

Day 4 Session 4:

Hands-On Mini Project: Telemetry Data Analysis Dashboard

This session is a hands-on mini-project designed to consolidate your learning from Day 4. You will apply your skills in NumPy, Pandas, and Matplotlib to build a small telemetry data analysis system. The goal is to simulate receiving telemetry data, process it to extract insights, and visualize key trends, moving towards the concept of a simple data dashboard.

1. Mini Project: Telemetry Data Analysis Dashboard

Problem Statement:

Imagine you are receiving daily telemetry logs from a satellite or a ground station. Your task is to develop a Python script that can:

1. Load raw telemetry data from a CSV file.
2. Clean and prepare the data for analysis (handle dates, missing values, correct data types).
3. Analyze the data to extract key insights (e.g., average sensor readings, error counts, daily trends).
4. Visualize the most important trends and summaries using plots.
5. Present these findings in a way that hints at a simple "dashboard" for quick understanding.

This project will guide you through a common data workflow in engineering and scientific fields.

Conceptual Approach:

We will break down the project into several manageable steps:

- Step 1: Simulate Telemetry Data: Create a realistic-looking CSV file as our input.
- Step 2: Load Data with Pandas: Read the CSV into a DataFrame.
- Step 3: Data Cleaning and Preprocessing: Make the data ready for analysis.
- Step 4: Data Analysis with Pandas: Calculate summaries and insights.

- Step 5: Data Visualization with Matplotlib: Create informative plots.

Let's begin!

2. Project Implementation Steps

We will use pandas for data loading, cleaning, and analysis, and matplotlib for visualization.

2.1 Step 1: Simulating Telemetry Data (Creating a Dummy CSV File)

Before we can analyze data, we need some! We'll create a simple CSV file that looks like telemetry data.

Why Simulate?

In real life, data comes from sensors or databases. For learning, creating a small, controlled file helps us focus on the Pandas and Matplotlib parts without needing actual hardware.

Python

```
import pandas as pd
import numpy as np
import os
import datetime

print("--- Step 1: Simulating Telemetry Data ---")

# Simulated telemetry content (CSV format as string)
# Each row is a sensor reading with timestamp, ID, value, unit, and status
telemetry_data_content = """Timestamp,Sensor_ID,Value,Unit,Status
2025-07-01 08:00:00,TEMP_ENGINE,85.2,C,OK
2025-07-01 08:00:00,PRES_FUEL,101.5,kPa,OK
2025-07-01 08:01:00,TEMP_ENGINE,85.5,C,OK
2025-07-01 08:01:00,FLOW_COOLANT,12.3,LPM,OK
2025-07-01 08:02:00,TEMP_ENGINE,85.1,C,WARNING
2025-07-01 08:02:00,PRES_FUEL,nan,kPa,ERROR
```

```

2025-07-01 08:03:00, FLOW_COOLANT, 12.5, LPM, OK
2025-07-01 08:03:00, HUMIDITY_CABIN, 60.1, %, OK
2025-07-01 08:04:00, TEMP_ENGINE, 86.0, C, OK
2025-07-01 08:04:00, PRES_FUEL, 101.7, kPa, OK
2025-07-02 08:00:00, TEMP_ENGINE, 87.0, C, OK
2025-07-02 08:00:00, PRES_FUEL, 102.0, kPa, OK
2025-07-02 08:01:00, TEMP_ENGINE, 87.3, C, ERROR
2025-07-02 08:01:00, FLOW_COOLANT, 12.8, LPM, OK
2025-07-02 08:02:00, PRES_FUEL, 102.5, kPa, OK
2025-07-02 08:02:00, HUMIDITY_CABIN, 62.5, %, OK
"""

# File name to save the CSV
file_name = "telemetry_data.csv"

# Write the CSV content to a file
with open(file_name, "w") as f:
    # Replace 'nan' as string with actual numpy NaN so pandas can recognize it
    f.write(telemetry_data_content.replace('nan', str(np.nan)))

print(f"✅ Dummy telemetry data saved to '{file_name}'.")

```

2.2 Step 2: Loading Data with Pandas (pd.read_csv())

Now, we'll load the CSV file we just created into a Pandas DataFrame.

`pd.read_csv()`: Read a CSV file into a DataFrame

- Purpose: The easiest way to load data from a CSV file.
- Syntax: `pd.read_csv(filepath_or_buffer, sep=',', header='infer', ...)`
 - `filepath_or_buffer`: The path to your CSV file.
 - `sep`: (Optional) The separator between columns (default is comma ,).

