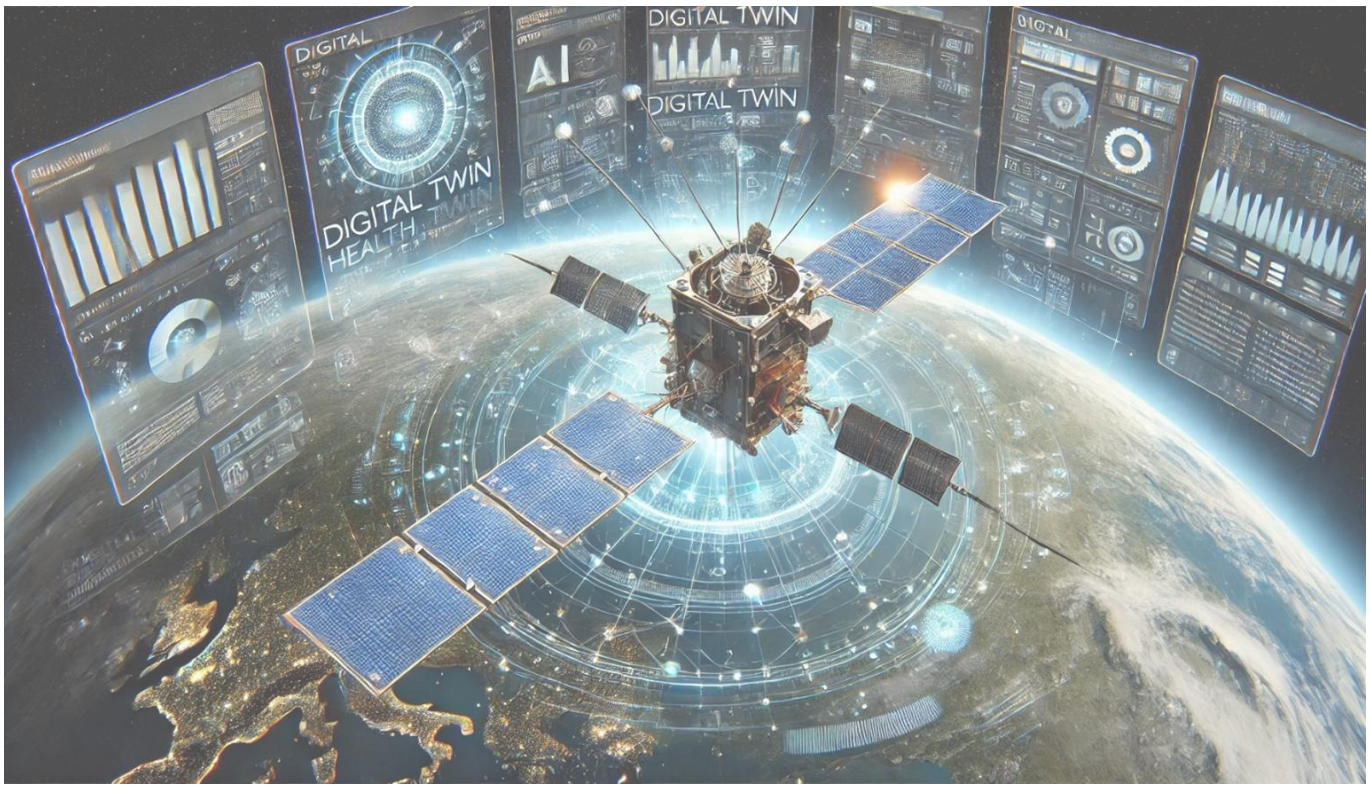
AI-POWERED DIGITAL TWIN PROTOTYPE FOR SATELLITE HEALTH MONITORING



- **Prepared by**

Madesh M

Data Science & AI Research and Development Intern

AI-POWERED DIGITAL TWIN PROTOTYPE FOR SATELLITE HEALTH MONITORING

Project Overview:

This task involves developing a prototype of an AI-powered digital twin for satellite health monitoring. The digital twin will simulate real-time satellite conditions and predict potential issues using AI models trained on simulated satellite telemetry data.

- Real-time Satellite Health Monitoring
- Predictive Analytics for anomaly detection
- Interactive Dashboard (built with Streamlit)
- Automated Alert System
- Visualization of Satellite Metrics

Usage

- Launch the dashboard using Streamlit.
- View real-time telemetry data and health status of satellites.
- Receive alerts if anomalies are detected.
- Access the Digital Twin simulation for in-depth analysis.

Phase 1: Data Collection & Digital Twin setup

- Identify key satellite parameters to simulate (e.g., battery voltage, solar panel efficiency, thruster health, etc.).
- Generate or use **pre-existing satellite telemetry datasets** for training & simulation.
- Set up a basic **database** (MySQL) to store simulated satellite data.

Dataset: Sample_satellite_telemetry_data.csv

Satellite Telemetry Parameters:

- timestamp
- battery_voltage
- battery_current
- state_of_charge
- solar_panel_voltage
- solar_panel_current
- solar_panel_efficiency
- power_consumption
- internal_temp
- battery_temp
- solar_panel_temp
- radiator_temp
- radiator_efficiency
- thermal_gradient
- position
- velocity
- gyroscope
- magnetometer_rpm
- reaction_wheel_rpm
- thruster_status
- signal_strength
- data_rate
- packet_loss
- payload_power
- sensor_data_rate
- camera_temp
- data_quality
- error_flags
- latency
- bit_error_rate
- sensor_discrepancies
- thruster_malfunctions
- thruster_efficiency
- orientation
- throughput
- power_anomalies
- thermal_anomalies
- aocs_faults
- payload_failures

Phase 2: AI Model Development:

- Train an anomaly detection model (Isolation Forest) using historical telemetry data.
- Test AI predictions against simulated satellite failures.

