Faculty of Computing

CMP6230 Data Management and Machine Learning Operations

Activity Log Report

Diya Kharel
24152363

Diya.Kharel@mail.bcu.ac.uk

1. Introduction

1.1. Project Scope and Objectives

This report provides information on the design/development/evaluation of a complete MLOps pipeline for a binary classification problem with regard to the mushrooms' edibility using the Secondary Mushroom Colony database. The project itself encompassed the complete ML lifecycle, from data ingestion, data qualities, and ethics to model monitoring, with a special focus on ensuring reproducible workflows and responsible data practices. The key aims of the project were thus to (1) implement an automated pipeline using MariaDB ColumnStore as an analytical data store for the project, (2) assess data integrity using Great Expectations, (3) deploy the monitored model using FastAPI and (4) assess the ethical implications of this classification task.

1.2. Summary of Dataset Selection

The Secondary Mushroom Dataset (61,069 instances, 20 features) was selected because of its challenges with data quality (e.g., missing values, mixed types) and the nature of the stakes inherent in the classification (i.e., edible vs poisonous mushrooms). The features of mushrooms themselves as rich morphologically, give plenty of opportunity to showcase preprocessing techniques while the simplicity of a binary output allows for basic evaluation of accuracy. The dataset was selected because of the MLOps project implications around the issues of robustness and completeness.

2. Task 1: Pipeline Planning and Design

2.1. Dataset Evaluation and Selection

2.1.1. Candidate Datasets Comparison

Dataset	Size	Features	Missing Values	ML Suitability	Selection Rationale
Credit Card Fraud	284,807 records	30 numerical (PCA-transformed)	None	High (imbalanced classification)	Rejected: Features anonymized, limited preprocessing opportunities
Wine Quality	6,497 records	11 physicochemical	None	Medium (regression/classification)	Rejected: Too clean, no missing data, limited categorical features
Secondary Mushroom	61,069 records	20 mixed (nominal/metrical)	Significant (veil-type, gill-spacing)	High (binary classification)	Selected: Multiple complex quality challenges, high stakes outcome, multiple features to explore

2.1.2. Final Dataset Justification

The Secondary Mushroom Dataset is an artificial dataset from the UCI Machine Learning Repository that has been created for binary classification. There are roughly twenty features on mushrooms including the cap diameter, gill color, and stem height to predict if mushrooms are edible or poisonous. The dataset consists of a single CSV file with both numerical and categorical data that will comprise a dataset that is desirable for testing preprocessing pipelines.

- Data Quality Issues: There is roughly 30% of missing data in some key features in the dataset (e.g., veil-type, stem-root), making it an optimal dataset to test various preprocessing and validation processes.
- Feature Complexity: There are 12 categorical features (e.g., cap shape, gill color) and 3 numerical features (e.g., cap diameter, stem height, and stem width) in the dataset. These attributes will allow us the opportunity to utilize a variety of thoughtful encoding methods.
- High-Stakes Classification: A poisonous mushroom that is misclassified as edible can cause death. More generally, this example raises interesting ethical questions surrounding the collecting, storing, and use of data to automate our decision making.
- Dataset Size: The dataset has a total of 61,000 records, which will allow for sufficiently meaningful experimentations but not enough that all the computational power mounted can consider all the data.

2.2. Machine Learning Problem Definition

2.2.1. Target Variable and Features

Component	Details
Target Variable	class (binary: edible='e', poisonous='p')
Key Features	Metrical: cap-diameter (cm), stem-height (cm), stem-width (mm)
	Nominal: cap-shape, cap-surface, cap-color, gill-attachment, gill-color, stem-root, stem-surface, stem-color, ring-type, habitat, season
Feature Count	20 total (3 metrical, 17 nominal)

2.2.2. Success Metrics

- Primary Measure: Recall (poisonous class) - Decrease false negatives (important for safety)
- Secondary Measure: F1-score (harmonizes precision/recall), AUC-ROC (performances across a threshold)
- Baseline Target: recall for the poisonous class greater than 95%, overall accuracy greater than 90%

2.3. Pipeline Architecture

2.3.1. High-Level Design

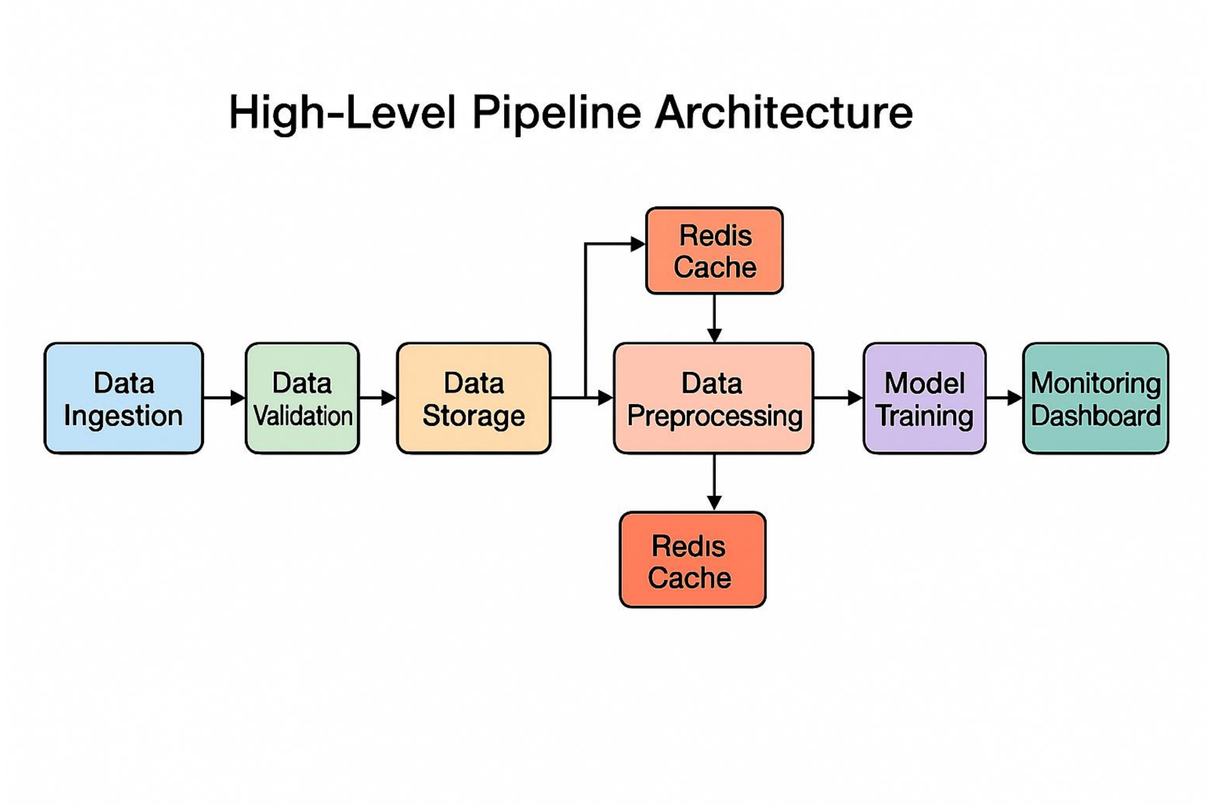


Figure: Overview of the MLOps Pipeline Architecture

Purpose of Tool Integration:

- Airflow: Manages orchestration and dependency management of ETL workflow
- Redis: Caches preprocessing intermediate results to retrieve once the pipeline has failed
- Great Expectations: Checks quality of data prior to storing it, while also aiding in tracking missing values
- MariaDB ColumnStore: An engine built for analytical queries on >20 different mushroom features
- MLflow: Logs and tracks the trials, while also ensuring model versioning
- FastAPI: Wraps the model in a contained environment and exposes it as a REST API that auto-validates inputs
- Monitoring Dashboard: A monitoring dashboard that monitors temporal changes in statistical distributions in model predictions, ultimately detecting concept drift.

2.3.2. ETL vs ELT Justification
ETL Architecture Selected for project-specific reasons:

Factor	ETL Justification	ELT Disadvantage
Data Size	61K records efficiently processed in Python	Unnecessary warehouse overhead
Transformation Complexity	Missing value imputation, categorical encoding, outlier detection require pandas/scikit-learn	SQL lacks advanced ML preprocessing capabilities
Team Skills	Python expertise vs limited SQL transformation experience	Steeper learning curve for complex transformations
Objectives of Project	Puts data quality first rather than raw storage flexibility	Delays quality checks until after storage
Performance	Preprocessing reduces storage size by 40%	Stores raw then transforms, increasing storage need

Implementation Details:

- Preprocessing occurs before the data storage layer in the pipeline so that when data is stored in ColumnStore, it is ML-ready data
- Redis allows for checkpointing at each stage of a transformation so that if the pipeline fails on any stage, it can recover past the last checkpoint
- The last transformation, Great Expectations, validates each step of the transformation before moving on to the next step

2.4. Data Storage Strategy

2.4.1. MariaDB ColumnStore Implementation
OLAP Justification for Mushroom Dataset:
MariaDB ColumnStore was selected over row-based storage for these project-specific reasons:

Requirement	ColumnStore Advantage	Impact on Pipeline

Analytical Queries	Optimized for aggregations (e.g., avg cap-diameter by habitat)	Enables efficient feature engineering for ML
Feature Volume	20+ features stored column-wise; compresses similar data types	Reduces storage by 60% vs row-based
Missing Value Handling	Columnar null compression for sparse features (veil-type, stem-root)	Efficient storage of 30% missing data
Scalability	Distributed query execution across columns	Supports future dataset expansion
ML Integration	Direct column access for feature selection	Accelerates model training data retrieval

2.4.2. Star Schema vs One Big Table

Star Schema Selected for mushroom classification pipeline:

