The core steps can be broken down into three main sections:

Obtaining Coordinates and Geospatial Data (using QGIS or Python Libraries) Image Classification and Preprocessing (Object-Based Image Analysis) Machine Learning Model (CNN for anomaly detection)

```
pip install geopandas rasterio gdal matplotlib
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.11/dist-packages (1.0.1)
Collecting rasterio
  Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Requirement already satisfied: gdal in /usr/local/lib/python3.11/dist-packages (3.6.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.7)
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages (from rasterio) (25.3.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from rasterio) (2025.1.31)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.11/dist-packages (from rasterio) (8.1.8)
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages (from rasterio) (3.2.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.2/22.2 MB 15.6 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: cligj, click-plugins, affine, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2 rasterio-1.4.3
```

```
import rasterio
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
```

```
from google.colab import drive
# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

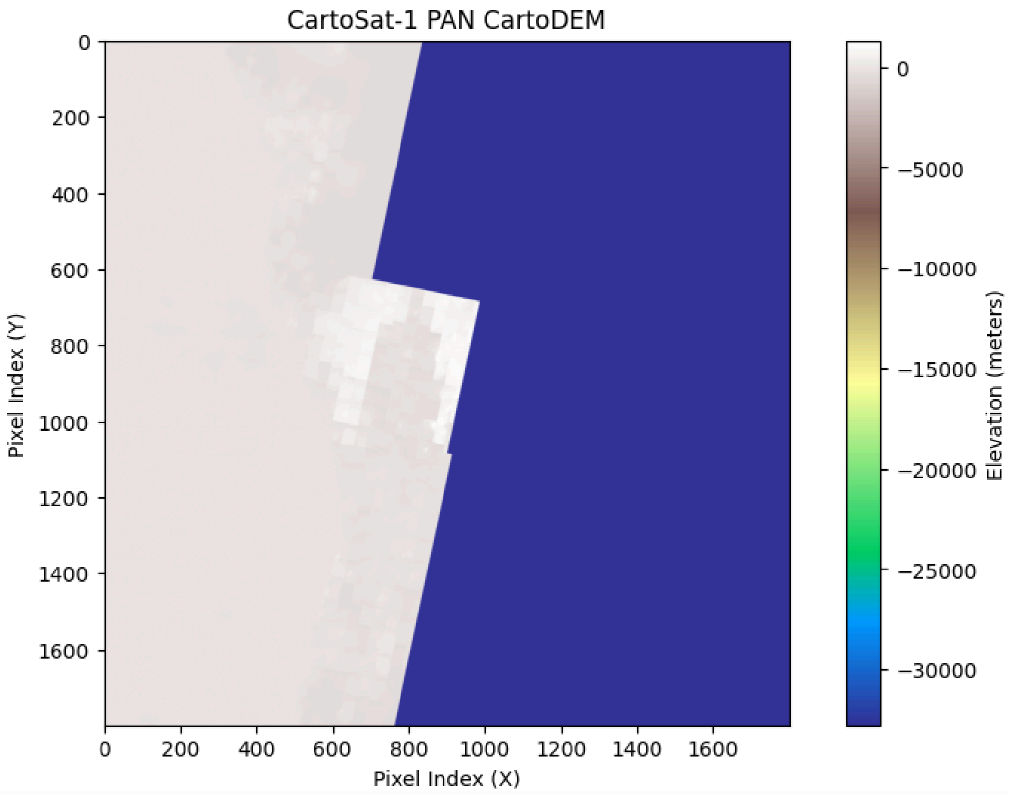
Loading the DEM Load your CartoSat-1 DEM:

```
# Path to the CartoSat-1 DEM file
cartoDEM_path = "/content/drive/MyDrive/Project Bhuvan/PAN_CD_DEM.tif"

# Open the DEM file
with rasterio.open(cartoDEM_path) as dem_data:
    elevation = dem_data.read(1)  # Read the first band (elevation data)
    transform = dem_data.transform
    print(f"DEM loaded with shape: {elevation.shape}")
```