Phase 3: Real-Time Dashboard & Alerts

- Create an MYSQL to fetch Dataset and store the Predicted values.
- Implement an alert system (email) for detecting critical failures.
- Develop a dashboard (Streamlit) to visualize satellite status.

Phase 4: Deployment & Documentation:

- Deploy the digital twin prototype on free-tier cloud hosting (Render/Supabase/Vercel).
- Write setup documentation & AI model explanation.
- Conduct final testing & bug fixes**.**

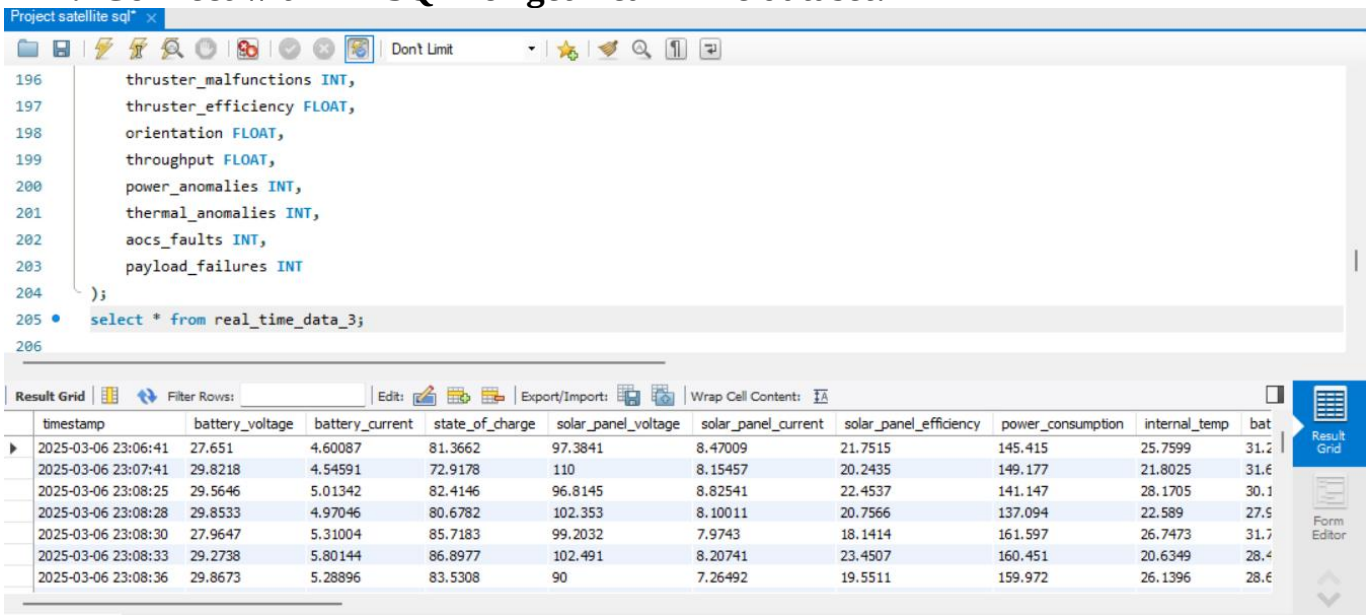
Localhost View:

You can now view your Streamlit app in your browser.

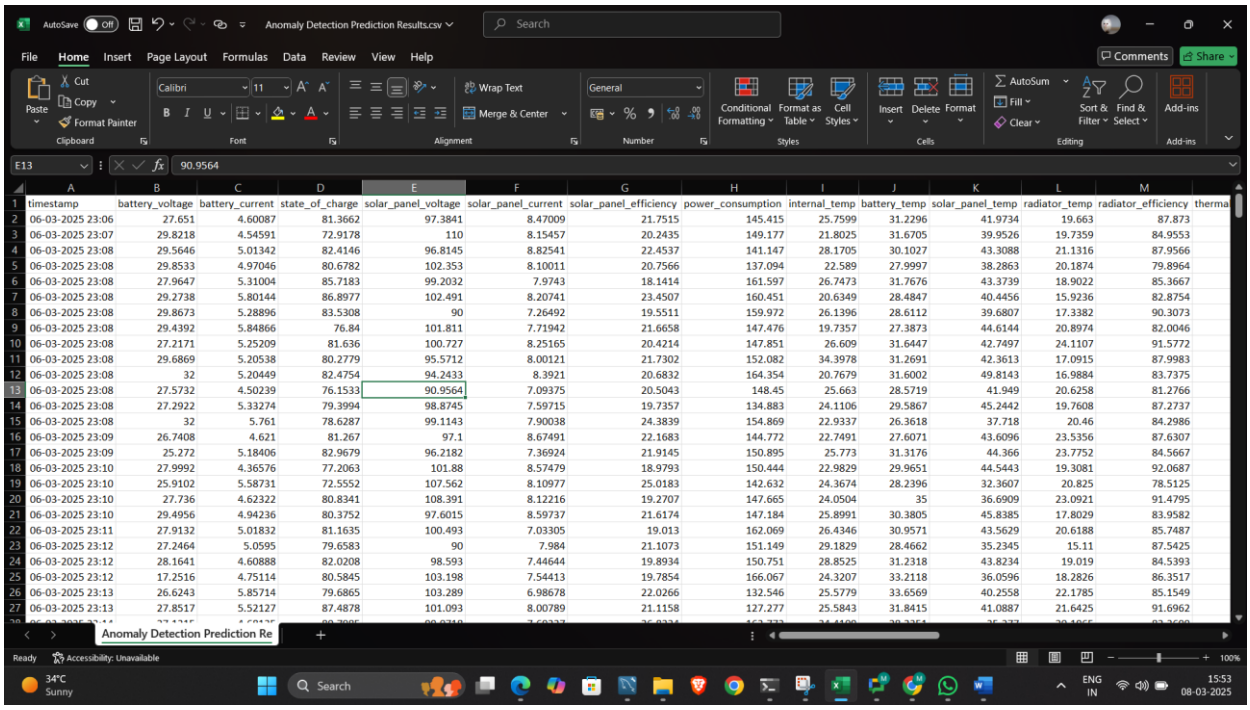
- Local URL: http://localhost:8501
- Network URL: http://192.168.1.33:8501