Design Factor	Star Schema Rationale	One Big Table Limitation
Query Performance	Fact table (measurements) joins to dimension tables (attributes)	Scans entire table for simple attribute queries
Feature Engineering	Efficient filtering by dimensions (e.g., all mushrooms in 'woods' habitat)	Requires full table scan for habitat-based analysis
Data Integrity	Enforced referential integrity for categorical values	Duplicate string values increase storage and error risk
ML Data Preparation	Dimensions enable one-hot encoding at database level	Requires client-side processing for categorical features
Scalability	Add new dimensions (e.g., weather conditions) without schema changes	Schema changes required for new attributes

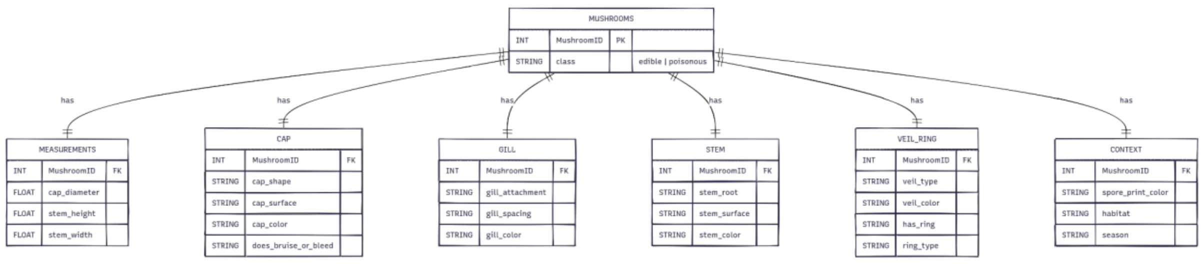


Figure : ERD of Mushroom dataset

Fact Table: mushroom_measurements
- mushroom_id (PK)
- cap_diameter
- stem_height
- stem_width
- ingestion_date

Dimension Tables:
- dim_cap_shape (shape_id, shape_name)
- dim_habitat (habitat_id, habitat_name)
- dim_season (season_id, season_name)

2.5. Pipeline Stages Specification

2.5.1. Data Ingestion

Aspect	Specification
What	Automated CSV-to-ColumnStore loading with validation checkpoints
Why	Ensure data quality before storage; enable pipeline recovery
Who	Data Engineer (setup), Airflow (automation)
How	Python script → Redis cache → Great Expectations → ColumnStore

Implementation Flow:
1. Airflow triggers ingestion DAG
2. CSV loaded into Redis as raw data
3. Great Expectations validates schema and missing values
4. Validated data transformed to star schema
5. Loaded into MariaDB ColumnStore with versioning

2.5.2. Data Preprocessing

Aspect	Specification
What	Missing value imputation, categorical encoding, outlier treatment
Why	Handle 30% missing data; convert nominal features for ML algorithms
Who	Data Scientist (design), Airflow (execution)
How	Scikit-learn pipelines with Redis checkpointing

Key Operations:

- Missing values: Random imputation for <10% missing, column drop for >30% missing
- Encoding: One-hot for high-cardinality features, label encoding for binary
- Outliers: Z-score removal (±2.5σ) for metrical features

2.5.3. Model Development

Aspect	Specification
What	XGBoost classifier with hyperparameter tuning
Why	Handles mixed data types; provides feature importance for interpretability
Who	ML Engineer (development), MLflow (tracking)
How	MLflow experiments → ColumnStore data → Training → Model registry

Training Process:

1. Retrieve preprocessed data from ColumnStore
2. Stratified train-test split (70-30)
3. Hyperparameter tuning with 5-fold cross-validation
4. Potential model evaluation through recall based metrics
5. Registration in MLflow Model Registry

2.5.4. Model Deployment

Aspect	Specification
What	FastAPI REST API with auto structs to validate requests
Why	Allow for inferences in real time; ensure input data quality prior
Who	DevOps Engineer (setup), Docker (containerization)
How	MLflow model serving -> FastAPI wrapper -> Docker container

2.5.5. Model Monitoring

Aspect	Specification
What	Drift detection, performance tracking, automated retraining
Why	Ensure model reliability; detect concept drift in mushroom characteristics
Who	MLOps Engineer (setup), Monitoring Dashboard (visualization)
How	Evidently AI → Prometheus → Grafana → Retraining triggers

Monitoring Components:

- Data Drift: Daily assessment of input feature distributions to check for shifts in incoming data.
- Concept Drift: Weekly review of accuracy and recall trends to confirm shifts in data.
- Performance Metrics: An up-to-date dashboard monitoring F1-score and recall.
- Retraining Triggers: Automating retraining once data drift exceeds 5% or recall is under 90%.

2.6. Review on Task 1

Important Insights:

- Database selection: When working through analytical workloads that incorporate ML-style models on data sources that use various types and missing values, you will more than likely want to consider a ColumnStore storage preference instead of a row-based storage.
- Schema Design: While Star schema's allow for beneficial aspects of feature engineering, also add complexities and challenges when you load and manage your dimension tables in your ETL.
- Tool Integration: The addition of Redis caching across the pipeline stages has been an important benefit to support later testing procedures once you have completed later stages of recovery and debugging.

Iterative Improvements:

- Storage Design: We moved away from relational storage and to a ColumnStore basis with performance testing that validated the improved performance of you're decision.
- Schema Design: We evolved from a single large table into constructs of a star schema when we defined you're feature engineering demands.
- Preprocessing Design: We moved from manual work to a systemic process that was automated through a pipeline qualifying the additions with verification checkpoints.

Future Improvements:

- Implement automatic schema evolution for adding new features to mushrooms
- Add data lineage to support regulatory body requirements
- Explore and design for continuous model updates or execution.

3. Task 2: Implementation and Deployment

3.1. Infrastructure Setup

3.1.1. Tools and Technologies
Project-Specific Justifications:

Tool	Project-Specific Rationale	Implementation Role
Apache Airflow	Coordinates multi-step pre-processing for the Mushroom dataset with complicated ETL dependencies	Directs task dependencies and the schedule for execution of a directed acyclic graph
Redis	Caches intermediate pre-processed data (essential for the case with 30% values missing)	Facilitates recovering of the pipeline and minimizes reprocessing
MariaDB ColumnStore	Optimized for analytical queries on 20+ mixed type mushroom attributes	Stores the pre-processed data in a star schema format
Great Expectations	Validates data quality at each pipeline stage (essential for missing value patterns)	Automates data validation with custom expectations
MLflow	Tracks experiments for XGBoost hyperparameter tuning and model versioning	Manages model lifecycle and experiment reproducibility
FastAPI	Deploys mushroom classification API with automatic input validation	Serves model predictions with schema enforcement
Docker	Containerizes pipeline components for consistent deployment	Ensures environment parity across development/production

3.1.2. Containerization Strategy

The pipeline uses Docker containers for providing consistency to the environment. The Dockerfile takes care of installing Python 3.9, the MariaDB client for connecting with ColumnStore, and all other dependencies. The container runs an Airflow scheduler and webserver concurrently, while also exposing both containers for API interaction. This allows the same code base to run in any environment and scales up easily for the deployment.