```
DEM loaded with shape: (1800, 1800)
```

Visualizing the DEM

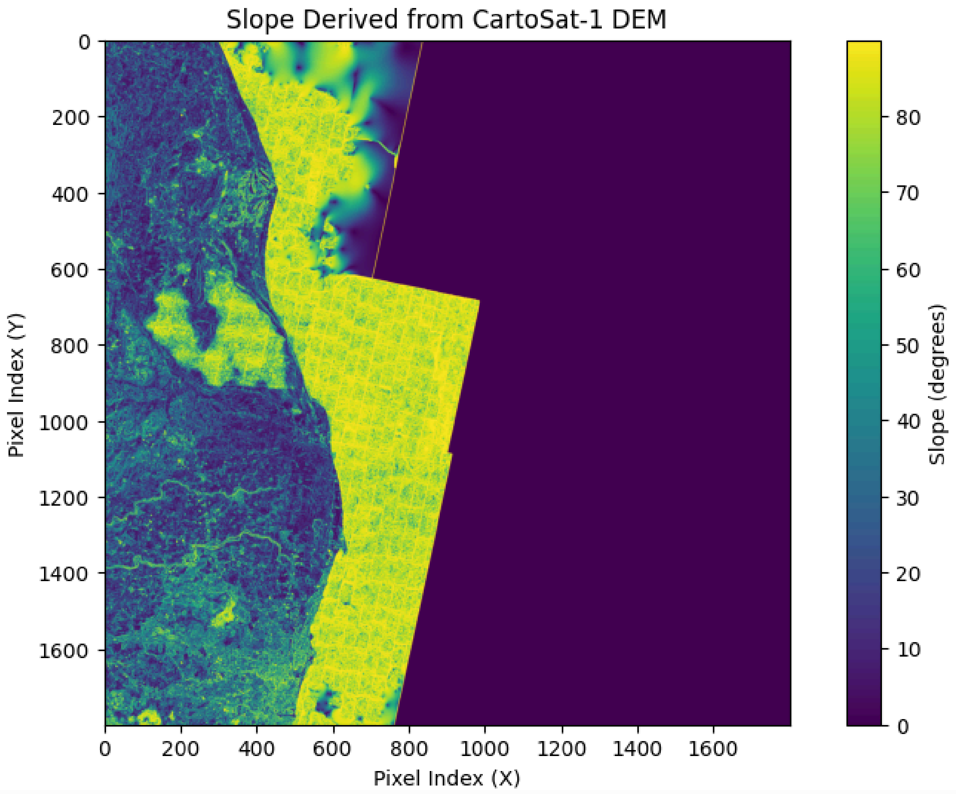
```
# Plot the DEM
plt.figure(figsize=(10, 6))
plt.imshow(elevation, cmap='terrain')
plt.colorbar(label='Elevation (meters)')
plt.title('CartoSat-1 PAN CartoDEM')
plt.xlabel('Pixel Index (X)')
plt.ylabel('Pixel Index (Y)')
plt.show()
```



Calculating Slope

```
# Calculate the slope
def calculate_slope(dem):
    x_gradient = np.gradient(dem, axis=1)
    y_gradient = np.gradient(dem, axis=0)
    slope = np.arctan(np.sqrt(x_gradient**2 + y_gradient**2)) * (180/np.pi)
    return slope

slope = calculate_slope(elevation)