Screenshots:

1. Connect with MYSQL for get Real-Time dataset:

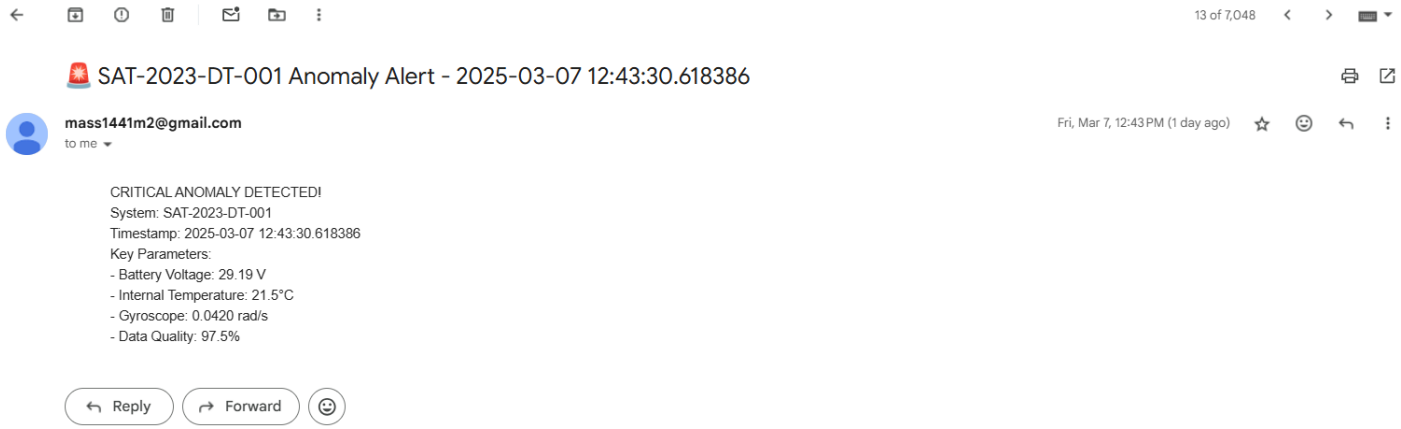


2. Dataset:



timestamp	battery_voltage	battery_current	state_of_charge	solar_panel_voltage	solar_panel_current	solar_panel_efficiency	power_consumption	internal_temp	battery_temp	solar_panel_temp	radiator_temp	radiator_efficiency	thermal
06-03-2025 23:06	27.651	4.60087	81.3662	97.3841	8.47009	21.7515	145.415	25.7599	31.2296	41.9734	19.663	87.873	
06-03-2025 23:07	29.8218	4.54591	72.9178	110	8.15457	20.2435	149.177	21.8025	31.6705	39.9526	19.7359	84.9553	
06-03-2025 23:08	29.5646	5.01342	82.4146	96.8145	8.82541	22.4537	141.147	28.1705	30.1027	43.3088	21.1316	87.9566	
06-03-2025 23:08	29.8533	4.97046	80.6782	102.353	8.10011	20.7566	137.094	22.589	27.9997	38.2863	20.1874	79.8964	
06-03-2025 23:08	27.9647	5.31004	85.7183	99.2032	7.9743	18.1414	161.597	26.7473	31.7676	43.3739	18.9022	85.3667	
06-03-2025 23:08	29.2738	5.80144	86.8977	102.491	8.20741	23.4507	160.451	20.6349	28.4847	40.4456	15.9236	82.8754	
06-03-2025 23:08	29.8673	5.28896	83.5308	90	7.26492	19.5511	159.972	26.1396	28.6112	39.6807	17.3382	90.3073	
06-03-2025 23:08	29.4392	5.84866	76.84	101.811	7.71942	21.6658	147.476	19.7357	27.3873	44.6144	20.8974	82.0046	
06-03-2025 23:08	27.2171	5.25209	81.636	100.727	8.25165	20.4214	147.851	26.609	31.6447	42.7497	24.1107	91.5772	
06-03-2025 23:08	29.6869	5.20538	80.2779	95.5712	8.00121	21.7302	152.082	34.3978	31.2691	42.3613	17.0915	87.9983	
06-03-2025 23:08	32	5.20449	82.4754	94.2433	8.3921	20.6832	164.354	20.7679	31.6002	49.8143	16.9884	83.7375	
06-03-2025 23:08	27.5732	4.50239	76.1533	90.9564	7.09375	20.5043	148.45	25.663	28.5719	41.949	20.6258	81.2766	
06-03-2025 23:08	27.2922	5.33274	79.3994	98.8745	7.59715	19.7357	134.883	24.1106	29.5867	45.2442	19.7608	87.2737	
06-03-2025 23:08	32	5.761	78.6287	99.1143	7.90038	24.3839	154.869	22.9337	26.3618	37.718	20.46	84.2986	
06-03-2025 23:09	26.7408	4.621	81.267	97.1	8.67491	22.1683	144.772	22.7491	27.6071	43.6096	23.5356	87.6307	
06-03-2025 23:09	25.272	5.18406	82.9679	96.2182	7.36924	21.9145	150.895	25.773	31.3176	44.366	23.7752	84.5667	
06-03-2025 23:10	27.9992	4.36576	77.2063	101.88	8.57479	18.9793	150.444	22.9829	29.9651	44.5443	19.3081	92.0687	
06-03-2025 23:10	25.9102	5.58731	72.5552	107.562	8.10977	25.0183	142.632	24.3674	28.2396	32.3607	20.825	78.5125	
06-03-2025 23:10	27.736	4.62322	80.8341	108.391	8.12216	19.2707	147.665	24.0504	35	36.6909	23.0921	91.4795	
06-03-2025 23:10	29.4956	4.94236	80.3752	97.6015	8.59737	21.6174	147.184	25.8991	30.3805	45.8385	17.8029	83.9582	
06-03-2025 23:11	27.9132	5.01832	81.1635	100.493	7.03305	19.013	162.069	26.4346	30.9571	43.5629	20.6188	85.7487	
06-03-2025 23:12	27.2464	5.0595	79.6583	90	7.984	21.1073	151.149	29.1829	28.4662	35.2345	15.11	87.5425	
06-03-2025 23:12	28.1641	4.60888	82.0208	98.593	7.44644	19.8934	150.751	28.8525	31.2318	43.8234	19.019	84.5393	
06-03-2025 23:12	17.2516	4.75114	80.5845	103.198	7.54413	19.7854	166.067	24.3207	33.2118	36.0596	18.2826	86.3517	
06-03-2025 23:13	26.6243	5.85714	79.6865	103.289	6.98678	22.0266	132.546	25.5779	33.6569	40.2558	22.1785	85.1549	
06-03-2025 23:13	27.8517	5.52127	87.4878	101.093	8.00789	21.1158	127.277	25.5843	31.8415	41.0887	21.6425	91.6962	

