Project 9: Classical ML Pipeline

Objective (Why?)

Build a simple but complete machine learning pipeline in 3 days that handles data preprocessing, model training, and API deployment. This project establishes ML foundations using popular, well-documented libraries with plenty of online support. You will practice:

- **Classical ML Algorithms**: Using scikit-learn for Random Forest, XGBoost, and Linear models
- **Data Pipeline Basics**: Simple preprocessing with pandas and scikit-learn
- **Model API Creation**: Building a FastAPI endpoint for model predictions
- **Model Saving/Loading**: Storing and retrieving trained models for reuse

Core Requirements (Simplified)

Component	Requirement
Data Processing	Load CSV, handle missing values, basic feature engineering
Model Training	Train 3 models (Random Forest, XGBoost, Logistic Regression) and compare
Model Evaluation	Use accuracy, confusion matrix, and basic metrics
API Deployment	FastAPI endpoint for making predictions

Development Approach: Milestone-Based Progression

Philosophy: Focus on **deliverable quality** and **comprehensive review compliance** rather than rigid timelines. Each milestone must pass all relevant review templates before proceeding.

Milestone 1: Data Pipeline & Feature Engineering

Estimated Time: 6-8 hours (flexible based on learning pace)

Deliverables:

- ☐ Complete data loading and validation system with support for CSV, Excel, and database sources
- ☐ Comprehensive data exploration and statistical analysis with visualizations
- ☐ Automated data validation with statistical tests and quality metrics
- ☐ Feature engineering pipeline with encoding, scaling, and transformation functions
- ☐ Proper data splitting with stratification and statistical validation

Review Requirements (Must Pass to Proceed):

- ☐ **Data Science Review**: Data methodology and statistical rigor assessment
- ☐ **Architecture Review**: Clean pipeline design and modularity evaluation
- ☐ **Performance Review**: Efficient data processing and memory management

Milestone 2: Model Training & Evaluation

Estimated Time: 6-8 hours (flexible based on Milestone 1 completion)

Deliverables:

- ☐ Multiple ML algorithm implementations (Random Forest, XGBoost, Linear models)
- ☐ Robust cross-validation framework with multiple metrics
- ☐ Hyperparameter optimization using GridSearchCV or RandomizedSearchCV
- ☐ Comprehensive model evaluation with performance visualization
- ☐ Statistical comparison of models with significance testing

Review Requirements (Must Pass to Proceed):

- ☐ **Data Science Review**: Model evaluation methodology and statistical rigor
- ☐ **Performance Review**: Training efficiency and optimization effectiveness
- ☐ **Code Quality Review**: Clean implementation and maintainable patterns

Milestone 3: API Deployment & Production Features

Estimated Time: 4-6 hours (flexible based on previous milestones)

Deliverables:

- ☐ FastAPI deployment system with prediction endpoints and documentation
- ☐ Model persistence and loading with metadata storage
- ☐ Comprehensive input validation and error handling
- ☐ Performance monitoring and logging system
- ☐ Production deployment preparation and documentation

Review Requirements (Must Pass for Project Completion):

- ☐ **Architecture Review**: Complete ML pipeline architecture assessment
- ☐ **Performance Review**: API performance and scalability evaluation
- ☐ **Security Review**: Input validation and deployment security
- ☐ **Code Quality Review**: Production-ready code standards

Milestone Progression Rules:

- **Cannot advance** to next milestone without passing all review requirements
- **Flexible timing** allows for learning at individual pace
- **Quality gates** ensure each milestone meets professional standards
- **Mentor support** available for concept clarification and review failures

Measurable Goals & Review Template Compliance

Primary Objectives (Must Complete for Project Advancement)

- ☐ **ML Pipeline Excellence**: Pass Data Science Review with 9.0/10+ score
- ☐ **Architecture Quality**: Pass Architecture Review with 8.5/10+ score
- ☐ **API Performance**: Sub-100ms prediction response times
- ☐ **Model Accuracy**: Achieve >85% accuracy on test dataset
- ☐ **Code Quality Standards**: Pass Code Quality Review with 8.5/10+ score

Review Template Integration (All Must Pass)

Data Science Review Requirements (Critical for ML Pipeline)

```python
data_science_criteria = {
    "feature_engineering": {"target": 90, "weight": 30},      # Feature quality and selection
    "model_evaluation": {"target": 90, "weight": 25},         # Proper validation methodology
    "preprocessing_quality": {"target": 85, "weight": 20},    # Data cleaning and transformation
    "statistical_rigor": {"target": 85, "weight": 15},        # Cross-validation and metrics
    "interpretability": {"target": 80, "weight": 10}          # Model explanation capability
}
```

Phase Progression Requirements

Project 9 → Project 10 Advancement Requirements

Mandatory Review Template Compliance:

- ☐ **Data Science Review**: Minimum 9.0/10 score (ML methodology)
- ☐ **Architecture Review**: Minimum 8.5/10 score (pipeline design)
- ☐ **Performance Review**: Minimum 8.0/10 score (processing efficiency)
- ☐ **Code Quality Review**: Minimum 8.5/10 score (clean implementation)

```python
advancement_requirements = {
    "review_compliance": {
        "data_science": {"minimum_score": 9.0, "weight": 40},
        "architecture": {"minimum_score": 8.5, "weight": 30},
        "performance": {"minimum_score": 8.0, "weight": 20},
        "code_quality": {"minimum_score": 8.5, "weight": 10}
    },
    "functional_requirements": {
        "model_accuracy_threshold": 0.85,
        "api_response_time_ms": 100,
        "cross_validation_implemented": True,
        "hyperparameter_tuning_complete": True
    }
```

```
}
```