Python

```
print("\n--- Step 2: Loading Data with Pandas ---")

# Try loading the CSV file using pandas
try:
    df_raw_telemetry = pd.read_csv(file_name)
    print(f"✅ File '{file_name}' loaded successfully into a DataFrame.")
except FileNotFoundError:
    print(f"❌ Error: '{file_name}' not found. Make sure it's in the current folder.")
    # Fall back to an empty DataFrame for demo (not recommended in production)
    df_raw_telemetry = pd.DataFrame()

# --- Inspect the loaded data ---

# Display the first 5 rows
print("\n🔍 First 5 rows of data:")
print(df_raw_telemetry.head())

# Show column types and non-null counts
print("\n📊 DataFrame Info:")
df_raw_telemetry.info()

# Show how many missing values each column has
print("\n🔍 Missing Values Count per Column:")
print(df_raw_telemetry.isnull().sum())
```

2.3 Step 3: Data Cleaning and Preprocessing

Real-world data is rarely perfect. We need to clean it to make it useful.

Why Clean Data?

- **Correct Types:** Ensure numbers are stored as numbers, and dates as dates.
- **Missing Data:** Decide how to handle NaN values (fill them, or remove rows/columns).
- **Consistency:** Make sure similar data (like units or statuses) are written the same way.

Python

```
print("\n--- Step 3: Data Cleaning and Preprocessing ---")

# Make a copy so original data is preserved
df_telemetry_cleaned = df_raw_telemetry.copy()

# Step 3.1: Convert 'Timestamp' column to datetime
df_telemetry_cleaned['Timestamp'] =
pd.to_datetime(df_telemetry_cleaned['Timestamp'])
print("✅ Step 3.1: 'Timestamp' column converted to datetime.")
print("    > New type:", df_telemetry_cleaned['Timestamp'].dtype)

# Step 3.2: Set 'Timestamp' as index
df_telemetry_cleaned.set_index('Timestamp', inplace=True)
print("\n✅ Step 3.2: 'Timestamp' set as the DataFrame index.")
print("    > Here's how the top rows look now:")
print(df_telemetry_cleaned.head())

# Step 3.3: Convert 'Value' to numeric (handle missing or invalid entries)
df_telemetry_cleaned['Value'] = pd.to_numeric(df_telemetry_cleaned['Value'],
errors='coerce')
print("\n✅ Step 3.3: 'Value' column converted to numeric (non-numeric replaced
with NaN).")

# Step 3.4: Fill missing values in 'Value' with the column's average
mean_value = df_telemetry_cleaned['Value'].mean()
df_telemetry_cleaned['Value'].fillna(mean_value, inplace=True)
print(f"\n✅ Step 3.4: Missing values in 'Value' filled with average:
{mean_value:.2f}")

# Step 3.5: Convert 'Status' and 'Unit' to categorical (memory-efficient for
repeated text)
df_telemetry_cleaned['Status'] =
df_telemetry_cleaned['Status'].astype('category')
df_telemetry_cleaned['Unit'] = df_telemetry_cleaned['Unit'].astype('category')
print("\n✅ Step 3.5: 'Status' and 'Unit' converted to categorical type.")

# Summary of the cleaned data
print("\n📋 Cleaned DataFrame Overview:")
df_telemetry_cleaned.info()

# Double-check if any missing values still exist
print("\n🔍 Remaining Missing Values:")
```

```
print(df_telemetry_cleaned.isnull().sum())
```

2.4 Step 4: Data Analysis with Pandas

Now that our data is clean, we can start asking questions and getting insights!

Python

```
print("\n--- Step 4: Data Analysis with Pandas ---")

# ♦ 1. Summary statistics for all sensor values
overall_stats = df_telemetry_cleaned['Value'].describe()
print("📌 1. Summary Statistics for 'Value' column:")
print(overall_stats.round(2), "\n")

# ♦ 2. Average value recorded by each sensor
avg_value_per_sensor = df_telemetry_cleaned.groupby('Sensor_ID')['Value'].mean()
print("📌 2. Average Value per Sensor ID:")
print(avg_value_per_sensor.round(2), "\n")

# ♦ 3. Count of each status type (OK, WARNING, ERROR)
status_counts = df_telemetry_cleaned['Status'].value_counts()
print("📌 3. Count of Each Status Type:")
print(status_counts, "\n")

# ♦ 4. Daily average of temperature (TEMP_ENGINE only)
df_temp_engine = df_telemetry_cleaned[df_telemetry_cleaned['Sensor_ID'] == 'TEMP_ENGINE']
daily_avg_temp = df_temp_engine['Value'].resample('D').mean()
print("📌 4. Daily Average Temperature (TEMP_ENGINE):")
print(daily_avg_temp.round(2), "\n")

# ♦ 5. Maximum pressure recorded (PRES_FUEL only)
df_pres_fuel = df_telemetry_cleaned[df_telemetry_cleaned['Sensor_ID'] == 'PRES_FUEL']
max_pressure = df_pres_fuel['Value'].max()
```

```
print(f"📌 5. Maximum Pressure Recorded (PRES_FUEL): {max_pressure:.2f} kPa\n")
```

