■ Launch Vehicle Anomaly Detector

Project Walkthrough — Full Learning Guide

Days 1 – 4 | Python · NumPy · Pandas · ReportLab

■ The Big Picture

The goal of this project is to build a system that **automatically detects when something is wrong with a rocket during flight**, using only sensor data. The project is organised into modular day-by-day components that form a clean pipeline from data generation through to anomaly detection.

Day	File	Role	Output
1	day1_generator.py	Base telemetry simulator	time, altitude, velocity, engine_temp
2	day2_physics.py	Physics sensor models	fuel_pressure, vibration
3	anomalies.py	Fault injectors	corrupted signal arrays
3	assemble_dataset.py	Training dataset builder	data/train_normal.csv
4	make_test.py	Labelled test dataset builder	data/test_anomalies.csv

■ Day 1 — day1_generator.py

What it does: Simulates the basic physical signals a rocket produces during flight.

Function: `generate_telemetry()` — returns a 6,000-row DataFrame (600 s × 10 Hz).

Column	Formula	Real-world meaning
time	linspace(0, 600, 6000)	600 s at 10 readings/sec
altitude	$\text{time}^2 \times 0.25$	Parabolic climb (constant acceleration)
velocity	time × 0.5 + noise	Linear speed + sensor jitter
engine_temp	300 + time×0.1 + noise	Gradually heating engine (Kelvin)

Why time^2 for altitude?

From basic kinematics: $s = \frac{1}{2}at^2$. With $a = 0.5\ \text{m/s}^2$, altitude $= 0.25 \times t^2$. The rocket accelerates, so it climbs faster and faster — a parabolic curve.

Why add noise to velocity?

Real sensors are never perfect. Adding `np.random.normal(0, 1.0, n)` adds tiny random wobbles to mimic real sensor imprecision.

■ Day 2 — day2_physics.py

What it does: Adds two more realistic physical signals that require more complex physics modelling.

Function 1 (Jisto): `simulate_pressure(time_vector)`

```
P(t) = 5000 × e^(−0.008 × t) + noise
```

Part	Meaning
5000	Tank starts full — 5000 pressure units
e^(-0.008t)	Exponential decay — fuel consumed, pressure drops
noise	Gaussian jitter N(0, 50) — sensor imprecision
At t=600s	5000 × e^(-4.8) ≈ 41 units — tank nearly empty ■

Analogy: Like a balloon slowly deflating — pressure drops fast at first, then slows. That's exponential decay.

Function 2 (Devika): `simulate_vibration(velocity_vector)`

```
V(v) = A × exp( −(v − 300)² / (2 × 80²) ) × (v / 300) + noise
```

Part	Meaning
Max-Q (300 m/s)	Moment of peak aerodynamic stress on the rocket
Gaussian bell	Vibration peaks at Max-Q, lower on either side
velocity_scale	Ensures vibration also scales with overall speed

Analogy: Like turbulence on a plane — worst at a specific speed window, less severe before and after.

■ Day 3 — anomalies.py

What it does: Provides two functions to deliberately corrupt a clean signal, simulating sensor faults or real hardware failures.

Function 1: `inject_spike(signal, magnitude, probability)`

Randomly adds sudden sharp jumps to individual data points.

```
Normal: [100, 101, 99, 100, 100] Spiked: [100, 101, 99, 100, 450] ← sudden spike!
```

- Each sample independently has a *probability* (e.g. 2%) chance of being spiked
- Spike direction is random — can go up OR down
- Spike amplitude: Uniform(–magnitude, +magnitude)
- **Real-world analogy:** A loose wire causing a momentary incorrect reading

Function 2: `inject_drift(signal, drift_factor)`

Adds a slow, cumulative, ever-growing bias starting from a random time point.

```
Normal: [100, 101, 99, 100, 101] Drifted: [100, 101, 99, 108, 117] ← creeping upward from
index 3 bias grows: +8, +16, +24...
```

- Drift starts at a **random** onset index — simulates gradual sensor degradation
- Bias grows linearly: `drift_factor × (i − onset)`
- **Real-world analogy:** A temperature sensor slowly drifting out of calibration

■ *IMPORTANT: Both functions always return a **copy** — the original signal is never modified. This is safe programming practice.*

■ Day 3 — assemble_dataset.py

What it does: Combines Day 1 + Day 2 signals into one clean CSV with **no anomalies**. This is the training dataset.

Step	Function Called	Columns Added
1	generate_telemetry()	time, altitude, velocity, engine_temp
2	simulate_pressure(time)	fuel_pressure
3	simulate_vibration(vel)	vibration
4	df.to_csv(...)	→ data/train_normal.csv (6,000 rows, 6 cols)

■ NOTE: Why no anomalies here? The detector must first learn what normal looks like before it can spot deviations — just like a doctor studies healthy scans before identifying disease.

■ Day 4 — make_test.py

What it does: Generates a labelled test dataset WITH anomalies injected. Each row is tagged `is_anomaly = 1` if corrupted, else `0`.

Step	Action	Target Column	Result
1–3	Same as assemble_dataset.py	All 6 columns	Clean baseline
4	inject_spike (2% probability)	fuel_pressure	~120 rows spiked
5	inject_drift (factor=0.08)	engine_temp	~3000 rows drifting
6	Build is_anomaly mask	is_anomaly	1 if any column changed
7	Save to CSV	—	data/test_anomalies.csv

The `_changed_mask()` helper:

Compares the original vs. corrupted array element-by-element using `np.isclose()` to find exactly which rows were changed. The union of spike_mask and drift_mask gives the final `is_anomaly` labels.

■ Key Concepts — Quick Reference

Concept	What it means	Where used
Exponential decay	Value drops fast then levels off: e^(–kt)	simulate_pressure
Gaussian noise	Random normal jitter: N(mean, std)	All sensor signals
Max-Q	Peak aerodynamic stress at ~300 m/s	simulate_vibration
Spike anomaly	Sudden, random, isolated bad reading	inject_spike
Drift anomaly	Slow, cumulative sensor degradation	inject_drift
is_anomaly label	0 = normal row, 1 = corrupted row	make_test.py
Train/Test split	train_normal (clean) vs test_anomalies (faults)	Dataset design

■ Project File Map

```
launch_vehicle_anomaly_detection/ ■■■ src/ ■ ■■■ day1_generator.py ← Base telemetry
(altitude, velocity, temp) ■ ■■■ day2_physics.py ← Physics sensors (pressure,
vibration) ■ ■■■ anomalies.py ← Fault injection (spike, drift) ■ ■■■
assemble_dataset.py ← Builds train_normal.csv ■ ■■■ make_test.py ← Builds
test_anomalies.csv (labelled) ■■■ data/ ■ ■■■ train_normal.csv ← Clean data for
training ■ ■■■ test_anomalies.csv ← Corrupted + labelled for testing ■■■
requirements.txt ← numpy, pandas, matplotlib, scikit-learn, streamlit
```

■ What Comes Next

Day	Topic	Description
5	Z-score Detection	Flag readings statistically far from the training mean/std
6	Isolation Forest	ML model that isolates unusual data points in feature space
7	Evaluation Metrics	Precision, Recall, F1 — compare predictions vs is_anomaly labels
8	Streamlit Dashboard	Visual interface to see anomaly detections in real time