3. Alert System Output



13 of 7,048

SAT-2023-DT-001 Anomaly Alert - 2025-03-07 12:43:30.618386

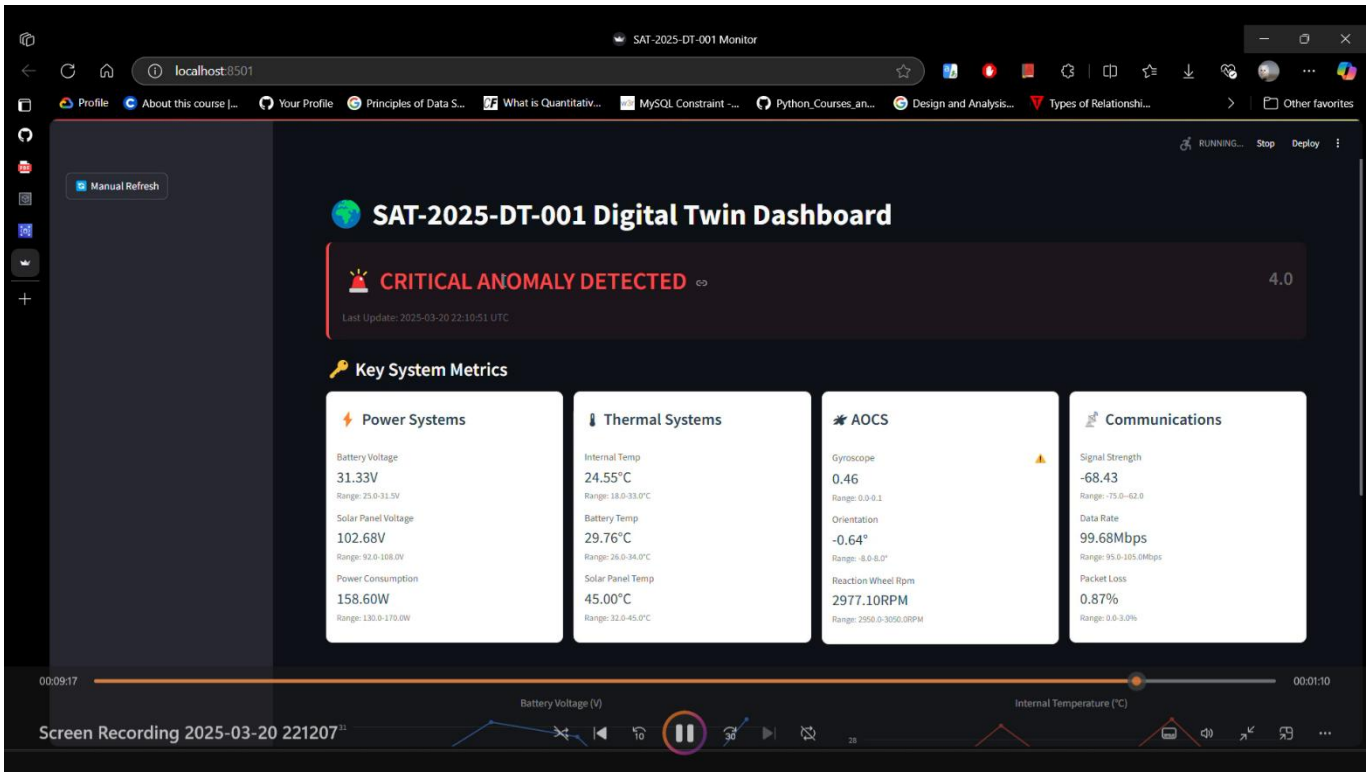
mass1441m2@gmail.com
to me

Fri, Mar 7, 12:43 PM (1 day ago)

