In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(12, 8))

    # Plot main parameter data
    plt.subplot(2, 1, 1)
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data', color='orange')
    plt.title('Main Parameter: {}'.format(selected_parameter))
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

    # Analyze anomalies
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]
    threshold = 0.1  # Adjust as needed
    anomalies = non_nominal_data[np.abs(difference) > threshold]
    num_anomalies = len(anomalies)

    # Identify related parameters
    related_parameters = dependencies.get(selected_parameter, [])

    # Plot related parameters data
    if related_parameters:
        plt.subplot(2, 1, 2)
        for related_param in related_parameters:
            plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.title('Related Parameters')
        plt.xlabel('Time')
        plt.ylabel('Parameter Value')
        plt.legend()
```

```
    # Display probable root causes
    output_text = "Number of anomalies detected for parameter {}: {}\n".format(selected_parameter, num_anomalies)
    output_text += "\nProbable Root Causes:\n"

    for anomaly_timestamp, _ in anomalies.iterrows():
        related_anomalies = []
        for related_param in related_parameters:
            if related_param in non_nominal_data.columns:
                related_diff = non_nominal_data[related_param] - nominal_data[related_param]
                related_anomaly = non_nominal_data[(np.abs(related_diff) > threshold) & (non_nominal_data.index == anc
                if not related_anomaly.empty:
                    related_anomalies.append(related_param)

        if related_anomalies:
            output_text += "Anomaly detected at timestamp {}. Possible root causes: {}\n".format(anomaly_timestamp, ",
        else:
            output_text += "Anomaly detected at timestamp {}. No related anomalies found.\n".format(anomaly_timestamp)

    window['-OUTPUT-'].update(output_text)

    # Display the plot
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
    window['-PLOT-'].update(data=img_data.read())
    plt.close()

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-')],
    [sg.Output(size=(60, 20), key='-OUTPUT-')]
]

# Load dependencies from file
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}
```

```
# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot main parameter data
    plt.subplot(len(dependencies) + 1, 1, 1)
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label=selected_parameter)
    plt.title(selected_parameter)

    # Plot related parameters data
    for i, related_param in enumerate(dependencies[selected_parameter], start=2):
        plt.subplot(len(dependencies) + 1, 1, i)
        plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.title('Dependency: {}'.format(related_param))

    plt.tight_layout()
    plt.legend()
    plt.show()

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
]
```

```
# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:

```
import pandas as pd
import PySimpleGUI as sg
import matplotlib.pyplot as plt

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, dependencies):
    plt.figure(figsize=(8, 6))

    # Plot main parameter data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label=selected_parameter)
    plt.title(selected_parameter)
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Plot related parameters data
    for related_param in dependencies[selected_parameter]:
        plt.figure(figsize=(8, 6))
        plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.title('Dependency: {}'.format(related_param))
        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.legend()
        plt.grid(True)
        plt.show()

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
```

```
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event Loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, dependencies)

window.close()
```

In [ ]:
```
import pandas as pd
import PySimpleGUI as sg
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, dependencies, window):
    plt.figure(figsize=(8, 6))

    # Plot main parameter data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label=selected_parameter)
    plt.title(selected_parameter)
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True)

    # Plot related parameters data
    for related_param in dependencies[selected_parameter]:
        plt.figure(figsize=(8, 6))
        plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.title('Dependency: {}'.format(related_param))
        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.legend()
        plt.grid(True)

    # Convert plots to PySimpleGUI compatible format
    canvas = FigureCanvasTkAgg(plt.gcf(), master=window)
    canvas.draw()
    canvas.get_tk_widget().pack(side='top', fill='both', expand=1)
```

```
# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, dependencies, window)

window.close()
```