Project 10 Preparation Requirements

- ☐ **LangChain Basics**: Understand LLM frameworks and agent concepts
- ☐ **API Integration**: Learn external API consumption patterns
- ☐ **Reasoning Patterns**: Study ReAct (Reasoning + Action) methodology
- ☐ **Tool Integration**: Understand function calling and tool selection

Technical Specifications

Data Processing Pipeline

- Implement comprehensive data loading with support for CSV, Excel, and database sources
- Create automated data validation with statistical tests and quality metrics
- Build feature engineering system with encoding, scaling, and transformation functions
- Design flexible preprocessing pipeline with configurable parameters and caching

Machine Learning Implementation

- Configure scikit-learn models with proper parameter tuning and cross-validation
- Implement XGBoost with early stopping and feature importance analysis
- Create model comparison framework with statistical significance testing
- Build hyperparameter optimization using GridSearchCV or RandomizedSearchCV

API Architecture

- Design RESTful FastAPI endpoints with proper HTTP status codes and documentation
- Implement Pydantic models for request/response validation and serialization
- Create model loading system with caching and efficient memory management
- Build comprehensive error handling with meaningful error messages and logging

Database Integration

- Store model metadata (accuracy, parameters, training date) in PostgreSQL
- Implement model versioning system with performance tracking
- Create prediction logging for monitoring and debugging
- Design efficient model artifact storage and retrieval

Project Structure

```
None
project-9-ml-pipeline/
├── src/
│   ├── data/
│   │   ├── loader.py         # Data loading utilities
│   │   ├── preprocessor.py   # Data preprocessing pipeline
│   │   └── validator.py      # Data validation functions
│   ├── models/
│   │   ├── base.py           # Base model interface
│   │   ├── classical.py      # Classical ML implementations
│   │   └── evaluator.py      # Model evaluation utilities
│   ├── api/
│   │   ├── main.py           # FastAPI application
│   │   ├── endpoints.py      # API endpoint definitions
│   │   └── schemas.py        # Pydantic models
│   └── utils/
│       ├── config.py         # Configuration management
│       ├── database.py       # Database operations
│       └── logging.py        # Logging configuration
├── data/                     # Dataset storage
├── models/                   # Trained model artifacts
├── notebooks/                # EDA and experimentation
├── tests/                    # Test suite
├── requirements.txt          # Python dependencies
└── README.md                 # Project documentation
```

Advanced Features (Stretch Goals)

- **Feature Importance Analysis**: SHAP values for model interpretability
- **Automated Model Selection**: AutoML-style automated algorithm selection

- **Model Ensemble**: Combine multiple models for improved performance
- **Real-time Retraining**: Automatic model updates based on new data
- **A/B Testing Framework**: Compare model performance in production
- **Advanced Preprocessing**: Automated feature engineering and selection

Deliverables

1. **Complete ML Pipeline** with data preprocessing and model training
2. **FastAPI Deployment** with prediction endpoints and documentation
3. **Model Evaluation Report** with performance analysis and comparison
4. **GitHub Repository** with comprehensive documentation and tests
5. **Live Demo** showing end-to-end ML workflow from data to prediction

Performance Requirements

Processing Performance

- Data preprocessing: < 30 seconds for 100MB datasets
- Model training: < 5 minutes for standard algorithms with tuning
- API response time: < 100ms for single predictions
- Batch prediction: < 1 second per 100 samples

Quality Standards

- Model accuracy: > 85% on test dataset (problem-dependent)
- Cross-validation stability: < 5% variance across folds
- API uptime: 99.9% availability during testing
- Error rate: < 1% for valid API requests

Testing Scenarios

Data Pipeline Testing

- Load various dataset formats (CSV, Excel, database)
- Handle missing values and data quality issues
- Validate feature engineering and preprocessing steps

- Test train/validation/test split consistency

Model Training Testing

- Verify algorithm implementations with known datasets
- Test hyperparameter tuning convergence and results
- Validate cross-validation methodology and metrics
- Compare model performance with statistical significance

API Integration Testing

- Test prediction endpoints with various input formats
- Validate error handling for invalid requests
- Test model loading and caching functionality
- Verify API documentation and response schemas

Security & Best Practices

Data Security

- Secure handling of sensitive data during preprocessing
- Proper validation of all user inputs and API requests
- Database connection security with environment variables
- Audit logging for all model predictions and operations

Code Quality

- Comprehensive unit and integration testing
- Type hints and documentation for all functions
- Error handling with meaningful messages and logging
- Code formatting and linting with professional standards

Sample Dataset Recommendations

- **Classification**: Iris, Titanic, or Boston Housing (well-known datasets)
- **Regression**: Boston Housing, California Housing, Sales Forecasting
- **Time Series**: Stock prices, Temperature data, Energy consumption

- **Business**: Customer segmentation, Fraud detection, Marketing response

Success Criteria Checklist

- Data pipeline processes various input formats without errors
- Multiple ML models train successfully with proper validation
- Cross-validation results show consistent performance across folds
- API endpoints respond within performance requirements
- Model predictions match expected accuracy thresholds
- Error handling provides meaningful feedback for all failure cases
- Documentation clearly explains system usage and architecture
- Code follows professional standards with comprehensive testing
- Database integration stores and retrieves model metadata correctly
- System demonstrates production-ready reliability and performance