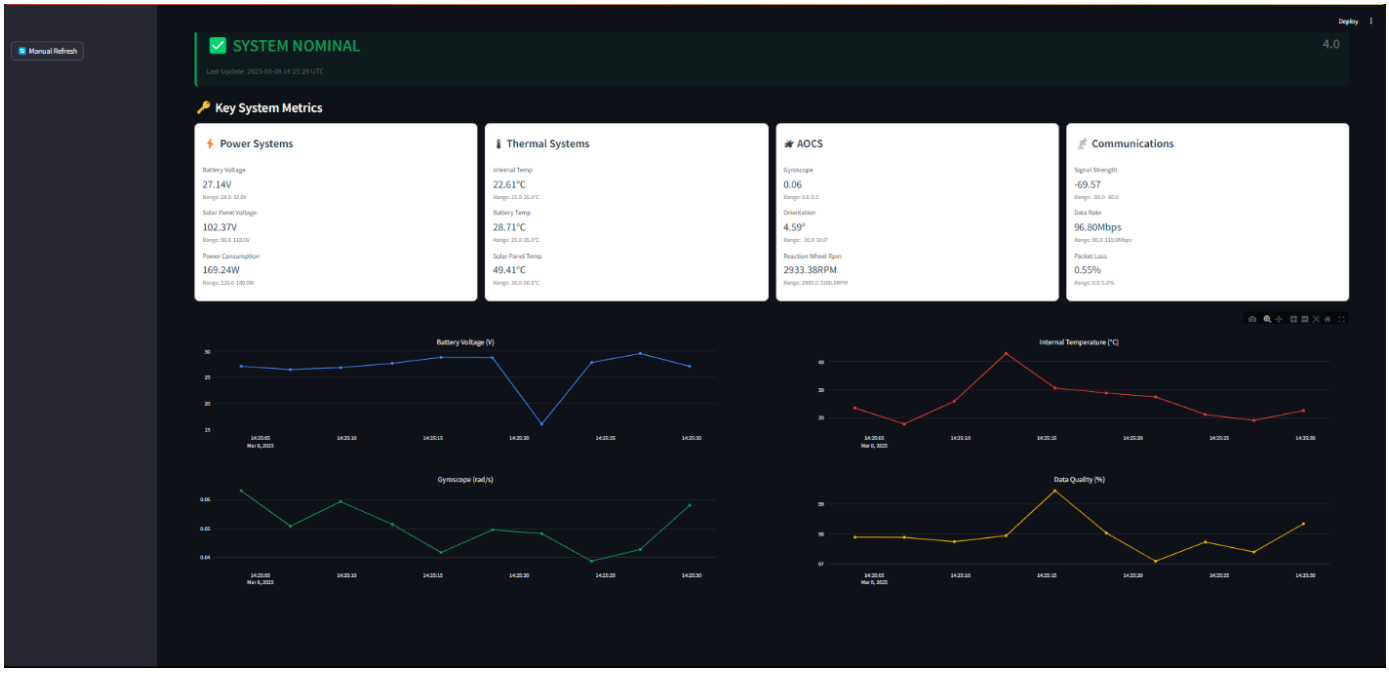
CRITICAL ANOMALY DETECTED!
System: SAT-2023-DT-001
Timestamp: 2025-03-07 12:43:30.618386
Key Parameters:
- Battery Voltage: 29.19 V
- Internal Temperature: 21.5°C
- Gyroscope: 0.0420 rad/s
- Data Quality: 97.5%

Reply Forward

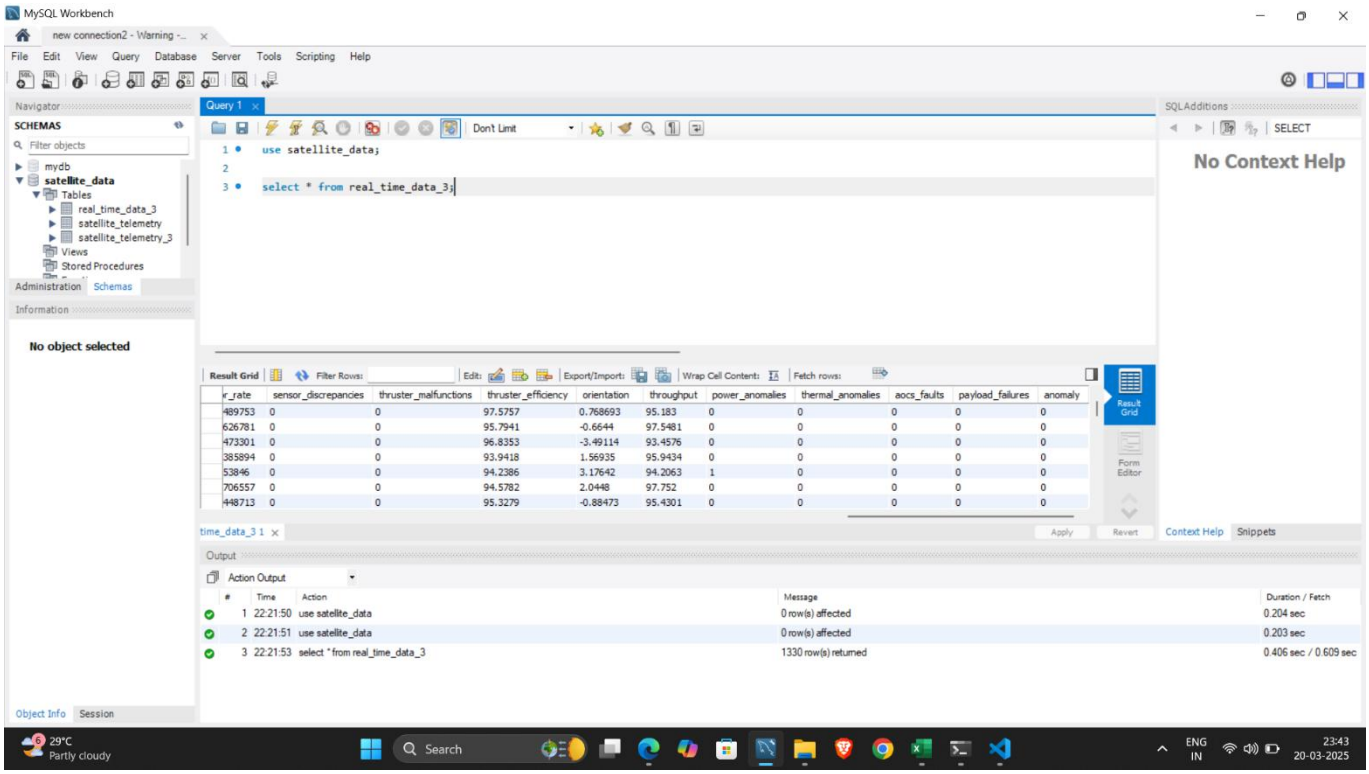
4. Anomaly Detected:



5. Streamlit Dashboard:



6. Output of Anomaly detected to Store in MYSQL:



Using Libraries:

1. Database & Data Handling

- mysql.connector → Connects to MySQL databases.
- pandas → Handles dataframes and data manipulation.
- numpy → Supports numerical computations.

