Data Management for Machine Learning (DMML) Assignment

1. Introduction

This project focuses on implementing a **data management pipeline** for a **bank churn prediction** model. The pipeline covers **data ingestion, validation, preprocessing, feature storage, model building, and orchestration** using **DVC and Airflow**.

Objectives

- Automate the data pipeline for efficient handling of raw and processed data.
- Ensure data quality through validation checks.
- Store and retrieve features efficiently using PostgreSQL.
- Automate the pipeline using Apache Airflow.
- Implement version control using DVC to track dataset and model changes.

2. Pipeline Overview

The workflow consists of the following stages:

1. **Data Ingestion** – Fetch data from **Kaggle** and an external **API**, saving it locally.
2. **Raw Data Storage** – Store data in an **AWS S3 bucket** with timestamp-based versioning.
3. **Data Validation** – Perform checks for missing values, duplicates, and data types.
4. **Data Preparation & Transformation** – Clean and transform data for model training.
5. **Feature Store** – Store selected features in a **PostgreSQL database** for structured access.
6. **DVC for Versioning** – Track dataset, features, and model changes.
7. **Model Building** – Train and evaluate multiple machine learning models.
8. **Orchestration & Automation** – Automate the pipeline using Apache Airflow.

---

3. Data Ingestion

Sources of Data

- **Kaggle dataset**: The dataset containing historical customer churn data is fetched using the Kaggle API.
- **API endpoint**: An external API provides fresh data for inference.

Implementation Details

- The data ingestion script (`1_Data_Ingestion.py`) automates downloading from both sources.
- Data is saved locally in the format:

```
data/raw/bank_churn_YYYYMMDD_HHMMSS.csv
```

- Logs are maintained to track data downloads.

Challenges

- **Kaggle API authentication issues** → Resolved by configuring the `kaggle.json` credentials file correctly.
- **Handling API rate limits** → Implemented retry logic with exponential backoff.

---

4. Raw Data Storage

Storage Location

- AWS S3 bucket: `dmml-bank-churn-data`
- Directory structure:
  - `raw_data/` – Stores original datasets.
  - `reports/` – Stores validation reports.

Versioning Strategy

- Each uploaded file is named with a **timestamp** to maintain a history of changes.
- The S3 upload script ensures older versions are retained for reproducibility.

Challenges

- **S3 permission errors** → Updated bucket policies and IAM role permissions.
- **Network failures during upload** → Implemented retries with progressive backoff.
- **IAM policy updates affecting access** → Regular monitoring and policy adjustments were necessary.

---

5. Data Validation

Validation Checks

- **Missing values**: Identifies columns with NULL values.
- **Data type mismatches**: Ensures column data types match expectations.
- **Duplicate records**: Checks for redundant entries.
- **Outlier detection**: Identifies anomalies in numerical fields.

Report Generation

- Validation results are logged and stored in `reports/`.
- Summary statistics and visualizations (histograms, box plots) are generated.

Screenshot Placeholder: `![Validation Report](path/to/validation_report.png)`

---

6. Data Preparation & Transformation

Feature Engineering

- **BalancePerProduct**: Computed as `Balance / NumOfProducts`.
- **Geography Encoding**: One-hot encoding for `Geography`.

Data Cleaning

- Drop unnecessary columns.
- Handle missing values through imputation or removal.

Storage Format

- Processed data is saved in S3 under `processed_data/`.
- Transformed data is stored in `transformed_data/`.

Challenges

- **Handling categorical variables** → Applied one-hot encoding.
- **Detecting incorrect values** → Applied statistical validation.
- **Data inconsistency in API vs training data** → Applied pre-processing rules to standardize formats.

---

7. Feature Store

Database Setup

- PostgreSQL is used to store feature values.

Table Schema

```sql
CREATE TABLE feature_values (
    id SERIAL PRIMARY KEY,
    CreditScore FLOAT,
    Age FLOAT,
    Tenure INT,
    Balance FLOAT,
    NumOfProducts INT,
    IsActiveMember INT,
    Geography_France BOOLEAN,
    Geography_Germany BOOLEAN,
    Geography_Spain BOOLEAN,
    BalancePerProduct FLOAT,
    Exited INT,  -- Nullable for API data
    data_source VARCHAR(10),  -- 'train' or 'api'
    version TIMESTAMP  -- Timestamp-based versioning
);
```

Screenshot Placeholder: `![Feature Store Schema](path/to/feature_store.png)`

Challenges

- **Database connection refused** → Resolved by configuring PostgreSQL to accept external connections.
- **Efficient querying** → Indexed key columns for optimized retrieval.

---

11. Conclusion

This project successfully implemented an end-to-end **data management pipeline** for **bank churn prediction**, ensuring **efficient data ingestion, validation, transformation, feature storage, model building, and orchestration**. The integration of **DVC** enabled version control, while **Airflow** automated workflow execution.

Key Takeaways

- **Scalability**: The pipeline supports new data ingestion and retraining without manual intervention.
- **Reproducibility**: Using DVC and structured storage ensured that past versions of data and models could be retrieved.
- **Automation**: Airflow orchestrated all steps, minimizing manual workload and errors.
- **Performance Optimization**: Feature selection and model comparison ensured the best model was chosen efficiently.

Future Enhancements

- **Real-time Feature Store**: Implement a real-time feature store for live inference.
- **Advanced Model Monitoring**: Deploy model monitoring to detect drift in predictions.
- **Deployment**: Integrate the trained model into a web API for real-time customer predictions.

This assignment laid a strong foundation for handling machine learning workflows in a **structured, versioned, and automated** manner, essential for real-world deployments.

Uploads In the zip file, each folder will contain respective python script, a small document about the code, necessary output screenshots.