In [ ]:

```
import pandas as pd
import PySimpleGUI as sg
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.ensemble import IsolationForest

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Anomaly detection function
def detect_anomalies(data):
    clf = IsolationForest(contamination=0.05, random_state=42)
    clf.fit(data)
    predictions = clf.predict(data)
    anomalies = data[predictions == -1]
    return anomalies

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, dependencies, window):
    layout = []

    # Plot main parameter data (nominal vs. non-nominal)
    plt.figure(figsize=(12, 6))
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal')
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal', color='red')
    plt.title(selected_parameter)
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    fig_photo = plt_to_img(plt)

    # Add main parameter plot to layout
```

```
    layout.append([sg.Image(data=fig_photo)])

    # Plot related parameters data
    for related_param in dependencies[selected_parameter]:
        plt.figure(figsize=(12, 6))
        plt.plot(nominal_data.index, nominal_data[related_param], label='Nominal')
        plt.plot(non_nominal_data.index, non_nominal_data[related_param], label='Non-Nominal', color='red')
        plt.title('Dependency: {}'.format(related_param))
        plt.xlabel('Time')
        plt.ylabel('Value')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        fig_photo = plt_to_img(plt)

        # Add related parameter plot to layout
        layout.append([sg.Image(data=fig_photo)])

    # Detect anomalies for the selected parameter
    anomalies = detect_anomalies(nominal_data[[selected_parameter]])
    if not anomalies.empty:
        layout.append([sg.Text(f'Anomalies detected in {selected_parameter}:')])
        layout.append([sg.Table(values=anomalies.values.tolist(), headings=anomalies.columns.tolist(),
                                auto_size_columns=True, display_row_numbers=False, justification='center')])

    # Display plots and anomalies in PySimpleGUI window
    window['-PLOT-'].update(layout)

# Function to convert Matplotlib plot to bytes
def plt_to_img(plt):
    img_data = plt.gcf().canvas.print_png()  # Convert Matplotlib plot to PNG image data
    return img_data

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Column(layout=[], size=(1200, 600), scrollable=True, key='-PLOT-')],
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)
```

```
# Event Loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, dependencies, window)

window.close()
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io
from PIL import Image

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies between parameters
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
```

```
plt.xlabel('Time')
plt.ylabel('Parameter Value')
plt.legend()

# Plot related parameters data
num_related_params = len(dependencies[selected_parameter])
num_subplots = num_related_params + 1  # Main plot + related parameters
for i, related_param in enumerate(dependencies[selected_parameter], start=2):
    plt.subplot(num_subplots, 1, i)
    plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
    plt.title('Dependency: {}'.format(related_param))
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

plt.tight_layout()

# Save plot as an image
img_data = io.BytesIO()
plt.savefig(img_data, format='png')
img_data.seek(0)
window['-PLOT-'].update(data=img_data.read())
plt.close()

# Print the number of values deviating beyond the threshold
num_anomalies = len(anomalies)
if num_anomalies == 0:
    window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
else:
    output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

    # Sort the anomalies by deviation
    anomalies_sorted = anomalies.copy()
    anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
    anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

    # Print the details of the time stamps where anomalies were detected
    output_text += "\nDetails of anomaly timestamps:\n"
    for timestamp, row in anomalies_sorted.iterrows():
        output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

    window['-OUTPUT-'].update(output_text)
```

```
# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-')],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io
from PIL import Image

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies between parameters
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
```

```
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(800, 600))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event Loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
```

```
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies between parameters
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
```

```
    plt.ylabel('Parameter Value')
    plt.legend()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)

    # Plot related parameters
    for i, related_param in enumerate(dependencies[selected_parameter], start=2):
        plt.subplot(len(dependencies) + 1, 1, i)
        plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.plot(non_nominal_data.index, non_nominal_data[related_param], label=related_param + ' (Non-Nominal)', line
        plt.title('Dependency: {}'.format(related_param))