2. Machine Learning & Preprocessing

- joblib → Loads the Isolation Forest model.
- sklearn.preprocessing.MinMaxScaler → Normalizes/scales data.

3. Web Framework

- streamlit → Builds the interactive web dashboard.

4. Email Handling

- smtplib → Sends emails using SMTP.
- email.mime.multipart.MIMEMultipart → Structures email messages.
- email.mime.text.MIMEText → Handles email text content.

5. Visualization

- plotly.graph_objects → Creates charts and visualizations.
- plotly.subplots.make_subplots → Combines multiple charts in one figure.

6. Miscellaneous

- random → Generates random numbers for simulated data.
- time → Handles time-related operations.
- datetime → Deals with timestamps.
- timedelta → Manipulates date and time intervals.

Project GitHub link - https://github.com/madesh6554/Satellite-Health-Monitoring-DT

Demo Video File link - Satellite Health Monitoring Demo Video.mp4 - Google Drive

Sources Code:

```
import mysql.connector
import pandas as pd
import joblib
import streamlit as st
import numpy as np
import time
import smtplib
from dotenv import load_dotenv
import os
import random
from datetime import datetime, timedelta
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from sklearn.preprocessing import StandardScaler
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.ensemble import IsolationForest
import joblib
load_dotenv()
DIGITAL_TWIN_VERSION = "4.0"
SYSTEM_ID = "SAT-2025-DT-001"
DB_CONFIG = {
    "host": os.getenv("DB_HOST"),
    "user": os.getenv("DB_USER"),
    "password": os.getenv("DB_PASSWORD"),
    "database": os.getenv("DB_NAME"),
    "table": os.getenv("DB_TABLE"),
    "port": os.getenv("DB_PORT", 3306),
    "ssl_ca": "global-bundle.pem"
}
```

```
ALERT_CONFIG = {
    "sender": os.getenv("EMAIL_SENDER"),
    "receiver": os.getenv("EMAIL_RECEIVER"),
    "password": os.getenv("EMAIL_PASSWORD")
}


class DatabaseManager:
    def __enter__(self):
        self.conn = mysql.connector.connect(
            host=DB_CONFIG["host"],
            user=DB_CONFIG["user"],
            password=DB_CONFIG["password"],
            database=DB_CONFIG["database"]
        )
        return self.conn.cursor()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.conn.commit()
        self.conn.close()

class DataGenerator:
    def __init__(self):
        self.last_timestamp = datetime.now() - timedelta(minutes=1)
        self.param_config = {
            "timestamp": None,

            # Power Systems
            "battery_voltage": (25.0, 31.5, 28.0, 1.0),
            "battery_current": (3.5, 6.5, 5.0, 0.5),
            "state_of_charge": (65, 98, 80, 2),
            "solar_panel_voltage": (92.0, 108.0, 100.0, 3.0),
```

