Real-Time Public Transit Anomaly Detection and Alerting System

1. **Project Title:** Real-Time Public Transit Anomaly Detection and Alerting System
2. **Project Goal/Objective:** To develop a prototype system that processes simulated real-time or recent historical public transit data to detect anomalies (e.g., significant delays, unusual clustering) and presents these findings via a deployed, accessible web dashboard.
3. **Problem Statement/Background:** Public transit systems generate vast operational data. Manual monitoring is inefficient. An automated system highlighting anomalies can help operators identify potential problems quickly, enabling faster responses and improving service reliability.
4. **Scope:**
    - **In Scope:** Ingesting historical/simulated time-series transit data; developing ML models for anomaly detection; creating a dashboard application to visualize transit status and alerts; containerizing the dashboard; deploying the dashboard to be web-accessible; potentially setting up a scheduled job to update anomaly data periodically.
    - **Out of Scope:** True real-time stream processing; integration with live transit APIs; predicting future delays; complex user management for the dashboard.
5. **Key Deliverables:**
    - Documented data simulation/acquisition process.
    - Trained anomaly detection model(s) (code and potentially saved model files).
    - A containerized (Docker) dashboard web application.
    - Deployed dashboard accessible via a URL.
    - (Optional but Recommended) Code for a scheduled job to update anomaly results.
    - Final technical report detailing methodology, system design, deployment process, model evaluation, and results.
    - Final presentation demonstrating the deployed dashboard and findings.
6. **Potential Datasets to Get Started:**
    - **Toronto Transit Commission (TTC) Open Data:** GTFS and GTFS-RT feeds/archives. Need to investigate archiving or capture data.
        - *Link:* https://www.google.com/search?q=https://www.ttc.ca/transparency-and-accountability/Open-Data
    - **TransitFeeds / OpenMobilityData:** Aggregated GTFS/GTFS-RT data from various agencies.
        - *Link:* https://database.mobilitydata.org/
    - **City of Toronto Open Data Portal:** Supplementary data (traffic, road closures).
        - *Link:* https://open.toronto.ca/
    - **Note:** Passenger counts likely need simulation. Focus on location/schedule deviations.
7. **Potential Technical Approach:**

- **Data:** Use historical GTFS/GTFS-RT. Simulate missing components. Clean/structure time-series data.
- **ML Models:** Time-series analysis, clustering (DBSCAN, K-Means), statistical methods (Z-score, IQR), potentially LSTMs. Unsupervised learning is likely.
- **Dashboard Framework:** Python libraries like Plotly Dash or Streamlit.
- **Containerization:** Use Docker for the dashboard application. If implementing a separate update job, containerize that too.
- **Cloud Platform:** AWS (e.g., Elastic Beanstalk, App Runner, Fargate for dashboard; Lambda/Fargate + EventBridge for scheduled job), GCP (e.g., Cloud Run, App Engine for dashboard; Cloud Functions/Cloud Run + Cloud Scheduler for job), Azure (e.g., App Service, Container Apps for dashboard; Functions + Timer Trigger/Logic Apps for job).

8. **Deployment Requirements & Tips:**
   - **Requirement:** The final dashboard application must be deployed as a containerized web application on a chosen cloud platform (AWS, GCP, or Azure), accessible via a public URL for demonstration. The system should process data (either on a schedule or on dashboard load using recent data) to display anomalies.
   - **Tips for Students:**
     - **Deploy Dashboard First:** Focus on getting the Dash/Streamlit application containerized and running on the cloud service, perhaps initially with static sample data.
     - **Data Update Strategy:** Decide how the dashboard gets fresh anomaly data.
       - *Option 1 (Simpler):* Package recent historical data within the container or have the dashboard load it from cloud storage on startup. Analysis happens when the dashboard runs. Good for demos, not truly "updating".
       - *Option 2 (Better):* Implement a separate, scheduled cloud function/job (e.g., runs daily/hourly) that performs anomaly detection on new data, saves results (e.g., to cloud storage, a simple database), and the dashboard reads these latest results.
     - **Containerize:** Docker is essential for packaging the dashboard and its dependencies (including ML libraries).
     - **Choose Cloud Service:** For dashboards, services like Google Cloud Run, Azure App Service/Container Apps, or AWS Elastic Beanstalk/App Runner are often good choices as they handle web serving.
     - **Permissions:** If using Option 2, ensure the scheduled job has permission to write to the data store, and the dashboard app has permission to read from it (configure IAM roles/policies).
     - **Dashboard Performance:** Be mindful of how much data/computation the dashboard tries to do on load, especially if not using pre-calculated results. Optimize data loading and plotting.

        - **Resource:** Use cloud provider documentation for deploying containerized web apps (Dash/Streamlit have specific guides too). Leverage `Cloud Computing for Machine Learning` course concepts.
9. **Potential Challenges:** Acquiring comprehensive data; defining anomalies; handling spatial-temporal data; dashboard performance; debugging deployment issues; managing cloud service interactions and permissions.
10. **Success Metrics:** Deployed dashboard functionality and usability; validity/insightfulness of detected anomalies; quality of analysis/report; successful demonstration.
11. **Team Size / Duration:** 3-4 Students / 14 Weeks