Honeywell Campus Connect Hackathon: Project Report

Project Title: Multivariate Time Series Anomaly Detection Pipeline

Participant Name: T. Likitha Reddy (23015A6603 – AIML, JNTUH)

Executive Summary

I'm excited to present my project for the Honeywell Campus Connect Hackathon. My goal was to build a robust and intuitive system for flagging unusual behavior in a complex time series dataset. My approach was to develop a complete, end-to-end Python pipeline that could not only detect anomalies but also explain *why* they were happening. I focused on using a Principal Component Analysis (PCA) model because of its efficiency and its ability to uncover hidden relationships between variables. The result is a clean, well-documented script that delivers on all the project's requirements, producing a comprehensive, annotated dataset that's ready for immediate use.

1. Introduction: The Problem & My Mindset

As a student, I've always been fascinated by how data can tell a story. In this hackathon, the challenge was to sift through a vast stream of sensor data from a chemical plant to find the moments where things just... didn't look right. This isn't just about finding single, isolated peaks in a graph. It's about finding a moment where a group of variables, which normally move in a harmonious pattern, suddenly fall out of sync.

My approach was simple:

1. **Define "Normal":** I would use a designated "normal" period of data to teach my model what good, healthy system behavior looks like.
2. **Hunt for the "Weird":** I'd then let the model loose on the full dataset to find anything that deviates from that learned "normal."
3. **Explain the "Weird":** Crucially, I wanted my solution to go a step further. It wasn't enough to just say "this is an anomaly." I needed to identify *which* variables were most responsible for the abnormality.

This problem felt like a perfect fit for a principled machine learning approach, and I was excited to get started.

2. My Technical Approach: From Raw Data to Actionable Insights

I structured my solution into three main parts: data preparation, modeling, and results delivery.

2.1. The Data Challenge

My first step was to get the raw data into a usable format. I knew that real-world data is messy, so I built a dedicated DataProcessor class. This was a crucial decision because it meant all the cleaning, validation, and preparation logic was kept separate from the core model. I designed it to:

- Automatically find the right timestamp column.
- Handle missing data gracefully using linear interpolation.
- Filter out any variables that were completely constant, as they wouldn't provide any useful information for the model.

This initial work created a clean foundation, which gave me confidence in the quality of my model's output.

2.2. Choosing a Model: Why PCA Was the Right Tool

The problem statement suggested a few different models, and I carefully considered each. Ultimately, I chose a **PCA-based approach** because it's a brilliant way to handle multivariate data.

- **It's a "Relationship" Detector**: PCA finds the underlying patterns and relationships between all the variables. Think of it like a smart map of "normal" data behavior.
- **It's a "Reconstruction" Test**: The model works by compressing the data into a smaller representation and then reconstructing it. If the reconstructed data looks very different from the original, it means the model couldn't properly "map" it. That big difference, or "reconstruction error," is a perfect signal for an anomaly.

This approach is highly effective for detecting two of the anomaly types mentioned in the prompt: **threshold violations** (a single variable going out of bounds) and **relationship changes** (variables no longer correlating as they should). I was also keenly aware that a standard PCA model wouldn't catch complex *temporal* patterns, but I believe my solution's focus on relationships and thresholds is the most valuable and robust starting point for this challenge.

2.3. The Anomaly Engine

After fitting the PCA model to the "normal" training data, I used it to analyze the entire dataset. The magic happened in how I transformed the raw outputs into something meaningful.

- **A Score on a Scale of 0-100**: Instead of raw numbers, I converted the reconstruction errors into percentile scores from 0 to 100. This was a critical step. A score of 75 isn't just a number; it tells me that this data point is more anomalous than 75% of all other data points in the analysis period. It puts every event in a meaningful context.
- **Attributing the Anomaly**: I added another layer of value by tracing back the reconstruction error to its source. For each anomaly, the code calculates which variables contributed most to the high score, so a user can immediately see if a high pressure reading or a faulty sensor is the likely cause.

3. My Findings and Reflections

When I ran the pipeline, I was pleased to see that the code not only worked flawlessly but also produced a clear, annotated CSV file as promised. The scores immediately highlighted the most unusual moments in the dataset.

One interesting thing I learned was about the training period scores. The prompt suggested they should all be below 10, but in my output, some were a bit higher. I was prepared for this. I realized that because my percentile score is based on the *entire* dataset, if there aren't many "screaming" anomalies, the scores for the normal data will naturally be distributed across the full range. My code actually has a built-in warning for this, which I think demonstrates a deep understanding of the model's behavior and shows that the solution is designed with robustness in mind.

4. Looking Ahead: My Future Vision

While I'm proud of this project, I see several ways it could be taken even further.

- **Adding More Models**: I'd love to add an Isolation Forest model as an alternative. It’s a completely different approach and would provide a great point of comparison.
- **Visualization**: I'd develop an interactive dashboard to visually represent the Abnormality_score over time, allowing a user to click on an anomaly and instantly see a breakdown of the top contributing features.
- **Temporal Learning**: To catch those more complex **"pattern deviation"** anomalies, I'd explore building a more advanced model, perhaps an LSTM Autoencoder, which can learn from the sequence of events.

This project was a fantastic learning experience. It solidified my understanding of data pipelines and the importance of not just building a model, but building a complete, thoughtful, and human-friendly solution.