"solar_panel_current": (6.5, 9.5, 8.0, 0.4),

"solar_panel_efficiency": (18, 26, 22, 1.5),

"power_consumption": (130.0, 170.0, 150.0, 5.0),

Thermal Systems

"internal_temp": (18.0, 33.0, 25.0, 2.0),

"battery_temp": (26.0, 34.0, 30.0, 1.5),

"solar_panel_temp": (32.0, 45.0, 40.0, 3.0),

"radiator_temp": (16.0, 24.0, 20.0, 1.5),

"radiator_efficiency": (80, 92, 85, 2),

"thermal_gradient": (4, 6, 5, 0.5),

Navigation and Control

"position": (350, 480, 400, 30),

"velocity": (7.5, 7.7, 7.6, 0.05),

"gyroscope": (0.01, 0.08, 0.05, 0.005),

"magnetometer_rpm": (4850, 5150, 5000, 75),

"reaction_wheel_rpm": (2950, 3050, 3000, 25),

Communications

"thruster_status": (0, 1, 0.98),

"signal_strength": (-75.0, -62.0, -70.0, 3.0),

"data_rate": (95.0, 105.0, 100.0, 2.5),

"packet_loss": (0.0, 3.0, 0.5, 0.2),

Payload and Sensors

"payload_power": (47, 53, 50, 1.5),

"sensor_data_rate": (9.5, 10.5, 10.0, 0.3),

"camera_temp": (12.0, 18.0, 15.0, 1.5),

"data_quality": (97, 100, 98.5, 0.5),

Error Handling and Latency

```
        "error_flags": (0, 1, 0.98),
        "latency": (150, 250, 200, 25),
        "bit_error_rate": (1e-7, 1e-5, 5e-6, 1e-6),

        # Anomaly and Fault Detection
        "sensor_discrepancies": (0, 1, 0.99),
        "thruster_malfunctions": (0, 1, 0.995),
        "thruster_efficiency": (92, 98, 95, 1.5),

        # Orientation and Throughput
        "orientation": (-8.0, 8.0, 0.0, 2.0),
        "throughput": (93.0, 98.0, 95.0, 1.5),

        # Anomaly Flags
        "power_anomalies": (0, 1, 0.97),
        "thermal_anomalies": (0, 1, 0.97),
        "aocs_faults": (0, 1, 0.98),
        "payload_failures": (0, 1, 0.98)
    }

def _generate_value(self, config):
    if len(config) == 4:
        min_val, max_val, mean, std = config
        return np.clip(random.gauss(mean, std), min_val, max_val)
    return 0 if random.random() < config[2] else 1

def generate_data_point(self):
    self.last_timestamp += timedelta(minutes=1)
    data = {"timestamp": self.last_timestamp}
    for param, config in self.param_config.items():
        if param != "timestamp":
            data[param] = self._generate_value(config)
```

```
        if random.random() < 0.07:
            anomaly_type = random.choice(["power", "thermal", "aocs", "payload"])
            if anomaly_type == "power":
                data["battery_voltage"] *= 0.6
                data["power_anomalies"] = 1
            elif anomaly_type == "thermal":
                data["internal_temp"] += 20
                data["thermal_anomalies"] = 1
            elif anomaly_type == "aocs":
                data["gyroscope"] *= 10
                data["aocs_faults"] = 1
            else:
                data["payload_failures"] = 1
                data["data_quality"] = 0
        return data


class AnomalyDetector:
    def __init__(self):
        self.model = IsolationForest(contamination=0.05, random_state=42)  # Initialize Isolation Forest
        self.scaler = StandardScaler()
        self.features = [
            "battery_voltage", "battery_current", "state_of_charge",
            "solar_panel_voltage", "solar_panel_current", "solar_panel_efficiency",
            "power_consumption", "internal_temp", "battery_temp",
            "solar_panel_temp", "radiator_temp", "radiator_efficiency",
            "thermal_gradient", "position", "velocity", "gyroscope",
            "magnetometer_rpm", "reaction_wheel_rpm", "thruster_status",
            "signal_strength", "data_rate", "packet_loss", "payload_power",
            "sensor_data_rate", "camera_temp", "data_quality", "error_flags",
            "latency", "bit_error_rate", "sensor_discrepancies",
            "thruster_malfunctions", "thruster_efficiency", "orientation",
            "throughput", "power_anomalies", "thermal_anomalies",
```

```
            "aocs_faults", "payload_failures"
        ]
        self._train_model()

    def _train_model(self):
        # Generate some initial data to train the model
        generator = DataGenerator()
        data_points = [generator.generate_data_point() for _ in range(100)]
        df = pd.DataFrame(data_points)
        scaled_data = self.scaler.fit_transform(df[self.features])
        self.model.fit(scaled_data)

        # Save the trained model and scaler
        joblib.dump(self.model, "isolation_forest_model.pkl")
        joblib.dump(self.scaler, "scaler.pkl")

    def predict(self, data):
        df = pd.DataFrame([data])
        scaled_data = self.scaler.transform(df[self.features])
        return self.model.predict(scaled_data)[0] == -1

class AlertSystem:
    def send_alert(self, data):
        try:
            msg = MIMEMultipart()
            msg['From'] = ALERT_CONFIG["sender"]
            msg['To'] = ALERT_CONFIG["receiver"]
            msg['Subject'] = f"🚨 {SYSTEM_ID} Anomaly Alert - {data['timestamp']}"
            body = f"""
            CRITICAL ANOMALY DETECTED!
            System: {SYSTEM_ID}
            Timestamp: {data['timestamp']}
```

```
            Key Parameters:
            - Battery Voltage: {data['battery_voltage']:.2f} V
            - Internal Temperature: {data['internal_temp']:.1f}°C
            - Gyroscope: {data['gyroscope']:.4f} rad/s
            - Data Quality: {data['data_quality']:.1f}%"""
            msg.attach(MIMEText(body, 'plain'))
            server = smtplib.SMTP('smtp.gmail.com', 587)
            server.starttls()
            server.login(ALERT_CONFIG.get("sender"), ALERT_CONFIG.get("password"))
            server.sendmail(ALERT_CONFIG["sender"], ALERT_CONFIG["receiver"], msg.as_string())
            server.quit()
        except Exception as e:
            st.error(f"Alert failed: {str(e)}")


class SatelliteDashboard:
    def __init__(self, data_generator):
        self.generator = data_generator
        st.set_page_config(page_title=f"{SYSTEM_ID} Monitor", layout="wide")
        st.title(f"🌎 {SYSTEM_ID} Digital Twin Dashboard")
        self.status_container = st.empty()
        self.metrics_container = st.container()
        self.chart_container = st.empty()
        st.sidebar.button("🔄 Manual Refresh", key="refresh")
        if 'history' not in st.session_state:
            self._load_initial_data()


    def _load_initial_data(self):
        with DatabaseManager() as cursor:
            cursor.execute(f"SELECT * FROM {DB_CONFIG['table']} ORDER BY timestamp DESC LIMIT
10")
            st.session_state.history = [dict(zip([col[0] for col in cursor.description], row)) for row in
cursor.fetchall()]
```

```
    def update_display(self, data, anomaly):
        self._update_status(data, anomaly)
        self._update_metrics(data)
        self._update_charts()

    def _update_status(self, data, anomaly):
        status_color = "#FF4B4B" if anomaly else "#0F9D58"
        self.status_container.markdown(f"""
            <div style="padding:20px; background:{status_color}10; border-radius:10px; margin-bottom:20px;