# Plot the slope
plt.figure(figsize=(10, 6))
plt.imshow(slope, cmap='viridis')
plt.colorbar(label='Slope (degrees)')
plt.title('Slope Derived from CartoSat-1 DEM')
plt.xlabel('Pixel Index (X)')
plt.ylabel('Pixel Index (Y)')
```

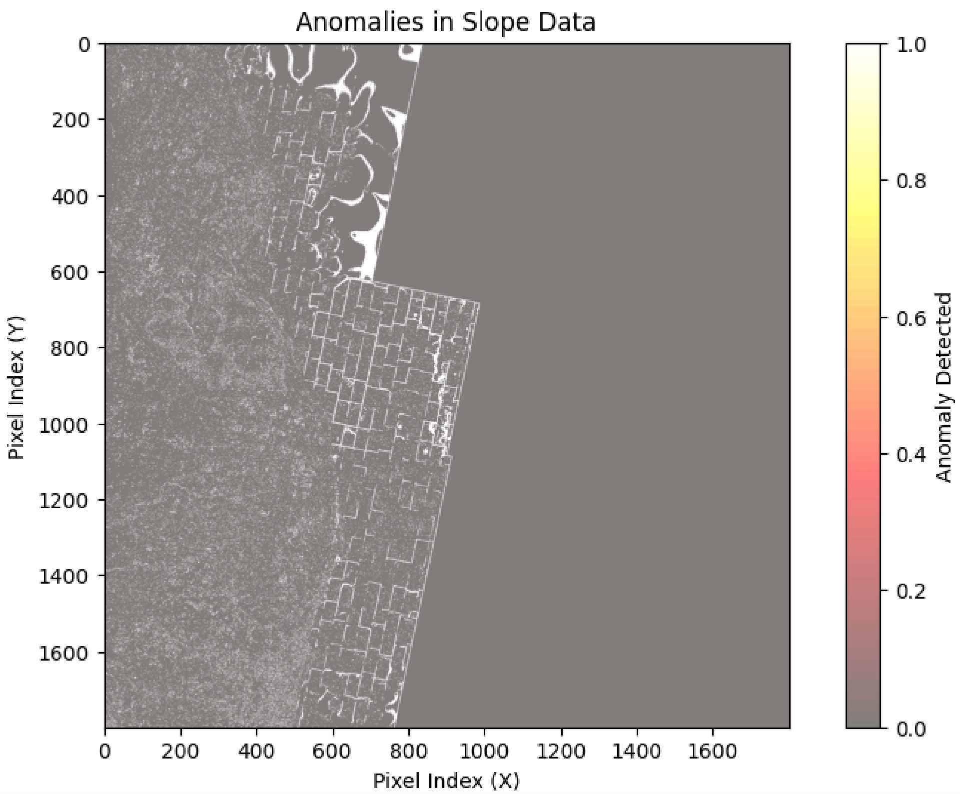
```
plt.show()
```



Anomaly Detection with Machine Learning

```
# Flatten the slope data for ML
slope_flat = slope.flatten().reshape(-1, 1)  # Reshape for model input

# Create and fit the Isolation Forest model
model = IsolationForest(contamination=0.05)  # Adjust contamination based on expected anomaly rate
model.fit(slope_flat)