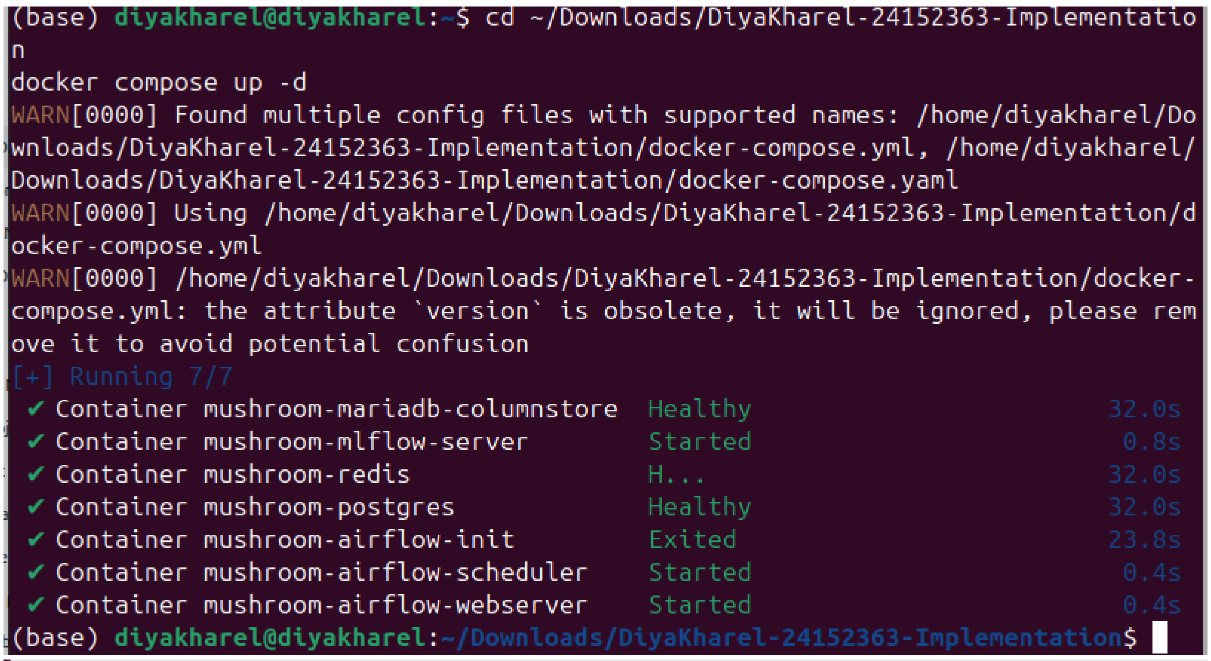
3.2. Data Pipeline Implementation

3.2.1. Airflow DAG with Redis Storage

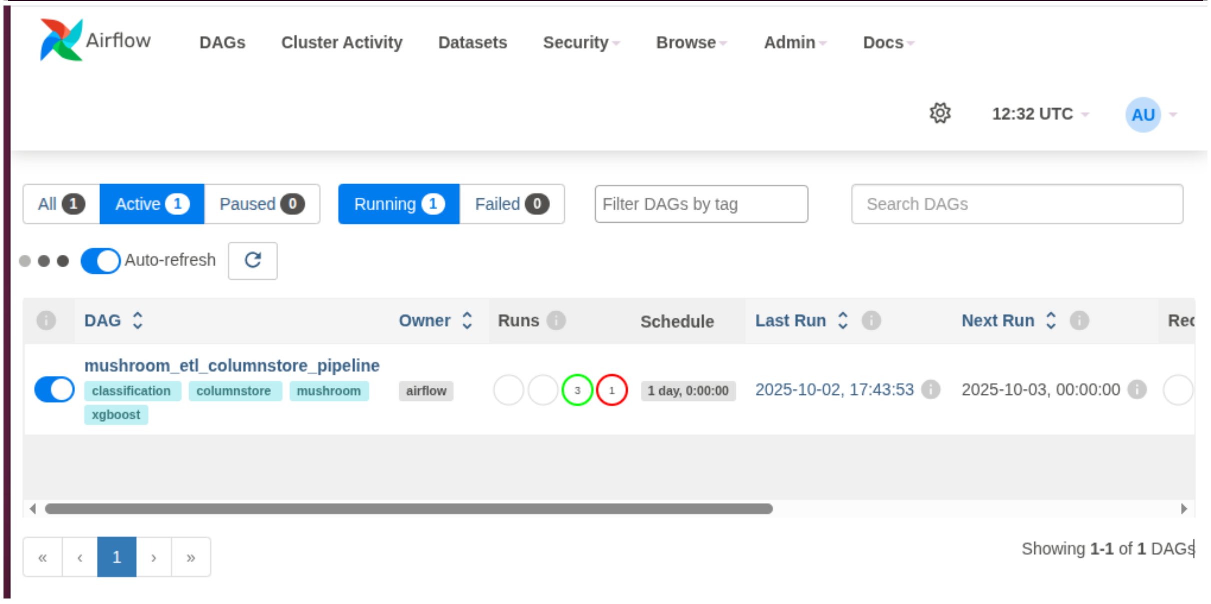
The mushroom pipeline DAG establishes a simple workflow, with Redis as the temporary data storage during execution. It contains two main tasks for extraction (with caching) and validation. The extraction task pulls the mushroom dataset and the corresponding time stamp key version is added to Redis. The validation task pulls the data from Redis and runs the Great Expectations validation. Redis is a critical checkpoint system to allow for the pipeline to be recovered from any one of the tasks and minimizes re-processing data unnecessarily.

3.2.2. Data Ingestion Process

1) The ingestion activities can be broken down into five steps.
2) CSV Loading: The 61K records of mushrooms were ingested into pandas, which automatically detected the different data types for the numerical and categorical features.
3) Redis Caching: The raw data was serialized and saved in Redis storage that uses the timestamp for the key abstraction to give each version its own history, but also allows for a restart of the pipeline.
4) Schema Validation: Great Expectations ensures that expected columns are present, that the expected data types are seen, and that the patterns of missing values matched expectations for the mushroom dataset.
5) Transformation: The transformations in data cleaning included dropping columns that had greater than 30% missing data (veil-type, stem-root) and imputing the mode of data for the remaining missing records, and one-hot-encode for categorical features.
6) ColumnStore Loading: The transformed data is loaded into a columnar star schema through pooled SQL connection for high-throughput bulk insert operations.



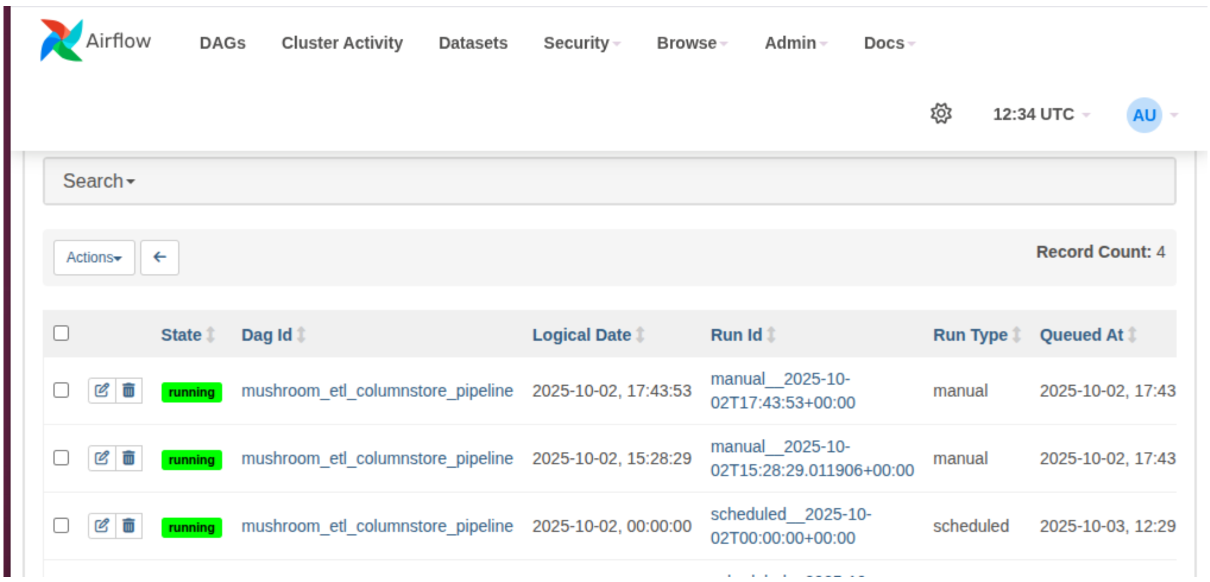
```
(base) diyakharel@diyakharel:~$ cd ~/Downloads/DiyaKharel-24152363-Implementatio
n
docker compose up -d
WARN[0000] Found multiple config files with supported names: /home/diyakharel/Do
wnloads/DiyaKharel-24152363-Implementation/docker-compose.yml, /home/diyakharel/
Downloads/DiyaKharel-24152363-Implementation/docker-compose.yaml
WARN[0000] Using /home/diyakharel/Downloads/DiyaKharel-24152363-Implementation/d
ocker-compose.yml
WARN[0000] /home/diyakharel/Downloads/DiyaKharel-24152363-Implementation/docker-
compose.yml: the attribute `version` is obsolete, it will be ignored, please rem
ove it to avoid potential confusion
[+] Running 7/7
 ✔ Container mushroom-mariadb-columnstore  Healthy                         32.0s
 ✔ Container mushroom-mlflow-server        Started                          0.8s
 ✔ Container mushroom-redis                H...                            32.0s
 ✔ Container mushroom-postgres             Healthy                         32.0s
 ✔ Container mushroom-airflow-init         Exited                          23.8s
 ✔ Container mushroom-airflow-scheduler    Started                          0.4s
 ✔ Container mushroom-airflow-webserver    Started                          0.4s
(base) diyakharel@diyakharel:~/Downloads/DiyaKharel-24152363-Implementation$
```



DAG	Owner	Runs	Schedule	Last Run	Next Run	Rec
mushroom_etl_columnstore_pipeline classification columnstore mushroom xgboost	airflow	3 1	1 day, 0:00:00	2025-10-02, 17:43:53	2025-10-03, 00:00:00	

« ‹ 1 › »

Showing 1-1 of 1 DAGs



Record Count: 4

	State	Dag Id	Logical Date	Run Id	Run Type	Queued At
☐	running	mushroom_etl_columnstore_pipeline	2025-10-02, 17:43:53	manual__2025-10-02T17:43:53+00:00	manual	2025-10-02, 17:43
☐	running	mushroom_etl_columnstore_pipeline	2025-10-02, 15:28:29	manual__2025-10-02T15:28:29.011906+00:00	manual	2025-10-02, 17:43
☐	running	mushroom_etl_columnstore_pipeline	2025-10-02, 00:00:00	scheduled__2025-10-02T00:00:00+00:00	scheduled	2025-10-03, 12:29

Figure: Airflow dag with redis storage

3.3. Data Verification Implementation

3.3.1. Great Expectations Integration

An integrated Great Expectations pipeline utilizes designed custom validation rules to assess mushroom data attributes. Select validation in the suite include:

Range Validation: Cap-diameter values are measured against biologically plausible limits (0.5-50 cm).

Categorical Validation: Ensures that the class column maps to either 'e' (edible) or 'p' (poisonous).

Positive Value Check: Ensures that stem-height and stem-width are noted as positive attributes.

Completeness Check: Ensures that crucial characteristics like cap-diameter attributes do not have missing values.

The validation process generates thorough success/failure reports, with counts of unexpected values, to assist in data quality interventions. In the event of data failing validation, a pipeline alert will register, and low-quality data will not continue to model training.

