# Assignment Requirements

## 1. Data Ingestion & Versioning

1. **Data Source & Storage**
   - Assume raw data about customer activity is continuously uploaded to an object store (e.g., S3, GCS).
   - Implement or outline a data ingestion pipeline that pulls new data **hourly**.
   - The pipeline should handle potential data anomalies (missing columns, unexpected data types) gracefully and either alert or auto-correct.
2. **Data Versioning**
   - Use a data versioning tool (e.g., DVC, Delta Lake, or LakeFS) or implement an equivalent approach to maintain reproducibility.
   - Ensure that **each training run** has a clear record of the data snapshot used.
3. **Documentation**
   - Provide a **README** or a design doc illustrating how the ingestion pipeline works, how data versions are tracked, and how you handle schema changes.

**Deliverables**

- Configuration or scripts (e.g., Databricks, Airflow DAGs, Prefect flows) that show how data is ingested and stored.
- Documentation explaining your design, tools used, and versioning strategy.

---

## 2. Data Processing & Feature Engineering

1. **Transformation Pipeline**
   - Create a robust data transformation or feature engineering pipeline that converts raw data (e.g., logs, clickstream, subscription info) into model-ready features.
   - Include steps such as normalization, missing value imputation, feature selection, and encoding categorical variables.
2. **Orchestration & Reproducibility**
   - Containerize your feature engineering process (e.g., Docker).
   - Show how the container can be run on a local machine as well as in a CI environment or a Kubernetes cluster.
   - Keep track of the exact transformation code versions using Git, along with pinned library versions.
3. **Testing**
   - Include **unit tests** and/or **integration tests** for your transformation pipeline.
   - Provide an automated way (within CI/CD) to run these tests every time code changes.

**Deliverables**

- Scripts or notebooks (e.g., Python scripts, PySpark jobs) and corresponding Dockerfile(s).
- Sample output data (transformed features) in a versioned location (could be a dedicated bucket or folder).

- Automated tests for transformations (in a `tests/` directory or integrated into the pipeline).

---

## 3. Model Training & Experiment Tracking

1. **Initial Model**
   - Train a **baseline model** (e.g., logistic regression, random forest) on your processed dataset.
   - Log metrics (accuracy, F1-score, precision, recall) in an experiment tracking tool (e.g., MLflow, Weights & Biases, Comet).
2. **Experiment Tracking**
   - Show how you track hyperparameters, metrics, and artifacts (e.g., model weights) in a reproducible manner.
   - The solution should enable comparing multiple runs easily.
3. **Model Versioning**
   - Use a registry (e.g., MLflow's Model Registry, S3 versioning, or custom artifact store) to store and version your models.
   - Demonstrate how to **promote** a model from a "development" stage to a "production" stage.
4. **Automated Training**
   - Integrate training into your pipeline (e.g., triggered daily or upon new data availability).
   - Ensure the training job can be invoked via a CI/CD pipeline or job scheduler.

**Deliverables**

- Model training code (Python scripts, notebooks, etc.).
- Experiment tracking logs (via MLflow or similar).
- Model artifact registry configuration with versioned artifacts.
- CI/CD pipeline snippet showing automated training triggers.

---

## 4. Deployment & CI/CD

1. **Containerize the Model**
   - Build a container image for your model serving (e.g., Flask, FastAPI, BentoML).
   - Ensure that the container can scale and is platform-agnostic (runs locally, in Kubernetes, etc.).
2. **Continuous Integration**
   - On each commit to the main branch, run:
     - **Linting & static analysis** (e.g., flake8, black).
     - **Unit tests** and **integration tests** (including some tests against a mock or small dataset).
   - Provide a **badge** or status indicating CI results.
3. **Continuous Delivery (Staging)**
   - Automatically deploy the new model container to a **staging** environment (e.g., a separate Kubernetes namespace or staging cluster) upon successful CI.

    - Run a **smoke test** or simple health check to confirm the service is up.
4. **Production Deployment**
    - Enable manual or automated promotion to **production**. This could involve:
        - A canary release (deploy new version alongside the old one, redirect small traffic, compare performance).
        - A blue/green deployment (deploy new version in parallel, switch traffic if healthy).
5. **Infrastructure as Code (IaC)**
    - Show how you define your deployment infrastructure (Kubernetes cluster, networking, security) using IaC (e.g., Terraform, AWS CloudFormation, Azure Resource Manager, or Pulumi).
    - Must include relevant environment variables, secrets management (e.g., using a secrets manager), resource configurations.

**Deliverables**

- Dockerfile and any scripts/manifests for Kubernetes/ECS or equivalent.
- CI/CD configuration (Jenkinsfile, GitHub Actions workflow, GitLab CI YAML, etc.).
- Terraform/CloudFormation/Pulumi files for spinning up the infrastructure.
- Documentation detailing your deployment strategy and IaC approach.

## 5. Security & Compliance

1. **Access Control & Secrets**
    - Demonstrate how you manage sensitive information (API keys, DB passwords, etc.) using a **secure secrets manager**.
    - Implement **role-based access control (RBAC)** for your CI/CD pipeline and for any cloud resources.
2. **Container Security**
    - Use a container security scanner (e.g., Clair, Trivy) to check for vulnerabilities in your Docker images.
    - Show how the pipeline fails or alerts if critical vulnerabilities are detected.
3. **Data Privacy**
    - Address how you would handle **PII** (personally identifiable information) in logs and model training.
    - At a minimum, design anonymization or tokenization for sensitive fields and log scrubbing to avoid leaking PII.
4. **Documentation**
    - Provide a short section in your README about **compliance** (e.g., GDPR considerations if you're in the EU, HIPAA if relevant to healthcare, etc.).

**Deliverables**

- Secrets management configuration and policy.
- Container security scanning results and explanation of how vulnerabilities are handled.
- Documentation describing data handling and privacy considerations.