2.5 Step 5: Data Visualization with Matplotlib

Time to create some plots to make our insights easy to see!

Why Visualize?

- Quick Understanding: See trends and patterns much faster than looking at numbers.
- Communication: Share your findings clearly with others.
- Discovery: Spot things you might miss in raw data.

Python

```
import matplotlib.pyplot as plt

print("\n--- Step 5: Data Visualization with Matplotlib ---")

# ♦ 5.1: Plot overall sensor value trends
plt.figure(figsize=(10, 5))
plt.plot(
    df_telemetry_cleaned.index,
    df_telemetry_cleaned['Value'],
    marker='o',
    linestyle='-',
    markersize=4,
    alpha=0.7
)
plt.title("Overall Sensor Values Over Time")
plt.xlabel("Timestamp")
plt.ylabel("Sensor Value")
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.grid(True)
plt.tight_layout()
```

```

plt.show()

# ♦ 5.2: Bar chart of sensor status distribution
plt.figure(figsize=(8, 5))
status_counts.plot(
    kind='bar',
    color=['lightgreen', 'gold', 'lightcoral'] # OK, WARNING, ERROR
)
plt.title("Distribution of Sensor Statuses")
plt.xlabel("Status")
plt.ylabel("Count")
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# ♦ 5.3: Plot daily average engine temperature
plt.figure(figsize=(10, 5))
plt.plot(
    daily_avg_temp.index,
    daily_avg_temp.values,
    marker='o',
    linestyle='-',
    color='blue'
)
plt.title("Daily Average Temperature (TEMP_ENGINE)")
plt.xlabel("Date")
plt.ylabel("Average Temp (°C)")
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

print("\n✅ Visualization complete. Plots displayed above.")

# ♦ Clean up CSV (optional for demo purposes)
if os.path.exists(file_name):
    os.remove(file_name)
    print(f"🗑️ Clean-up: '{file_name}' removed from your directory.")

```


3. Dashboard Concept for Telemetry Data

In this mini-project, we've produced several key analyses and visualizations. Imagine if all these were available in one interactive screen, constantly updating, with filters and options to drill down. That's the essence of a Dashboard.

A dashboard for telemetry data would typically:

- Show key metrics at a glance (e.g., overall average temperature, max pressure).
- Display trends over time (like our overall value plot, or daily temperature).
- Provide summaries of status (like our status counts bar chart).
- Allow interaction (e.g., select a specific sensor, choose a date range, toggle different metrics).

Our plots and printed summaries form a very basic, static version of such a dashboard. The next step is to make it interactive and accessible via a web browser.

4. Prospect to Streamlit (Building Interactive Web Apps from Python)

Building interactive web dashboards usually requires knowledge of web technologies like HTML, CSS, and JavaScript. However, for Python users, tools like Streamlit offer a fantastic shortcut!

What is Streamlit?

Streamlit is an open-source Python library that allows you to create beautiful, interactive web applications for data science and machine learning with pure Python. You don't need any web development experience.

Why is Streamlit useful for Dashboards?

- Pure Python: Write your data analysis and visualization code, and Streamlit turns it into a web app.

- Interactive Widgets: Easily add sliders, buttons, dropdowns, text inputs, etc., to your app.
- Quick Deployment: Share your dashboards easily.
- Focus on Data: You focus on the data and insights, not on web design.