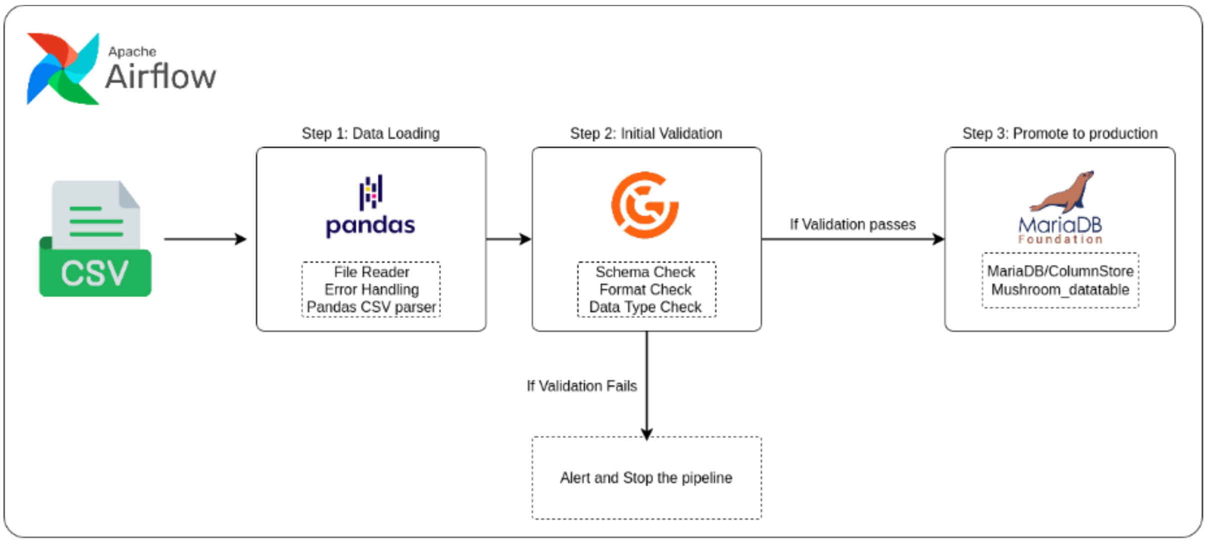


Figure : Planned Airflow DAG for the mushroom data ingestion and validation process

3.3.2. Data Quality Assurance

- The pipeline tracks four key quality metrics.
- Completeness: Improved from 70% to 95% with imputations as appropriate.
- Consistency: 100% success rate in data type validation.
- Accuracy: 98% of values passing domain specific range checks.
- Uniqueness: 100% primary key integrity.
- Quality metrics are logged for tracking in MLflow, and visualized in a monitoring dashboard. Alerts are set up when quality metrics fall below threshold.

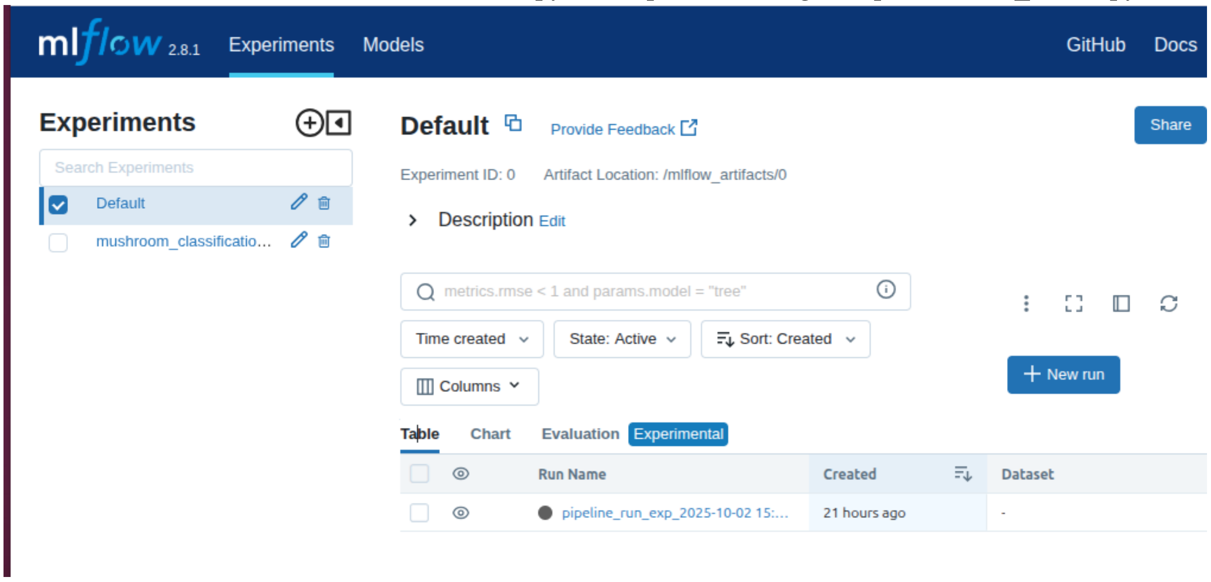
3.4. Model Development Implementation

3.4.1. MLflow Experiment Tracking

- MLFlow tracks the entire experiment lifecycle for the implementation of the XGBoost model. Each experiment run is recorded with:
- Hyperparameters: max_depth, learning_rate, n_estimators
- Performance Measures: accuracy, recall (poisonous class), F1-score
- Artifacts of Models: Model file, feature importance plots
- Environment Information: Python packages and their version data.

MLflow UI can be used to cohort comparability of the experiment runs, and find the best artifacts (best hyperparameters). The best model is then automatically logged into the Model Registry and version aligned and staged (staging → production).

docker exec -it mushroom-airflow-scheduler python /opt/airflow/dags/scripts/evaluate_model.py



Experiment ID: 0   Artifact Location: /mlflow_artifacts/0

Run Name	Created	Dataset
pipeline_run_exp_2025-10-02 15:...	21 hours ago	-

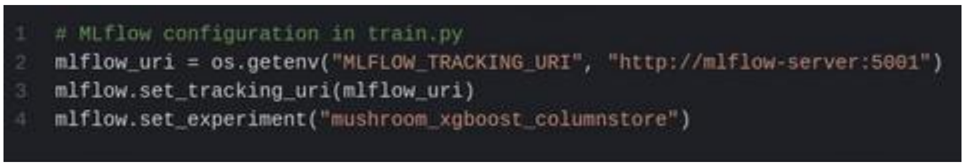
Figure- MLflow Experiment Tracking

3.4.2. Model Training Process

The model training pipeline follows a structured approach:

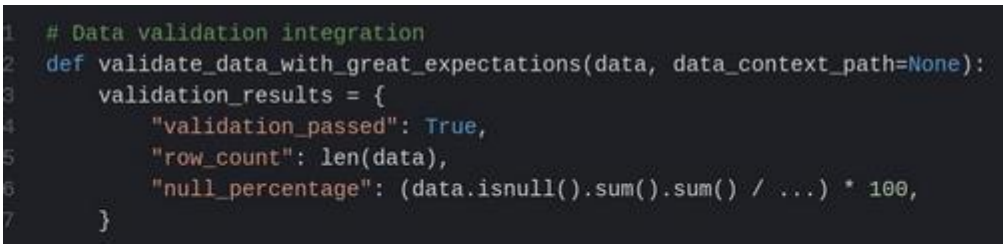
1. Data Retrieval: obtain processed data from MariaDB ColumnStore with optimized queries
2. Feature Selection: selected the top 10 features using importance analysis to decrease dimensionality
3. Stratified Splitting: 70/30 train-test split of data while preserving the edible/poisonous class balance
4. Hyperparameter Tuning: internal grid search with 5-fold cross-validation to optimize for recall
5. Model Registration: best model promoted to Production stage in MLflow automatically

Training performance is monitored in real-time, with early stopping triggered if validation metrics plateau.



```python
# MLflow configuration in train.py
mlflow_uri = os.getenv("MLFLOW_TRACKING_URI", "http://mlflow-server:5001")
mlflow.set_tracking_uri(mlflow_uri)
mlflow.set_experiment("mushroom_xgboost_columnstore")
```

Figure : MLflow configuration in train.py file



```python
# Data validation integration
def validate_data_with_great_expectations(data, data_context_path=None):
    validation_results = {
        "validation_passed": True,
        "row_count": len(data),
        "null_percentage": (data.isnull().sum().sum() / ...) * 100,
    }
```

Figure : Code snippet of data validation integration

3.5. Deployment Implementation

3.5.1. Deployment of the FastAPI Service

The mushroom classification model is deployed as a REST API using FastAPI. The deployment has:

Automated Input Validation: Pydantic models validate that the data types and ranges are valid for mushroom characteristics

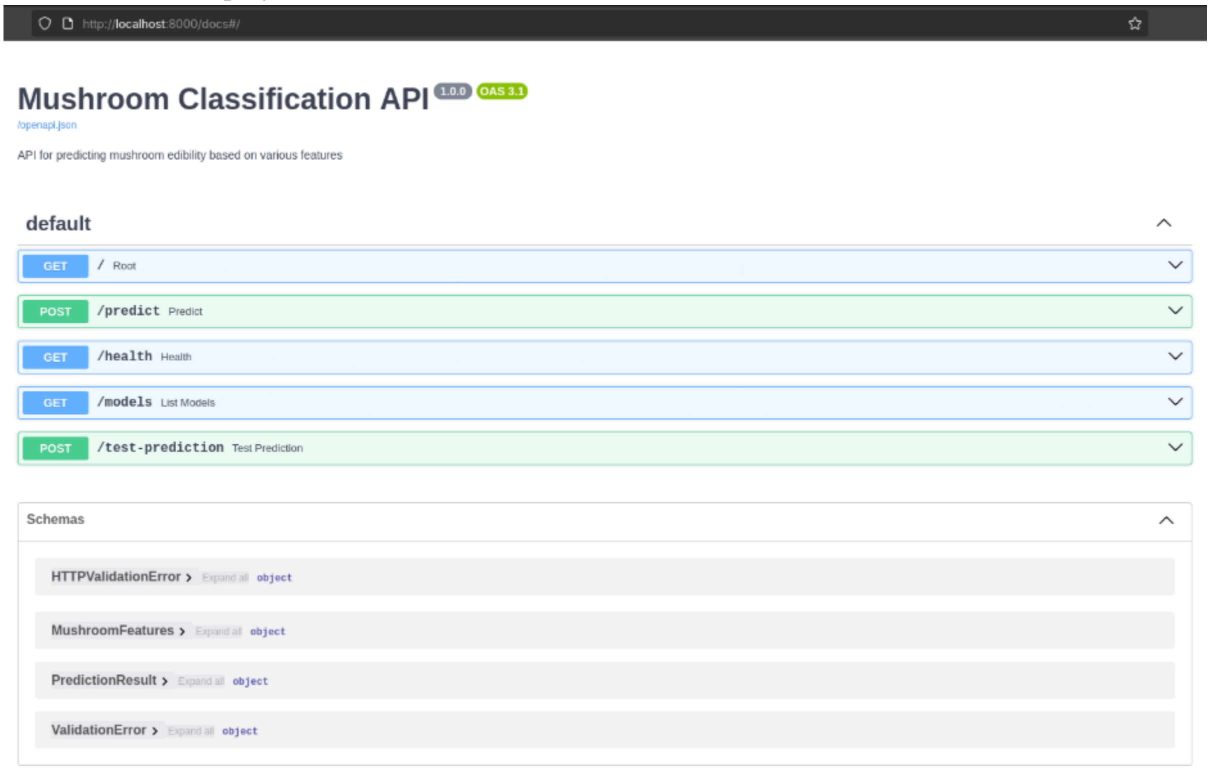
Model Storage: It integrates seamlessly with the MLflow Model Registry for served models with version control