# Predict anomalies
anomaly_pred = model.predict(slope_flat)
anomalies = anomaly_pred == -1  # -1 indicates an anomaly

# Reshape anomalies back to the original slope shape
anomaly_map = anomalies.reshape(slope.shape)
```

```
# Plot the anomalies detected
plt.figure(figsize=(10, 6))
plt.imshow(anomaly_map, cmap='hot', alpha=0.5)  # Show anomalies over the slope
plt.colorbar(label='Anomaly Detected')
plt.title('Anomalies in Slope Data')
plt.xlabel('Pixel Index (X)')
plt.ylabel('Pixel Index (Y)')
plt.show()
```

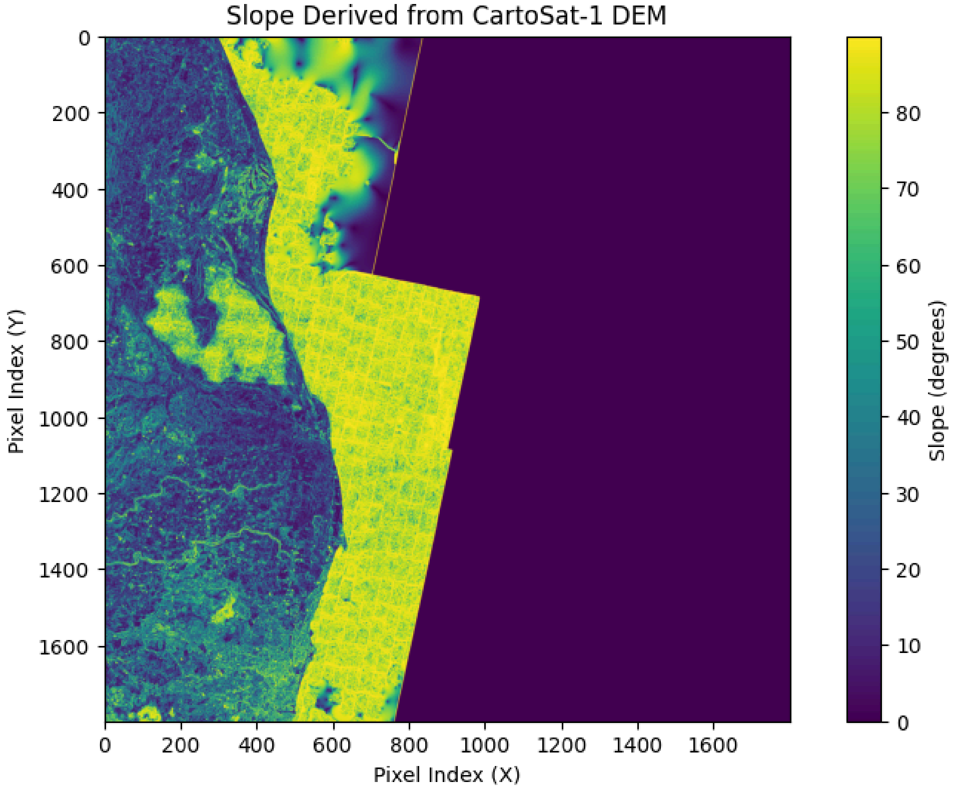


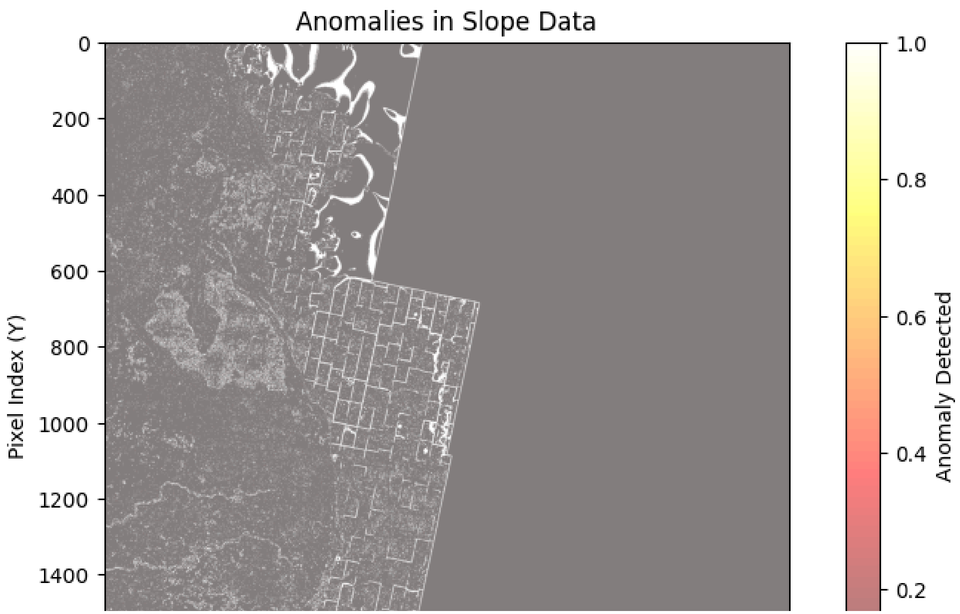
Output

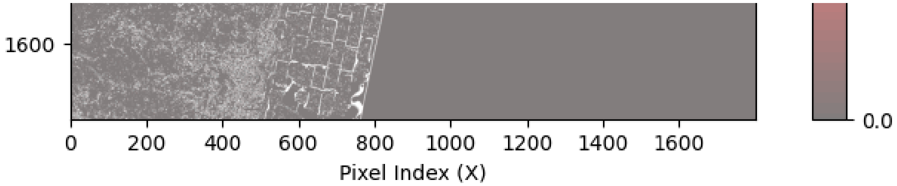
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Function to calculate slope from DEM
def calculate_slope(dem):
    x_gradient = np.gradient(dem, axis=1)
```

```
    y_gradient = np.gradient(dem, axis=0)
    slope = np.arctan(np.sqrt(x_gradient**2 + y_gradient**2)) * (180/np.pi)
    return slope

# Assuming 'elevation' is your DEM data loaded as a NumPy array
slope = calculate_slope(elevation)

# Plot the slope
plt.figure(figsize=(10, 6))
plt.imshow(slope, cmap='viridis')
plt.colorbar(label='Slope (degrees)')
plt.title('Slope Derived from CartoSat-1 DEM')
plt.xlabel('Pixel Index (X)')
plt.ylabel('Pixel Index (Y)')
plt.show()

# Flatten the slope data for ML
slope_flat = slope.flatten().reshape(-1, 1)  # Reshape for model input

# Create and fit the Isolation Forest model
model = IsolationForest(contamination=0.05)  # Adjust contamination based on expected anomaly rate
model.fit(slope_flat)

# Predict anomalies
anomaly_pred = model.predict(slope_flat)
anomalies = anomaly_pred == -1  # -1 indicates an anomaly

# Reshape anomalies back to the original slope shape
anomaly_map = anomalies.reshape(slope.shape)