Conceptual Example of Streamlit Code:

If you had a Streamlit app, your code might look something like this in a file named `my_dashboard_app.py`:

```
Python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime

# Sample Data Setup (Replace this with real CSV load in production)
@st.cache_data
def load_telemetry_data():
    data = {
        'Timestamp': pd.date_range(start='2025-07-01 08:00:00', periods=100,
freq='H'),
        'Sensor_ID': np.random.choice(['TEMP_ENGINE', 'PRES_FUEL',
'FLOW_COOLANT'], 100),
        'Value': np.random.normal(100, 10, 100),
        'Unit': np.random.choice(['C', 'kPa', 'LPM'], 100),
        'Status': np.random.choice(['OK', 'WARNING', 'ERROR'], 100)
    }
    df = pd.DataFrame(data)
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])
    df.set_index('Timestamp', inplace=True)
    return df

# Load data
df_telemetry_cleaned = load_telemetry_data()

# Streamlit layout
st.set_page_config(layout="wide")
```

```

st.title("📊 Telemetry Dashboard (Powered by Streamlit)")

# Sidebar
st.sidebar.header("🔍 Filter Options")
sensor_ids = df_telemetry_cleaned['Sensor_ID'].unique()
selected_sensor = st.sidebar.selectbox("Select Sensor ID", sensor_ids)

min_date = df_telemetry_cleaned.index.min().date()
max_date = df_telemetry_cleaned.index.max().date()
date_range = st.sidebar.date_input("Select Date Range", (min_date, max_date),
min_value=min_date, max_value=max_date)

# Filter data
start_date, end_date = pd.to_datetime(date_range[0]),
pd.to_datetime(date_range[1])
filtered_df = df_telemetry_cleaned[
    (df_telemetry_cleaned['Sensor_ID'] == selected_sensor) &
    (df_telemetry_cleaned.index.date >= start_date.date()) &
    (df_telemetry_cleaned.index.date <= end_date.date())
]

# Show filtered data
st.subheader(f"📄 Data for {selected_sensor}")
st.dataframe(filtered_df)

# Plot sensor value trend
st.subheader("📈 Sensor Value Trend")
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(filtered_df.index, filtered_df['Value'], marker='o', linestyle='--',
color='teal')
ax.set_title(f"Trend of {selected_sensor} Readings")
ax.set_xlabel("Timestamp")
ax.set_ylabel("Sensor Value")
ax.grid(True)
st.pyplot(fig)

# Status breakdown
st.subheader("📊 Sensor Status Counts")
status_counts = filtered_df['Status'].value_counts()

fig2, ax2 = plt.subplots(figsize=(6, 3))
status_counts.plot(kind='bar', color=['green', 'orange', 'red'], ax=ax2)

```

```
ax2.set_title("Status Breakdown")
ax2.set_ylabel("Count")
st.pyplot(fig2)
```

How to run a Streamlit App (after installation):

1. Save your Python code (e.g., the conceptual code above) into a file named `my_dashboard_app.py`.
2. Open your terminal or Anaconda Prompt.
3. Navigate to the directory where you saved the file.
4. Run the command: `streamlit run my_dashboard_app.py`
5. Streamlit will open the app in your web browser.

Installation: `pip install streamlit`

5. Key Takeaways

This hands-on mini-project provided a practical application of the numerical and data analysis skills you've gained in Day 4.




- **Integrated Workflow:** Successfully performed a complete data analysis workflow: simulating data, loading, cleaning, analyzing, and visualizing, using NumPy, Pandas, and Matplotlib together.
- **Practical Data Cleaning:** Reinforced the importance and methods for cleaning real-world data (timestamps, missing values, data types) to ensure accurate analysis.
- **Insight Extraction:** Applied Pandas' powerful tools like `groupby()`, `mean()`, `max()`, `value_counts()`, and `resample()` to extract meaningful insights from telemetry data.
- **Effective Visualization:** Created informative plots (line charts, bar charts) using Matplotlib to visually present data trends and summaries, crucial for reports and

quick understanding.

- Dashboard Concept: Understood the basic idea of a data dashboard for consolidated insights.
- Prospect to Streamlit: Gained awareness of Streamlit as a valuable tool for quickly turning Python scripts into interactive web applications and dashboards, bridging the gap between analysis and deployment.
-

By completing this project, you've taken a significant step from learning individual tools to applying them in a cohesive manner to solve a real-world engineering data problem.

ISRO URSC – Python Training | Analog Data | Rajath Kumar

 For queries: rajath@analogdata.ai |  [\(+91\) 96633 53992](tel:+919663353992) |  <https://analogdata.ai>

This material is part of the ISRO URSC Python Training Program conducted by Analog Data (June 2025). For educational use only. © 2025 Analog Data.