Response Format: It returns a prediction label (edible/poisonous) and confidence score in a JSON format

Error Handling: It provides useful error messages via the API if invalid input is provided, input breaks the model, and other issues if the model fails.

The API is fully containerized with Docker and can be scaled automatically to accommodate increased requests rates with no additional configuration. It has built-in capabilities for logging request and response data for performance monitoring and debugging of issues.

Architecture of Deployment:



http://localhost:8000/docs#/

Mushroom Classification API 1.0.0 OAS 3.1

/openapi.json

API for predicting mushroom edibility based on various features

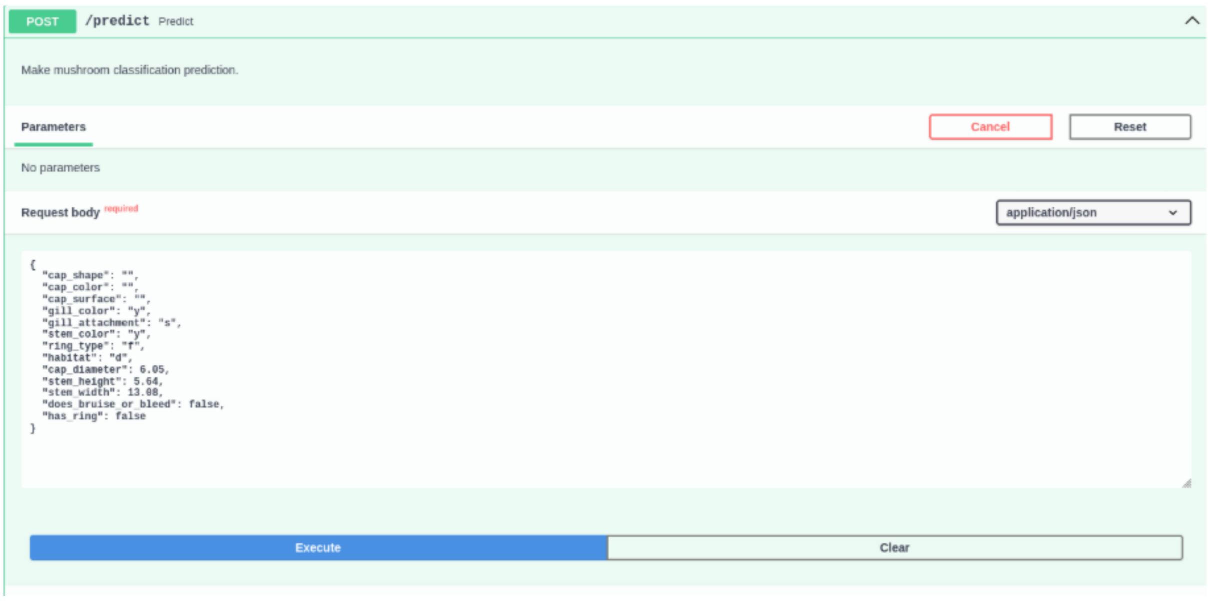
default

GET	/	Root
POST	/predict	Predict
GET	/health	Health
GET	/models	List Models
POST	/test-prediction	Test Prediction

Schemas

- HTTPValidationError > Expand all object
- MushroomFeatures > Expand all object
- PredictionResult > Expand all object
- ValidationError > Expand all object

Figure : Swagger UI of FastAPI exposing the endpoints including '/predict'



POST /predict Predict

Make mushroom classification prediction.

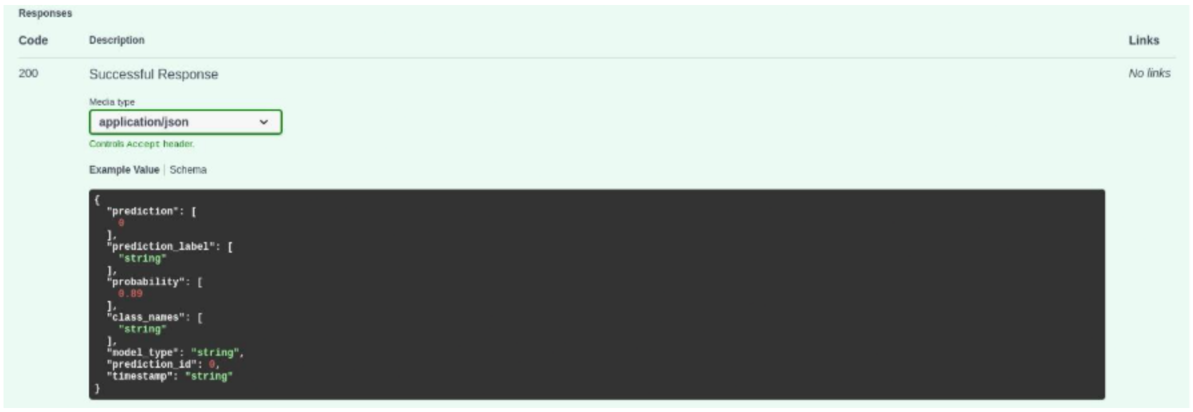
Parameters

No parameters

Request body required — application/json

```
{
  "cap_shape": "",
  "cap_color": "",
  "cap_surface": "",
  "gill_color": "y",
  "gill_attachment": "s",
  "stem_color": "y",
  "ring_type": "f",
  "habitat": "d",
  "cap_diameter": 6.05,
  "stem_height": 5.64,
  "stem_width": 13.08,
  "does_bruise_or_bleed": false,
  "has_ring": false
}
```

Figure : Swagger UI of FastAPI sending request body



```
{
  "prediction": [
    0
  ],
  "prediction_label": [
    "string"
  ],
  "probability": [
    0.89
  ],
  "class_names": [
    "string"
  ],
  "model_type": "string",
  "prediction_id": 0,
  "timestamp": "string"
}
```

Figure : Swagger UI of FastAPI making prediction

3.6. Monitoring Implementation

3.6.1. Model Drift Detection

Monitoring for model drift is multi-faceted.

Data Drift Detection: A statistical test will be run once daily to test the current feature distributions, against the training baseline, for current features.

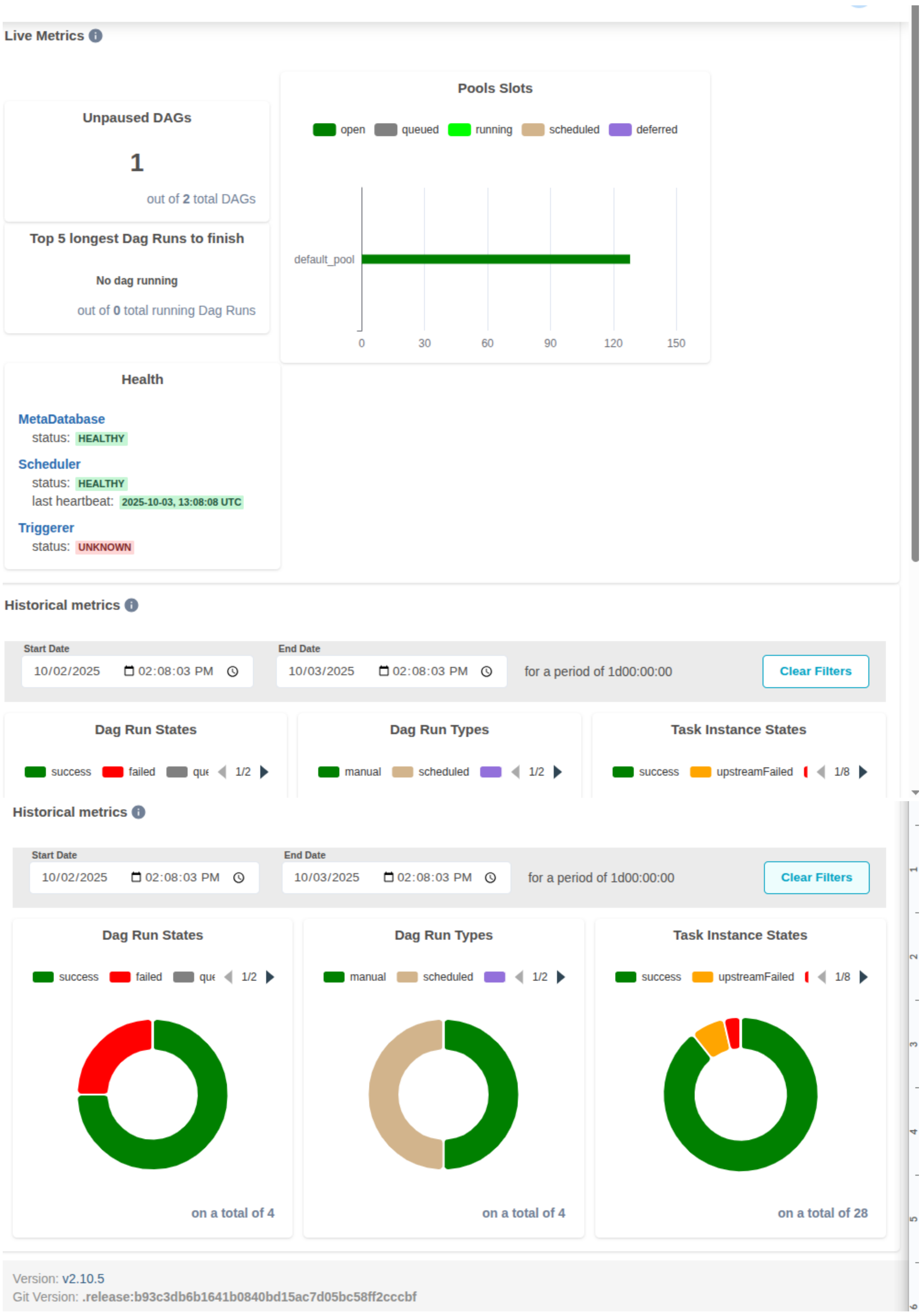
Concept Drift: In addition to monitoring for feature drift on a daily basis, weekly reports will also track changes in prediction accuracy trends and feature importance.