        plt.xlabel('Time')
        plt.ylabel('Parameter Value')
        plt.legend()

    # Convert plot to an image
    img_data = plt_to_img(plt)
    # Update the plot image in the window
    window['-PLOT-'].update(data=img_data)

def plt_to_img(plt):
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
    return img_data.read()
```

```
# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(800, 600))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies between parameters
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
```

```
plt.ylabel('Parameter Value')
plt.legend()

# Print the number of values deviating beyond the threshold
num_anomalies = len(anomalies)
if num_anomalies == 0:
    window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
else:
    output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

    # Sort the anomalies by deviation
    anomalies_sorted = anomalies.copy()
    anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
    anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

    # Print the details of the time stamps where anomalies were detected
    output_text += "\nDetails of anomaly timestamps:\n"
    for timestamp, row in anomalies_sorted.iterrows():
        output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

    window['-OUTPUT-'].update(output_text)

# Plot related parameters
num_related_params = len(dependencies[selected_parameter])
fig, axs = plt.subplots(num_related_params, 1, figsize=(10, num_related_params * 4))
if num_related_params == 1:
    axs = [axs]
for ax, related_param in zip(axs, dependencies[selected_parameter]):
    ax.plot(nominal_data.index, nominal_data[related_param], label='Nominal Data')
    ax.plot(non_nominal_data.index, non_nominal_data[related_param], label='Non-Nominal Data', linestyle='--')
    ax.set_title('Dependency: {}'.format(related_param))
    ax.set_xlabel('Time')
    ax.set_ylabel('Parameter Value')
    ax.legend()

# Save plot as an image
img_data = plt_to_img(fig)
plt.close(fig)

# Update the plot image in the window
window['-PLOT-'].update(data=img_data)
```

```
def plt_to_img(fig):
    img_data = io.BytesIO()
    fig.savefig(img_data, format='png')
    img_data.seek(0)
    return img_data.read()

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(800, 600))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies between parameters
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window, dependencies):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
```

```
plt.ylabel('Parameter Value')
plt.legend()

# Print the number of values deviating beyond the threshold
num_anomalies = len(anomalies)
if num_anomalies == 0:
    window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
else:
    output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

    # Sort the anomalies by deviation
    anomalies_sorted = anomalies.copy()
    anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
    anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

    # Print the details of the time stamps where anomalies were detected
    output_text += "\nDetails of anomaly timestamps:\n"
    for timestamp, row in anomalies_sorted.iterrows():
        output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

    window['-OUTPUT-'].update(output_text)

# Plot related parameters
num_related_params = len(dependencies[selected_parameter])
fig, axs = plt.subplots(num_related_params, 1, figsize=(10, num_related_params * 4))
if num_related_params == 1:
    axs = [axs]
for ax, related_param in zip(axs, dependencies[selected_parameter]):
    ax.plot(nominal_data.index, nominal_data[related_param], label='Nominal Data')
    ax.plot(non_nominal_data.index, non_nominal_data[related_param], label='Non-Nominal Data', linestyle='--')
    ax.set_title('Dependency: {}'.format(related_param))
    ax.set_xlabel('Time')
    ax.set_ylabel('Parameter Value')
    ax.legend()

# Save plot as an image
img_data = plt_to_img(fig)
plt.close(fig)

# Update the plot image in the window
window['-PLOT-'].update(data=img_data)
```

```
def plt_to_img(fig):
    img_data = io.BytesIO()
    fig.savefig(img_data, format='png')
    img_data.seek(0)
    return img_data.read()

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Column(layout=[], size=(800, 600), scrollable=True)],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window, dependencies)

window.close()
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

    # Save plot as an image
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
```

```
    window['-PLOT-'].update(data=img_data.read())
    plt.close()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-')],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window)
```

```
window.close()
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                           'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
```

```
    plt.ylabel('Parameter Value')
    plt.legend()

    # Save plot as an image
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
    window['-PLOT-'].update(data=img_data.read())
    plt.close()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-')],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True)

# Event loop
while True:
```

```
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window)

window.close()
```

