementation:

```
-- Core tables with proper relationships
CREATE TABLE Patients (
    patient_id SERIAL PRIMARY KEY,
    sex_id INTEGER REFERENCES Sexes(sex_id),
    bp_id INTEGER REFERENCES BloodPressures(bp_id),
    cholesterol_id INTEGER REFERENCES Cholesterols(cholesterol_id),
    age INTEGER NOT NULL CHECK (age >= 15 AND age <= 74),
```

```
na_to_k FLOAT NOT NULL CHECK (na_to_k >= 6.0 AND na_to_k <= 40.0)
);
```

### Stored Procedure & Trigger Implementation:

- Created `log_new_patient` stored procedure for automated logging
- Implemented `patient_insert_trigger` trigger for automatic procedure execution
- Ensured error handling and transaction management

### MongoDB Implementation:

- Designed equivalent collections in MongoDB
- Implemented proper document structure with embedded references
- Created data loading scripts for MongoDB population

## Task 2: FastAPI Development

### API Architecture:

- Implemented CRUD operations for both databases
- Created dual database support with automatic fallback
- Added input validation using Pydantic models
- Implemented error handling and HTTP status codes

### Key Endpoints Developed:

```
# PostgreSQL CRUD endpoints
@app.post("/patients/")          # Create patient
@app.get("/patients/{patient_id}") # Read patient
@app.put("/patients/{patient_id}") # Update patient
@app.delete("/patients/{patient_id}") # Delete patient
@app.get("/patients/latest")      # Get latest patient

# MongoDB CRUD endpoints
@app.post("/mongo/patients/")     # Create patient
@app.get("/mongo/patients/{patient_id}") # Read patient
@app.put("/mongo/patients/{patient_id}") # Update patient
@app.delete("/mongo/patients/{patient_id}") # Delete patient

# Prediction endpoint
@app.post("/predict")             # Make predictions
```

### Technical Achievements:

- Implemented automatic database fallback (PostgreSQL → MongoDB)
- Created input validation with Pydantic
- Added proper error handling for all edge cases
- Ensured API documentation with automatic OpenAPI generation

### Commit History:

- 11+ commits with clear, descriptive messages
  - Focused on database design and API implementation
- 

## 2. Nicolle Marizani - Machine Learning Pipeline

### Task 3: Machine Learning Implementation

#### Model Training Pipeline:

- Developed comprehensive Jupyter notebook for model training
- Implemented multiple machine learning algorithms:
  - Logistic Regression (Accuracy: 87.5%)
  - Gradient Boosting (Accuracy: 97.5%)
  - Random Forest (Accuracy: 92.5%)
  - Neural Network (Accuracy: 82.5%)

#### Data Preprocessing:

```
# Feature engineering pipeline
numerical_features = ["Age", "Na_to_K"]
categorical_features = ["Sex", "BP", "Cholesterol"]

preprocessor = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", cat_pipeline, categorical_features),
])
```

#### Model Optimization:

- Implemented GridSearchCV for hyperparameter tuning
- Created cross-validation procedures for robust evaluation
- Developed model comparison and selection framework
- Saved optimized models for production deployment

#### Neural Network Implementation:

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_proc.shape[1],)),
    Dropout(0.2),
    Dense(5, activation='softmax')
])
```

#### Technical Achievements:

- Achieved 97.5% validation accuracy with Gradient Boosting

- Implemented proper train/validation/test splits (60:20:20)
- Created model evaluation metrics
- Developed automated model saving and loading procedures

**Commit History:**

- 5+ commits focused on ML pipeline development
  - Clear documentation of model training procedures
- 

### 3. Chance Karambizi - Prediction Logs Endpoint

**Task 4: Prediction Logs Handling****Schema & Endpoint Development:**

- Designed PredictionLogs schema to handle prediction logging input
- Implemented /predictionlogs/ POST endpoint
- Enabled support for both PostgreSQL and MongoDB backends

**Debugging & Error Handling:**

- Fixed issues with DB insert logic
- Improved timestamp handling across databases
- Added validation for db\_used field
- Ensured clear and consistent error responses

**Technical Contributions:**

- Integrated dual-DB logging system for predictions
- Verified schema consistency and DB compatibility
- Handled conditional DB routing logic (db\_used field)

**Commit History:**

- 5+ commits focused on documentation
  - Maintained code quality standards
- 

### 4. Diana Ruzindana - Pipeline Validation & Latest Entry Endpoint

**Task 4: Fetching Data and Validation Scripts****Code Review & Standards:**

- Conducted detailed review of Python scripts involved in the drug prediction pipeline.
- Ensured consistent formatting, meaningful function names, and modular logic.
- Verified that requests and responses in the API were handled using clear, fail-safe conditions.
- Confirmed all scripts followed PEP 8 standards and project naming conventions.

**Integration Testing:**

- Tested full pipeline execution including:
  - API availability (/docs)
  - Fetching the latest patient
  - Making predictions with model type "nn"
  - Displaying formatted prediction results
- Confirmed the API handles errors (connection failures, 404s, etc.) gracefully.
- Validated MongoDB collection structure and schema via automated validation scripts.

### Quality Assurance Achievements:

- Ensured reliable patient-fetch and prediction workflows under various scenarios.
- Verified robustness of exception handling and fallback mechanisms.
- Confirmed display output matches expected format and highlights prediction correctness.
- Validated MongoDB data integrity with custom collection validation scripts.

### Commit History:

- 5+ commits focused on API endpoint handling
- Clear documentation of testing procedures

---

## GitHub Repository Information

**Repository:** <https://github.com/excelasaph/-Database---Prediction-Pipeline---Group-11>

**Main Branch:** main

**Contributors:** 4 team members

**Total Commits:** 30+ commits

**Languages:** Python, SQL, Markdown

**Technologies:** FastAPI, PostgreSQL, MongoDB, TensorFlow, scikit-learn