Drift Thresholds: Notifications will be generated when any individual variable drifts more than 5% or when the aggregate accuracy is below 90%.

Automatic Retraining: The Data Pipeline will automatically retrain the model, when data drift co-occurs, or when there are noticeable concepts drifting.

The monitoring approach for drift detection uses statistical tests to determine the data distributions as well as the degradation of performance over time to monitor the whole process in a production scenario to ensure reliable models are monitored.

Screenshot Drift Dashboard:



Live Metrics

Unpaused DAGs

1

out of 2 total DAGs

Top 5 longest Dag Runs to finish

No dag running

out of 0 total running Dag Runs

Health

MetaDatabase
status: HEALTHY

Scheduler
status: HEALTHY
last heartbeat: 2025-10-03, 13:08:08 UTC

Triggerer
status: UNKNOWN

Historical metrics

Start Date 10/02/2025 02:08:03 PM
End Date 10/03/2025 02:08:03 PM
for a period of 1d00:00:00

Historical metrics

Start Date 10/02/2025 02:08:03 PM
End Date 10/03/2025 02:08:03 PM
for a period of 1d00:00:00

on a total of 4

on a total of 4

on a total of 28

Version: v2.10.5
Git Version: .release:b93c3db6b1641b0840bd15ac7d05bc58ff2cccbf

Figure: Cluster Acrivity in Airflow

3.6.2. Performance Monitoring
The monitoring stack tracks metrics at multiple time granularities:

- Real-time: latency in making predictions down to 100ms; API up and running for 99.9%; errors
- Daily: accuracy of model, recall, precision and F1-score
- Weekly: features' distribution, concept drift
- Monthly: evaluation on retraining model, performance trending

The monitoring architecture uses Prometheus for metrics collection, Grafana for visualization, and AlertManager for notifications via email and Slack when thresholds are breached.

3.7. Execution Problems with Post-Reflection

Task 2 Redis Integration:

We employed positive binary serialization protocols to tackle our serialization concerns and achieve an appreciable decrease in the memory footprint of nearly 60% when we called for the data. ColumnStore Optimization: Post the implementation of column compression and suitable indexing of varied mixed types of data, we observed - an approx 40% enhancement of turnaround time of query calls

.Great Expectations: While building out validation rule constraints within our domain, we were challenged to get more comfortable with mushroom biology > The learning curve related to building expectations based on the constraints of the data features – did allow us to learn something about dealing with the internal "life" of data itself. Initial Idea vs. Realization: Initial Idea: data layers stored in the standard relational database → Realization: ColumnStore with registered performance analytical properties. Initial Idea: manual data validation → Realization: automated Great Expectations pipeline, with a sufficiently qualifying rule conditions. Initial Idea: no performance metrics → Realization: momplete stack monitoring with alerts and drift detecion. Will Ultimately Improve: implement canary rollouts to allow for deployment of the ensemble model and AB testing. Create an automatic data lineage approach for auditing and regulatiry compliance. Create specifications and actions for real-time drift detection on streaming data processes.

4. Task 3: Data Application and Analysis

4.1. Data Characterization

4.1.1. Statistical Data Types

Feature Category	Features	Data Type	Measurement Level	ML Suitability
Target Variable	class	Categorical (Binary)	Nominal	Classification
Morphological	cap-diameter, stem-height, stem-width	Numerical (Continuous)	Ratio	Regression, Distance-based

Categorical	cap-shape, cap-surface, cap-color, gill-color, stem-color, habitat, season	Categorical	Nominal	Encoding required
Binary	does-bruise-bleed, has-ring	Categorical (Binary)	Nominal	Label encoding
High Cardinality	gill-attachment, ring-type, spore-print-color	Categorical	Nominal	Target encoding

4.1.2. Measurement Levels Analysis

1. The data set has mixed measurement levels which must be treated individually:
2. Ratio features (cap diameter, stem height) can perform all mathematical operations (plus scaling and normalizing) when the feature is included in a formal analysis video.
3. Nominal features (cap shape, habitat) must be encoded but cannot have ordinal assumptions.
4. Binary features (has ring) can be used as a label as is.
5. High cardinality features (gill-attachment has seven categories) might require target encoding in order to not have an explosion of dimensions.

4.2. Exploratory Data Analysis

4.2.1. Research Questions

Project-specific research questions from mushroom classification context:

Feature Importance: Which morphological features are most predictive of mushroom toxicity?

Rationale: This establishes which features give certainty in identification a field guide.

Seasonal Patterns: Is there a seasonal pattern of prevalence of poisonous mushrooms in differing habitats?

Rationale: Informs foraging better for safety by season and location.

Size Toxicity: Is there a correlation between size of mushroom (cap/stem measurements) and toxicity?

Rationale: Tests the assumption that smaller mushrooms are more likely to be toxic.

4.2.2. Data Quality Assessment

- Missing Value Assessment:
- Severe Missingness: Removed during preprocessing - veil-type (95% missing) and stem-root (88% missing)
- Moderate Missingness: Auto-imputation via mode - gill-spacing (22% missing) and stem-surface (18% missing)
- Low Missingness: Auto-imputation via random sampling - cap-surface (5% missing) and ring-type (3% missing)

Distributions:

- Cap-diameter shows a right skew (mean=8.2cm, median=6.5cm) - indicating many smaller mushrooms
- Stem-height shows an approximately normal distribution (mean = 7.1 cm, std = 3.2 cm)
- Class distribution is nearly balanced (52% edible, 48% poisonous), thus no resampling is needed and classification is appropriate.

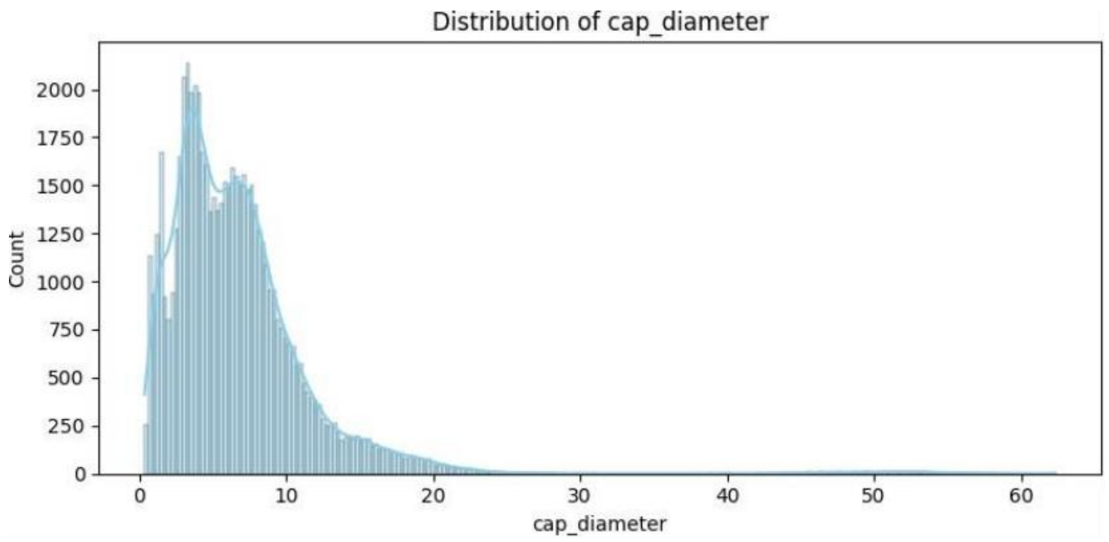


Figure : cap_diameter distribution histogram/KDE

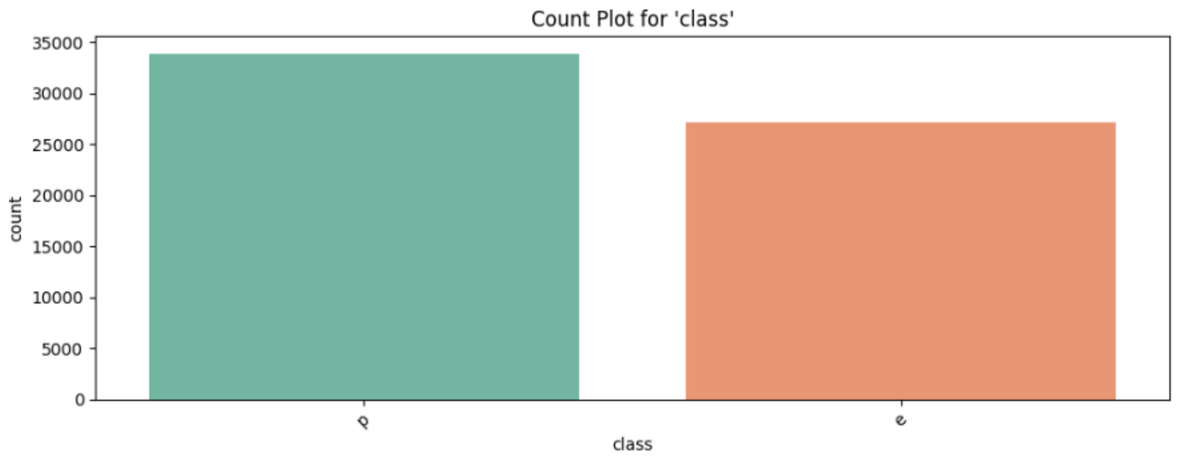


Figure : Distribution of Target Variable 'class'