In [ ]:
```
port pandas as pd
port numpy as np
port matplotlib.pyplot as plt
port PySimpleGUI as sg
port io

Load CSV files for nominal and non-nominal data
minal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
n_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

Define dependencies
pendencies = {
  'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - NA']
  'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust',
                                         'LanderYawWRTVertical', 'LanderRollWRTVertical', 'LanderPitchWRTVertical']


Function to plot data and detect anomalies
f plot_selected_parameter(selected_parameter, window):
  plt.figure(figsize=(10, 6))

  # Plot main parameter
  plt.subplot(len(dependencies) + 1, 1, 1)
  plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

  # Calculate the difference between non-nominal and nominal data
  difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

  # Define a threshold for anomaly detection
  threshold = 0.1  # Adjust as needed

  # Detect anomalies
  anomalies = non_nominal_data[np.abs(difference) > threshold]

  # Plot non-nominal data
  plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

  # Plot anomalies
  plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

  plt.title('Sensor Data Plot: {}'.format(selected_parameter))
```

```
plt.xlabel('Time')
plt.ylabel('Parameter Value')
plt.legend()

# Plot related parameters
for i, related_param in enumerate(dependencies.get(selected_parameter, []), start=2):
    plt.subplot(len(dependencies) + 1, 1, i)
    plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
    plt.plot(non_nominal_data.index, non_nominal_data[related_param], label=related_param + ' (Non-Nominal)', linesty
    plt.title('Dependency: {}'.format(related_param))
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

plt.tight_layout()

# Save plot as an image
img_data = io.BytesIO()
plt.savefig(img_data, format='png')
img_data.seek(0)
window['-PLOT-'].update(data=img_data.read())
plt.close()

# Print the number of values deviating beyond the threshold
num_anomalies = len(anomalies)
if num_anomalies == 0:
    window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
else:
    output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

    # Sort the anomalies by deviation
    anomalies_sorted = anomalies.copy()
    anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
    anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

    # Print the details of the time stamps where anomalies were detected
    output_text += "\nDetails of anomaly timestamps:\n"
    for timestamp, row in anomalies_sorted.iterrows():
        output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_para

    window['-OUTPUT-'].update(output_text)
```

```
Define the layout for PySimpleGUI
yout = [
  [sg.Text("Select Parameter:")],
  [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
  [sg.Image(key='-PLOT-')],
  [sg.Output(size=(60, 10), key='-OUTPUT-')]


Create the PySimpleGUI window
ndow = sg.Window("Anomaly Detection System", layout, finalize=True)

Event loop
ile True:
  event, values = window.read()
  if event == sg.WINDOW_CLOSED:
      break
  elif event == '-PARAMETER-':
      selected_param = values['-PARAMETER-']
      if selected_param:
          plot_selected_parameter(selected_param, window)

ndow.close()
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io
from PIL import Image

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window):
    plt.figure(figsize=(10, 6))

    # Plot nominal data
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies in the main parameter
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    # Check for anomalies in related parameters
    related_params = dependencies.get(selected_parameter, [])
```

```
    for related_param in related_params:
        # Detect anomalies in related parameters
        difference_related = non_nominal_data[related_param] - nominal_data[related_param]
        anomalies_related = non_nominal_data[np.abs(difference_related) > threshold]

        if not anomalies_related.empty:
            plt.scatter(anomalies_related.index, anomalies_related[related_param], color='blue', label='Anomalies in R

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

    # Save plot as an image
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
    window['-PLOT-'].update(data=img_data.read())
    plt.close()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
```

```
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(400, 400))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True, resizable=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window)

window.close()
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io
from PIL import Image

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro1/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro1/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window):
    plt.figure(figsize=(12, 8))

    # Plot selected parameter
    plt.subplot(2, 1, 1)
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies in the selected parameter
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')

    plt.title('Sensor Data Plot: {}'.format(selected_parameter))
```

```
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

    # Plot related parameters
    related_params = dependencies.get(selected_parameter, [])
    if related_params:
        plt.subplot(2, 1, 2)
        for related_param in related_params:
            plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
            plt.plot(non_nominal_data.index, non_nominal_data[related_param], label=related_param + ' (Non-Nominal)',

        plt.title('Related Parameters')
        plt.xlabel('Time')
        plt.ylabel('Parameter Value')
        plt.legend()

    # Save plot as an image
    img_data = io.BytesIO()
    plt.savefig(img_data, format='png')
    img_data.seek(0)
    window['-PLOT-'].update(data=img_data.read())
    plt.close()

    # Print the number of values deviating beyond the threshold
    num_anomalies = len(anomalies)
    if num_anomalies == 0:
        window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
    else:
        output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

        # Sort the anomalies by deviation
        anomalies_sorted = anomalies.copy()
        anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
        anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

        # Print the details of the time stamps where anomalies were detected
        output_text += "\nDetails of anomaly timestamps:\n"
        for timestamp, row in anomalies_sorted.iterrows():
            output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa

        window['-OUTPUT-'].update(output_text)
```

```
# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(800, 600))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True, resizable=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window)

window.close()
```