# Plot the anomalies detected
plt.figure(figsize=(10, 6))
plt.imshow(anomaly_map, cmap='hot', alpha=0.5)  # Show anomalies over the slope
plt.colorbar(label='Anomaly Detected')
plt.title('Anomalies in Slope Data')
plt.xlabel('Pixel Index (X)')
plt.ylabel('Pixel Index (Y)')
plt.show()

# Analyze anomalies
anomaly_indices = np.argwhere(anomalies)  # Get coordinates of detected anomalies
threshold = 5  # Lowered the threshold to ensure anomalies are detected (adjust based on your data)

# Debugging: Check how many anomalies are detected
print(f"Total anomalies detected: {len(anomaly_indices)}")

if len(anomaly_indices) == 0:
    print("No anomalies detected. Try adjusting the contamination rate or checking the data.")
else:
    for index in anomaly_indices:
        # Flatten index to ensure proper unpacking
```

```
        if index.size == 2:  # Ensure it has two elements
            y, x = index.flatten()  # Unpack coordinates
            elevation_value = slope[y, x]  # Use slope value for analysis

            # Print the coordinates and slope values for debugging
            print(f"Coordinates: ({x}, {y}), Slope Value: {elevation_value}")

            # Basic interpretation and suggestion
            if elevation_value > threshold:  # Check if it crosses the threshold
                print(f"Anomaly detected at ({x}, {y}) with slope value: {elevation_value:.2f}")
                print("Suggestion: This area may be prone to landslides. Consider reinforcing the slope or monitoring for heavy rains.")
            else:
                print(f"Anomaly detected at ({x}, {y}) with slope value: {elevation_value:.2f}")
                print("Suggestion: Further investigation is needed to understand potential erosion or instability.")
```


Slope Derived from CartoSat-1 DEM
Pixel Index (Y)
Pixel Index (X)
Slope (degrees)
0
200
400
600
800
1000
1200
1400
1600
10
20
30
40
50
60
70
80


Anomalies in Slope Data
Pixel Index (Y)
Anomaly Detected
0
200
400
600
800
1000
1200
1400
0.2
0.4
0.6
0.8
1.0

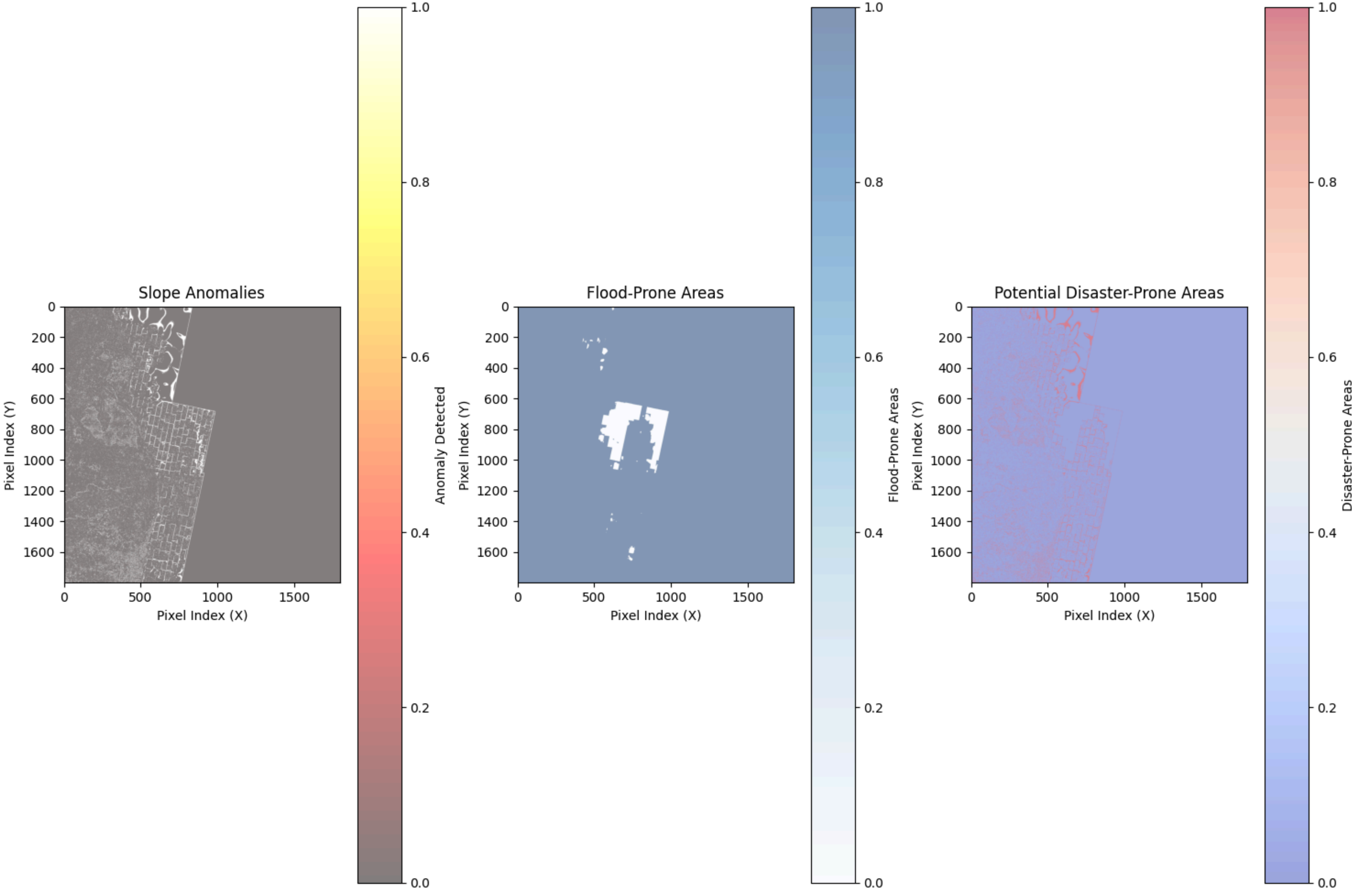



```
Total anomalies detected: 161674
```

Improved code: This code detects slope anomalies and identifies flood-prone areas in a Digital Elevation Model (DEM), ultimately combining these results to highlight potential disaster-prone regions.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Assuming slope and elevation are your 2D numpy arrays representing the slope and elevation data from the DEM
slope_flat = slope.flatten().reshape(-1, 1)  # Reshape for model input
elevation_flat = elevation.flatten().reshape(-1, 1)  # Reshape elevation for model input

# Create and fit the Isolation Forest model
model = IsolationForest(contamination=0.05)  # Adjust contamination based on expected anomaly rate
model.fit(slope_flat)

# Predict anomalies in slope data
anomaly_pred = model.predict(slope_flat)
slope_anomalies = anomaly_pred == -1  # -1 indicates an anomaly

# Reshape anomalies back to the original slope shape
slope_anomaly_map = slope_anomalies.reshape(slope.shape)

# Detect flood-prone areas based on elevation (below a threshold)
flood_threshold = 50  # Elevation threshold in meters (customize based on your area)
flood_prone_mask = elevation < flood_threshold  # Areas below the flood threshold

# Ensure flood_prone_mask has the same shape as slope_anomaly_map
if flood_prone_mask.shape != slope_anomaly_map.shape:
    flood_prone_mask = flood_prone_mask.reshape(slope.shape)  # Reshape if necessary

# Combine slope anomalies with flood risk
disaster_prone_mask = slope_anomaly_map & flood_prone_mask  # Areas that are both slope anomalies and flood-prone