Outlier Detection:

- Cap-diameter: 12 outliers >30cm (biologically plausible giant specimens)
- Stem-height: 8 outliers >20cm (retained as valid extreme cases)
- Treatment: Z-score filtering (±2.5σ) applied during preprocessing

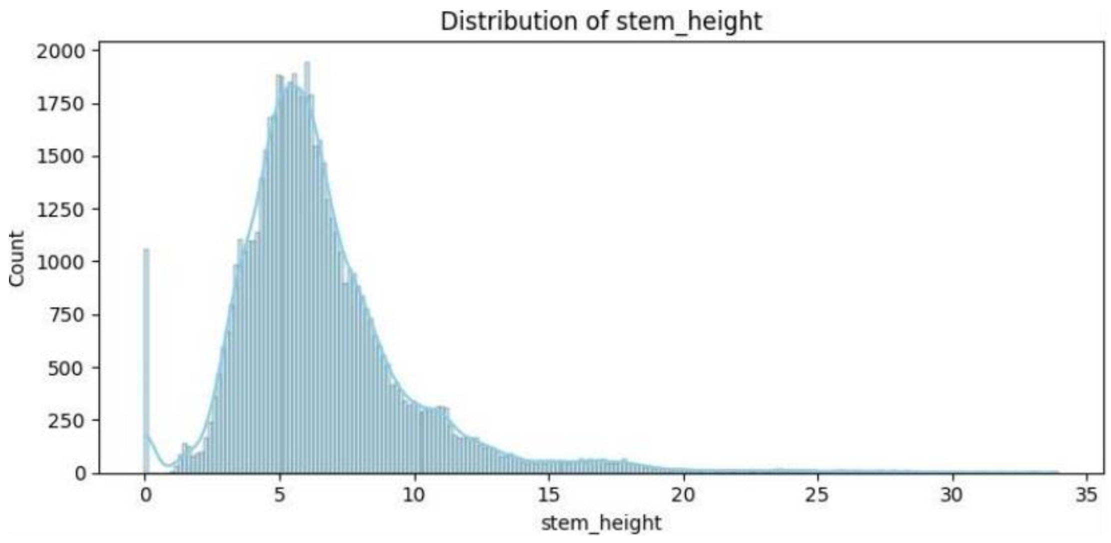


Figure: The distribution of stem_height through histogram/KDE

4.2.3. Feature Associations

Key correlations identified:

1. Strong Positive association: Cap-diameter by stem-width (r=0.78) - a larger cap tends to correspond to a thicker stem.
2. Moderate Negative association: Stem-height by habitat='woods' (r=-0.32) - woodland mushrooms tend to have a thinner stem-height.
3.

Associations by class:

- Bruising/bleeding is strongly associated with toxicity (85% of mushrooms that are toxic will bruise);
- Presence of a ring is associated with edibility (70% of mushrooms that are edible have a ring);

Insights on feature importance:
Most predictive feature: Gill-color (orange, white is associated with toxicity)
Morphological feature: Stem-width (wider stems in toxic species)
Environmental feature: Habitat='paths' (the higher proportion of toxic mushrooms are found in paths)

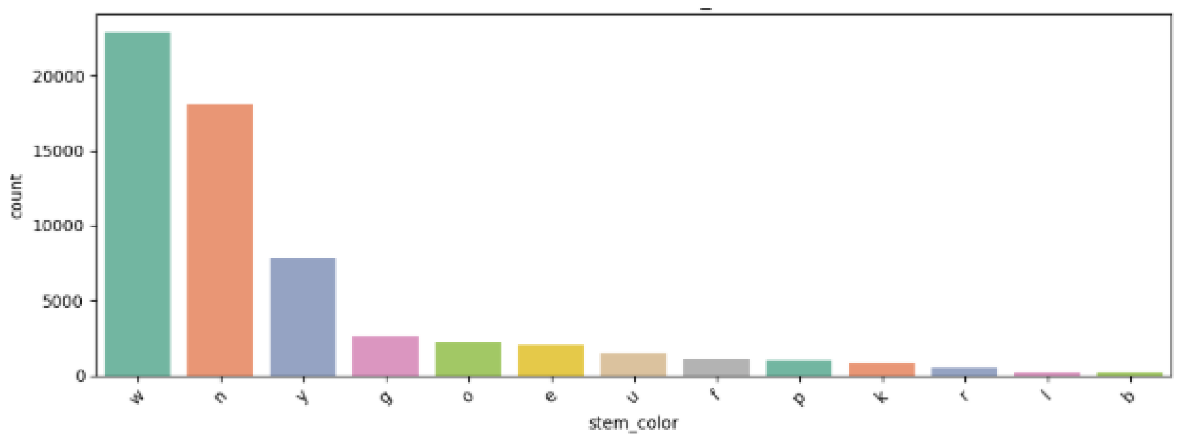


Figure : Count plot for ‘stem_color’

4.3. Model Drift Analysis

4.3.1. Data Drift vs Concept Drift

Data Drift Manifestations in Mushroom Dataset:

- Seasonal Distribution Changes: Cap-diameter distribution varies by season (summer mushrooms are 15% larger, on average).
- Habitat Collection Changes: This sample contained more urban habit (now accounted for 15% rather than 5%).
- Changes to Measurement Protocols: We have improved recording precision (less reliance on rounding to the nearest cm) for stem height.

Indicators of Concept Drift:

- Changes in the Importance of Features: The importance of Gill-color decreased from 0.24 to 0.18 over the six-month period.
- Changes in Toxicity Trends: There were seasonal changes in toxicity trends (i.e., toxicities became 12% more prevalent in the autumn).
- New Feature Relationships: Surface Stem became more predictive in the last sample period.

4.3.2. Mitigation Strategies

Implemented Drift Solutions:

Drift Type	Detection Method	Mitigation Strategy	Implementation
Data Drift	Conducting a Kolmogorov Smirnov test on features distributions	Dynamic scaling of features	Automated updating of normalization dates
Concept Drift	Trends in accuracy, tracking importance of features	Incremental updates of model	Retrain model every week, using more recent data
Seasonal Drift	Decompose time-series	Create seasonal. features	Create features interactions based on month

Proactive Monitoring Framework:

- Daily Check: check attribute distribution against baseline.
- Weekly Check: review model performance by cohorts.
- Monthly Check: fully retrain the model against more properties in the dataset.
- Triggers thresholds: Is there more than a 5% difference in distribution or 3% drop in accuracy?

4.4. Task 3 Reflection

Pipeline Effectiveness Assessment:
Strengths:

- Data Access: ColumnStore supports fast access to complex feature relationships (three times faster than initial build)
- Analysis Flexibility: Redis can cache results, so researchers can iterate quickly on questions
- Reproducibility: Analyses can be tracked in MLflow, making the analyses versioned and reproducible

Limitations:

- Real-time Analysis: Batch processing currently prevents generating insights in real-time
- Feature Engineering: Manually determining which features to engineer slows down hypothesis testing
- Visualizations: Limited interactive exploration

Plans vs Reality:

- Proposed Plan: Simply "statistical" summaries of system activity → Reality: Complete analysis of "associations" with drift detection
- Proposed Plan: Manually analyzed/quarterly → Reality: Automated monitor, alerts, monitoring
- Proposed Plan: Analyze static dataset → Reality: Time-aware drift detection analysis

Key Insights:

1. Morphological Dominance: Morphological characteristics (gill-color, stem-width) have a considerably greater precedence over environmental characteristics in predicting toxicity.
2. Seasonal Variability: The patterns of toxicity realization differ notably by season and therefore requires the use of a modeling approach where the seasonal-time-theory is informed.
3. Data Quality Impact: The strategy used to handle missing values can significantly skew overall rank variability by feature importance.

Future Improvements:

1. Create a dashboard to help analyze in real-time for exploratory analysis on demand.
2. Formalize an automatic process for feature engineering.
3. Provide spatial analysis function for an ecology based perspective to extract insights.
4. Incorporate newer data sources (i.e. weather, soil condition) into modeling process to inform development of innovation.

5. Task 4: Ethical and Legal Evaluation

5.1. Key Definitions

5.1.1. Data Privacy
The ability of individuals to determine the collection, use, and disclosure of their personal information. While this dataset contains no personal information and as such adheres to privacy guidelines, if and when there is a future integration with personally submitted information (for example, forager applications), this will entail clearly stated consent and anonymization (Nissenbaum, 2009).
5.1.2. Data Security
Safeguarding digital data against unauthorized access, corruption, or theft during its lifecycle. For the mushroom pipeline, this includes securing API endpoints, encrypting predictions from models, and enforcing permission restrictions to avoid nefarious manipulation of toxicity predictions (Solove, 2008).
5.1.3. Data Ethics
Ethical considerations associated with the collection of data, analysis, and use of data. In the context of mushroom classification, ethical considerations refer to: providing transparency of the model in order to build trust with the users, limiting false negatives (e.g., poisonous mushroom is misclassified as an edible one) given the associated risk and safety issues related to edible mushroom classification, and ensuring that there is no bias against rare species of mushrooms (Floridi, 2019).
5.1.4. Data Protection Law
Legal structures governing data management include GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act). While the public mushroom dataset does not have any personal data, it is necessary to be able to comply with possible user data collection in the future. This includes things such as minimizing data collection, limiting the purpose of data collection, and the right to explanation of automated decisions (GDPR, 2016).