In [*]:
```
#wORKINGGGGGGGGGGGGGG

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
import io
from PIL import Image

# Load CSV files for nominal and non-nominal data
nominal_data = pd.read_csv("D:/Isro project/NOMINAL.csv", index_col=0, parse_dates=True)
non_nominal_data = pd.read_csv("D:/Isro project/AE01.csv", index_col=0, parse_dates=True)

# Define dependencies
dependencies = {
    'mass-dynamics': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'CentalEngine-Thrust - N
    'LanderHeightFromSurface (With DEM)': ['Engine1-Thrust', 'Engine2-Thrust', 'Engine3-Thrust', 'Engine4-Thrust', 'La
}

# Function to plot data and detect anomalies
def plot_selected_parameter(selected_parameter, window):
    plt.figure(figsize=(12, 8))

    # Plot selected parameter
    plt.subplot(2, 2, 1)
    plt.plot(nominal_data.index, nominal_data[selected_parameter], label='Nominal Data')

    # Calculate the difference between non-nominal and nominal data
    difference = non_nominal_data[selected_parameter] - nominal_data[selected_parameter]

    # Define a threshold for anomaly detection
    threshold = 0.1  # Adjust as needed

    # Detect anomalies in the selected parameter
    anomalies = non_nominal_data[np.abs(difference) > threshold]

    # Plot non-nominal data
    plt.plot(non_nominal_data.index, non_nominal_data[selected_parameter], label='Non-Nominal Data')

    # Plot anomalies
    plt.scatter(anomalies.index, anomalies[selected_parameter], color='red', label='Anomalies')
```

```
plt.title('Sensor Data Plot: {}'.format(selected_parameter))
plt.xlabel('Time')
plt.ylabel('Parameter Value')
plt.legend()

# Plot related parameters
related_params = dependencies.get(selected_parameter, [])
if related_params:
    plt.subplot(2, 2, 2)
    for related_param in related_params:
        plt.plot(nominal_data.index, nominal_data[related_param], label=related_param)
        plt.plot(non_nominal_data.index, non_nominal_data[related_param], label=related_param + ' (Non-Nominal)',

    plt.title('Related Parameters')
    plt.xlabel('Time')
    plt.ylabel('Parameter Value')
    plt.legend()

# Save plot as an image
img_data = io.BytesIO()
plt.savefig(img_data, format='png')
img_data.seek(0)
window['-PLOT-'].update(data=img_data.read())
plt.close()

# Print the number of values deviating beyond the threshold
num_anomalies = len(anomalies)
if num_anomalies == 0:
    window['-OUTPUT-'].update("No anomalies detected for parameter: {}".format(selected_parameter))
else:
    output_text = "Number of values deviating beyond the threshold: {}\n".format(num_anomalies)

    # Sort the anomalies by deviation
    anomalies_sorted = anomalies.copy()
    anomalies_sorted['Deviation'] = np.abs(difference[anomalies.index])
    anomalies_sorted = anomalies_sorted.sort_values(by='Deviation', ascending=False)

    # Print the details of the time stamps where anomalies were detected
    output_text += "\nDetails of anomaly timestamps:\n"
    for timestamp, row in anomalies_sorted.iterrows():
        output_text += "Timestamp: {}, Actual Value: {}, Deviation: {} points\n".format(timestamp, row[selected_pa
```

```
        window['-OUTPUT-'].update(output_text)

# Define the layout for PySimpleGUI
layout = [
    [sg.Text("Select Parameter:")],
    [sg.Combo(values=list(nominal_data.columns), key='-PARAMETER-', size=(30, 1), enable_events=True)],
    [sg.Image(key='-PLOT-', size=(800, 600)), sg.Multiline('', key='-DATA-', size=(40, 10))],
    [sg.Output(size=(60, 10), key='-OUTPUT-')]
]

# Create the PySimpleGUI window
window = sg.Window("Anomaly Detection System", layout, finalize=True, resizable=True)

# Event loop
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    elif event == '-PARAMETER-':
        selected_param = values['-PARAMETER-']
        if selected_param:
            plot_selected_parameter(selected_param, window)

window.close()
```

In [ ]: