**Lab 1 – Analyzers & Metrics Repo**

**Objective:**  
Centralize metric definitions and analysis logic in a single repository.

**Prerequisites:**

* Git & GitHub account
* Python 3.x installed
* Basic knowledge of data analysis

**Steps:**

1. Create a new GitHub repository: metrics-repo.
2. Inside the repo, create folder structure:

metrics\_repo/

analyzers/

\_\_init\_\_.py

churn\_analyzer.py

sales\_analyzer.py

tests/

test\_churn.py

1. Implement churn\_analyzer.py:

def calculate\_churn\_rate(df):

total\_customers = len(df)

churned = len(df[df['status'] == 'churned'])

return churned / total\_customers

1. Write a test in test\_churn.py using pytest:

from metrics\_repo.analyzers.churn\_analyzer import calculate\_churn\_rate

import pandas as pd

def test\_churn\_rate():

df = pd.DataFrame({'status': ['active', 'churned', 'active', 'churned']})

assert calculate\_churn\_rate(df) == 0.5

1. Push the repo to GitHub.

**Validation:**

* Run pytest locally; all tests pass.
* Repo has clear structure and README.

**Lab 2 – Red-Green-Refactor TDD**

**Objective:**  
Practice TDD cycle with metric calculation.

**Steps:**

1. **Red** – Write a failing test for a new metric (e.g., avg\_order\_value).
2. **Green** – Implement the simplest code to make it pass:

def avg\_order\_value(df):

return df['order\_value'].mean()

1. **Refactor** – Optimize by adding error handling for empty datasets.

**Validation:**

* Commit history clearly shows Red → Green → Refactor steps.

**Lab 3 – Drift Monitoring**

**Objective:**  
Detect data drift using statistical tests.

**Steps:**

1. Create two CSV datasets: data\_jan.csv and data\_feb.csv.
2. Load them and compare distribution of key features:

from scipy.stats import ks\_2samp

import pandas as pd

df\_jan = pd.read\_csv('data\_jan.csv')

df\_feb = pd.read\_csv('data\_feb.csv')

stat, p\_value = ks\_2samp(df\_jan['feature1'], df\_feb['feature1'])

print("P-value:", p\_value)

if p\_value < 0.05:

print("Drift detected!")

else:

print("No drift.")

1. Log results into a metrics table.

**Validation:**

* Script correctly flags drift when dataset changes are introduced.

**Lab 4 – GitHub Actions Integration**

**Objective:**  
Set up automated testing for metrics repo.

**Steps:**

1. In the metrics repo, create .github/workflows/ci.yml:

name: Metrics CI

on: [push, pull\_request]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-python@v4

with:

python-version: '3.x'

- run: pip install pytest pandas scipy

- run: pytest

1. Commit and push; observe GitHub Actions running tests.

**Validation:**

* PRs fail if tests fail.
* Workflow status visible in GitHub UI.

**Lab 5 – Auto-Rollback Script**

**Objective:**  
Implement rollback in case of failed deployment.

**Steps:**

1. Create a script deploy.sh:

#!/bin/bash

set -e

echo "Deploying new version..."

# Simulate deployment failure

if [ "$1" == "fail" ]; then

echo "Deployment failed! Rolling back..."

git checkout stable

else

echo "Deployment succeeded!"

fi

1. Tag stable commit and test rollback.

**Validation:**

* Script reverts to stable state on failure.

**Lab 6 – Metrics Dashboard**

**Objective:**  
Visualize metrics using Grafana or Power BI.

**Steps:**

1. Store metrics in a PostgreSQL or Delta table.
2. Connect dashboard tool to the metrics data source.
3. Create visualizations:
   * Churn rate over time
   * Data drift alerts
   * Build success rate from CI pipeline

**Validation:**

* Dashboard updates automatically when new metrics are logged.