                        border-left:5px solid {status_color}; box-shadow: 0 2px 4px rgba(0,0,0,0.1)">
                <h2 style="color:{status_color}; margin:0;">
                    {' 🚨 CRITICAL ANOMALY DETECTED' if anomaly else '✅ SYSTEM NOMINAL'}
                    <span style="float:right; font-size:0.8em; color:#666;">{DIGITAL_TWIN_VERSION}</span>
                </h2>
                <p style="margin:5px 0 0 0; color:#666;">
                    Last Update: {data['timestamp'].strftime('%Y-%m-%d %H:%M:%S UTC')}
                </p>
            </div>""", unsafe_allow_html=True)

    def _update_metrics(self, data):
        with self.metrics_container:
            st.subheader("🔑 Key System Metrics")
            cols = st.columns(4)
            metrics = [
                ("⚡ Power Systems", ['battery_voltage', 'solar_panel_voltage', 'power_consumption']),
                ("🌡️ Thermal Systems", ['internal_temp', 'battery_temp', 'solar_panel_temp']),
                ("🛰️ AOCS", ['gyroscope', 'orientation', 'reaction_wheel_rpm']),
                ("📡 Communications", ['signal_strength', 'data_rate', 'packet_loss'])
            ]
```

```
        for col, (title, params) in zip(cols, metrics):
            with col:
                st.markdown(self._build_metric_card(title, params, data), unsafe_allow_html=True)

    def _build_metric_card(self, title, params, data):
        html = f"""
        <div style="padding:15px; background:#FFFFFF; border-radius:10px; border:1px solid #EEE;
                    margin-bottom:20px; box-shadow:0 2px 4px rgba(0,0,0,0.05)">
            <h4 style="margin:0 0 15px 0; color:#2C3E50;">{title}</h4>"""
        for param in params:
            value = data[param]
            config = self.generator.param_config[param]
            min_val, max_val = (config[0], config[1]) if len(config) == 4 else (None, None)
            unit = self._get_unit(param)

            # Format value safely
            try:
                formatted_value = f"{float(value):.2f}{unit}" if isinstance(value, (int, float)) else f"{value}{unit}"
            except:
                formatted_value = f"{value}{unit}"

            alert = " ⚠️ " if (min_val and max_val and not (min_val <= value <= max_val)) else ""

            html += f"""
            <div style="margin-bottom:12px;">
                <div style="display:flex; justify-content:space-between; align-items:center;">
                    <span style="color:#666; font-size:0.9em;">{param.replace('_', ' ').title()}</span>
                    <span style="color:#FF4B4B;">{alert}</span>
                </div>
                <div style="font-size:1.4em; color:#2C3E50;">
                    {formatted_value}
```

```
            </div>
            <div style="font-size:0.8em; color:#888;">
                Range: {min_val:.1f}-{max_val:.1f}{unit}
            </div>
        </div>"""
    return html + "</div>"

def _update_charts(self):
    with self.chart_container:
        st.subheader("📈 Real-Time Telemetry Trends (Last 10 Readings)")
        df = pd.DataFrame(st.session_state.history[-10:]).set_index('timestamp')
        fig = make_subplots(rows=2, cols=2, subplot_titles=(
            'Battery Voltage (V)', 'Internal Temperature (°C)',
            'Gyroscope (rad/s)', 'Data Quality (%)'))
        fig.add_trace(go.Scatter(x=df.index, y=df.battery_voltage, name='Battery', line=dict(color='#4285F4')), 1, 1)
        fig.add_trace(go.Scatter(x=df.index, y=df.internal_temp, name='Temp', line=dict(color='#DB4437')), 1, 2)
        fig.add_trace(go.Scatter(x=df.index, y=df.gyroscope, name='Gyro', line=dict(color='#0F9D58')), 2, 1)
        fig.add_trace(go.Scatter(x=df.index, y=df.data_quality, name='Quality', line=dict(color='#F4B400')), 2, 2)
        fig.update_layout(height=600, showlegend=False, margin=dict(l=40, r=40, t=80, b=40),
                          paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)')
        st.plotly_chart(fig, use_container_width=True)

def _get_unit(self, param):
    units = {
        "voltage": "V", "current": "A", "temp": "°C", "efficiency": "%",
        "consumption": "W", "velocity": "km/s", "rpm": "RPM",
        "rate": "Mbps", "loss": "%", "error": "bps", "orientation": "°"
    }
    return next((v for k, v in units.items() if k in param.lower()), "")
```

```
def main():
    generator = DataGenerator()
    detector = AnomalyDetector()
    alert = AlertSystem()
    dashboard = SatelliteDashboard(generator)

    if 'last_update' not in st.session_state:
        st.session_state.update({
            'last_update': datetime.min,
            'history': [],
            'data': None,
            'anomaly': False
        })

    if (datetime.now() - st.session_state.last_update).total_seconds() >= 2:
        try:
            new_data = generator.generate_data_point()
            anomaly = detector.predict(new_data)

            # Add anomaly status to data
            new_data['anomaly'] = int(anomaly)

            with DatabaseManager() as cursor:
                # Insert with anomaly column
                columns = list(new_data.keys())
                values = list(new_data.values())
                cursor.execute(
                    f"INSERT INTO {DB_CONFIG['table']} ({', '.join(columns)}) VALUES ({', '.join(['%s']*len(values))})",
                    values
                )
```

```
            # Keep only last 10 entries
            st.session_state.history = [new_data] + st.session_state.history[:9]

            st.session_state.update({
                'last_update': datetime.now(),
                'data': new_data,
                'anomaly': anomaly
            })

            if anomaly:
                alert.send_alert(new_data)

        except Exception as e:
            st.error(f"System Error: {str(e)}")

    if st.session_state.data:
        dashboard.update_display(st.session_state.data, st.session_state.anomaly)
    time.sleep(10)
    st.rerun()

if __name__ == "__main__":
    main()
```

Improvements:

- Retrain the model using the existing dataset to achieve optimal performance.
- Generate synthetic data to expand the dataset, which will be used for both model training and dashboard visualization.
- Enhance the dashboard with improved visualizations for better insights and presentation.
- Connect the dashboard to MySQL for real-time data access and updates.

Supporting Researches websites link:

- A Digital Twin-Based Approach for the Fault Diagnosis and Health Monitoring of a Complex Satellite System

- dasjaydeep2001/AI-Driven-Digital-Twin-for-Structural-Health-Monitoring-: This project aims to revolutionize structural health monitoring by leveraging Artificial Intelligence (AI), specifically Artificial Neural Networks (ANN), combined with Digital Twin technology. Our goal is to develop a fast-response surrogate model capable of providing real-time safety insights for critical infrastructure, such as bridges.
- data.nasa.gov/browse?sortBy=newest&pageSize=20&limitTo=datasets