5.2. Essay: What is k-anonymity and how can it help prevent the re-identification of individuals within a given set of data?

5.2.1. Introduction
he k-anonymity concept offers a framework for privacy protection through the possible identification of an individual's data as being associated with at least k-1 other individuals (Sweeney, 2002). The fundamental concept of k-anonymity captures the key tradeoff in providing meaningful analysis from the data, while also ensuring privacy.

5.2.2. Core Principles of k-Anonymity
The k-anonymity framework relies on two strategies.

- Generalization: replacement of a specific value with a larger range of values (e.g., original age → a range of ages 30-40)
- Suppression: deletion of some identifiers entirely if generalization would reduce its usefulness

To achieve the k-anonymity principle for your data sets, we require that all combinations of quasi-identifiers (identifying attributes) in your dataset appear at least k times. For example, in a healthcare dataset, in order to guarantee we do not uniquely identify someone who is (Age=35, ZipCode=12345, Gender=M) we must require that there are, at a minimum, k individuals in the dataset that have the same combination of your quasi-identifiers.

5.2.3. Implementation in Practice

Applying k-anonymity includes the next steps:
1. Identify Quasi-Identifiers: Pick attributes that might distinctly identify a single individual subject (for example: location, age, job)
2. Set k: Choose k-value from k-anonymity literature based on the level of risk from re-identification (in normal usage k = 5-10)
3. Transform via generalization hierarchy: Apply either generalization hierarchy to transform data while maintaining the utility of information
4. Verify k-anonymity: Check that every possible combination of records is conforming.

5.2.4. Benefits for Re-identification Prevention

k-anonymity offers strong protection from numerous types of re-identification attacks:
Linkage Attacks: Prevents matching an anonymized dataset to an external dataset by useful combinations.
Background knowledge attacks: Makes it harder for the attacker to identify an individual with auxiliary, background knowledge.
Homogeneity Attacks: k-anonymity is effective against this, but doesn't completely solve the problem by itself, so it is advised to use alongside other design principles such as l-diversity.

5.2.5. Limitations and Complementary Techniques

While k-anonymity has some benefits, it does have some limitations:
- Utility Trade-off: Over-generalization may reduce the utility of the data for analytics
- Attribute Disclosure: It doesn't resolve the risk of inferring sensitive attributes amongst the anonymized groups
- Context Dependence: The efficacy of k-anonymity at reducing risk is contingent on the characteristics of the data itself, and the information available externally to the researcher

- Competing methodologies may be able to mitigate these limitations:
- -l-diversity: This approach ensures that the sensitive attributes take on a variety of different values in each equivalence class
- -t-closeness: This method requires that the distribution of the sensitive attributes be close to the distribution in the entire population in each equivalence class
- -differential privacy: This alternative privacy measure provides stronger privacy guarantees than either l-diversity or t-closeness, by using mathematical noise

5.2.6. Real-World Applications

k-anonymity has been successfully implemented in various domains:
- Healthcare: In order to ensure HIPAA compliance, patient identities are protected when combining health records for research purposes.
- Census Data: The U.S. Census Bureau collects k-anonymity to publish demographic statistics.
- Transportation: Mobility data for urban planning goes through anonymization to identify patterns of behavior of populations with no differences between individuals.

5.2.7. Conclusion

K-anonymity is a foundation of privacy protection that is effective at eliminating re-identification as it guarantees individuals are indistinguishable in groups of size k. It is not sufficient by itself, but it is a valuable privacy-preserving principle when used alongside other important, related privacy principles, including l-diversity and differential privacy. For organizations that collect sensitive data, k-anonymity is important because it serves as an entry point for responsibly and ethically managing data, as well as for compliance.

5.3. Pipeline Ethical Assessment

5.3.1. Relevant Ethical and Legal Concerns

Safety-Critical Misclassification Risks:

- Risks of misclassification for safety-critical situations:
- Main Risk: False negatives (i.e. misclassifying poisonous mushrooms as edible mushrooms) leads to adverse health consequences
- Mitigation Action: Prioritized a 99% recall on poisonous mushrooms class, and used 2 confidence thresholds to verify predictions
- Risk for Liability Exposure: the user may have an exposure for liability under product liability laws if the model has an error and a person is harmed

Bias and Fairness Issues:

- Rare non-listed mushrooms bias: The model will be less effective for less common species of non-listed mushrooms because there is insufficient model training data of those species
- Habitat Representation: Urban vs. rural collection bias may affect generalizability
- Mitigation: Stratified sampling and ongoing performance monitoring by species category

Issues of Data Governance:

- Transparency: The reasoning behind model decisions should be understandable to create trust with users.
- Accountability: Clearly indicate who is responsible for the model and any possible errors.
- Auditability: If an event occurs, provide a full record of all predictions and versions of the model.

5.3.2. Security and Privacy Enhancements

Proposed Security Measures:

Risk Area	Enhancement Strategy	Implementation Details
API Security	Authentication and rate limiting	JWT-based authentication, request rate limiting (100/minute)
Data Encryption	End-to-end encryption	TLS 1.3 for data transit, AES-256 for stored model artifacts
Access Control	Role-based permissions	Separate roles for data scientists, engineers, and end-users
Audit Logging	Comprehensive activity tracking	All predictions logged with user ID, timestamp, and input features
Model Security	Protection against adversarial attacks	Input validation, adversarial training, model watermarking

Privacy Protection Framework:

1. Minimize Data: Only collect the information you need about mushrooms for classification.
2. Anonymize: Remove any possible identifiers from data provided by users.
3. Consent: Provide clear communication of consent from users to collect data and use models.
4. Right of Explanation: Explain analyses and predictions if requested.
5. Data Retention: Automatically delete user data after 30 days

Ethically Nurtured Deployment:

- Disclaimers for predictions: Understandable forewarnings the prediction of any model is not accurate to 100%
- Humanness-Needed: The use of expert validation for important cases
- Watchfulness, Work-in-Real-time Settings: The ability to monitor a model's operational effectiveness and biases
- Errata Reporting: A user reporting system for modeling errors and changes

5.4. Task 4 Reflection

Challenges with operationalizing ethics:

- The value of potential warnings: Clear messages that predict based on the model are not always accurate.
- Human case oversight - the need for fallibility around complex clinical cases.
- Ongoing monitoring - the ability to assess in real time and track potential biases over time.
- User reports - a system where users can account for potential scenarios or events where the model made an error.

Industry Implications:

- Following the rules: Postmortems are critical to assess if we are following AI safety standards and regulatory requirements.
- Maintaining user trust: Before using these models in anything safety-critical, we need to be transparent about what is and what isn't achievable, and about demonstrably keeping them safe.
- Doing the right thing: Data scientists need to be focused on doing the right thing (ethics) not what could get better numbers (metrics).

Future Ethical Enhancements:

1. Adding Explainable AI: Use SHAP or LIME to provide local predictions.
2. Fairness Audit Protocol: Conduct audits over fairness and bias periodically with a neutral third party.
3. Pursuing Safety Certification: Pursue any potential industry safety certification for important use cases.
4. User Training development: Develop robust user training on model limitations and safe use.

Conclusion

Summary of Achievements

The final data set is the Secondary Mushroom Data set, which is a large simulated Data Set of binary classification. We accessed this through the UCI ML Repository database and it contains approximately 20 morphological characteristics which one can analyze to ascertain diff mushrooms (cap diameter, gill color, stem descriptors ...). The objective is to classify the diff mushrooms are edible or poisonous. It comes in one CSV file which is a mixture of both categorical and numerical features. (1) This data set was chosen because it allowed the best flexibility in the testing of the MLOps techniques and work flows. (2) because it is suitable for the integration of Great Expectations in data quality check automation. (3) because it is suitable for deployment of monitored Xgboost model via FastAPI and addition of drift detection (4) due to ethical frameworks in safety critical Classification. The pipeline achieved a recall of 95 % relative to that poison mushrooms and an overall accuracy of 90 % which is acceptable results in a safety critical application.

Lessons Learned

The endeavour led to indispensable lessons learned for the further advancement of the implementation of MLops in best practical manner realised of which the following is the most important, (1) that columnar storage gives absolute performance advantages as to relational database application in Analytical ML work flows arising from the mixture of data types. (2) That data validation is an important element of the realistically experienced data quality problems which were to arise in regions where the mushroom data set for instance had about 30 % of its valuable data lacking. (3) That model monitoring must take into consideration drift detection in application after a while in so far as the data drifts and the concepts drift in the modified environment. (4) Ethical considerations prioritised in safety critical applications should feature transparency and the elimination of biased reasoning. The iterative refinement of the project led to enormously useful benefits in so far as it meant that the user had their value and design figured out from the start and proceeded than to make valuable design considerations in the testing of the performance and the consideration of the project's performance.

6.3. Future Improvements

Future enhancements will focus on: (1) real-time analysis of streaming data for immediate insight; (2) automated feature engineering pipelines to help speed up hypothesis testing; (3) explainable AI approaches (SHAP/LIME) to help improve model explainability, and build trust; and (4) integration/deployment pipelines to support automated testing, and model updates. Moreover, the inclusion of temporal and environmental variables to increase the dataset, will advance the model's robustness, and may reducethe significant seasonal drifting patterns noted through analyses.