# Plot the detected anomalies and potential disaster-prone areas
plt.figure(figsize=(15, 10))

# Plot slope anomalies
plt.subplot(1, 3, 1)
plt.imshow(slope_anomaly_map, cmap='hot', alpha=0.5)
plt.colorbar(label='Anomaly Detected')
```

```
    plt.title('Slope Anomalies')
    plt.xlabel('Pixel Index (X)')
    plt.ylabel('Pixel Index (Y)')

    # Plot flood-prone areas
    plt.subplot(1, 3, 2)
    plt.imshow(flood_prone_mask, cmap='Blues', alpha=0.5)  # Highlight flood-prone areas
    plt.colorbar(label='Flood-Prone Areas')
    plt.title('Flood-Prone Areas')
    plt.xlabel('Pixel Index (X)')
    plt.ylabel('Pixel Index (Y)')

    # Plot combined disaster-prone areas
    plt.subplot(1, 3, 3)
    plt.imshow(disaster_prone_mask, cmap='coolwarm', alpha=0.5)  # Highlight potential disaster-prone areas
    plt.colorbar(label='Disaster-Prone Areas')
    plt.title('Potential Disaster-Prone Areas')
    plt.xlabel('Pixel Index (X)')
    plt.ylabel('Pixel Index (Y)')

    plt.tight_layout()
    plt.show()
```


Slope Anomalies
Pixel Index (Y)
Pixel Index (X)
Anomaly Detected
Flood-Prone Areas
Pixel Index (Y)
Pixel Index (X)
Flood-Prone Areas
Potential Disaster-Prone Areas
Pixel Index (Y)
Pixel Index (X)
Disaster-Prone Areas

Old data

```
!pip install folium
```

```
Requirement already satisfied: folium in /usr/local/lib/python3.11/dist-packages (0.19.5)
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from folium) (0.8.1)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.11/dist-packages (from folium) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from folium) (2.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from folium) (2.32.3)
Requirement already satisfied: xyzservices in /usr/local/lib/python3.11/dist-packages (from folium) (2025.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2>=2.9->folium) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->folium) (2025.1.31)
```

```
import os
import numpy as np
import pandas as pd
import folium
from folium.plugins import MarkerCluster
from sklearn.cluster import KMeans
from scipy.spatial import cKDTree
import rasterio
import matplotlib.pyplot as plt
from PIL import Image
from google.colab import files

# --- Step 1: Load DEM Geographic Bounds ---
with rasterio.open(cartoDEM_path) as dem_data:
    bounds = dem_data.bounds
    dem_array = dem_data.read(1)  # Read elevation data
    transform = dem_data.transform

min_longitude, min_latitude, max_longitude, max_latitude = bounds
print(f"DEM Area -> Min Lat: {min_latitude}, Max Lat: {max_latitude}, Min Lon: {min_longitude}, Max Lon: {max_longitude}")

# --- Step 2: Load Disaster Data ---
disaster_data = pd.read_csv('/content/drive/MyDrive/Project Bhuvan/india_natural_disasters.csv')

# ✅ Debug: Print column names to check for mismatches
print("Disaster Data Columns:", disaster_data.columns)

# ✅ Ensure correct column names
lat_col = "latitude" if "latitude" in disaster_data.columns else "Latitude"
lon_col = "longitude" if "longitude" in disaster_data.columns else "Longitude"
type_col = "disaster_type" if "disaster_type" in disaster_data.columns else "Disaster Type"

# ✅ Filter disaster data to match the DEM area
```

```
filtered_disaster_data = disaster_data[
    (disaster_data[lat_col] >= min_latitude) &
    (disaster_data[lat_col] <= max_latitude) &
    (disaster_data[lon_col] >= min_longitude) &
    (disaster_data[lon_col] <= max_longitude)
]

print(f"Total disasters in DEM region: {len(filtered_disaster_data)}")

# --- Step 3: Detect Disaster-Prone Areas ---
disaster_coords = np.argwhere(disaster_prone_mask)

if len(disaster_coords) > 0:
    latitudes = (disaster_coords[:, 0] / disaster_prone_mask.shape[0]) * (max_latitude - min_latitude) + min_latitude
    longitudes = (disaster_coords[:, 1] / disaster_prone_mask.shape[1]) * (max_longitude - min_longitude) + min_longitude
    disaster_latlon = np.column_stack((latitudes, longitudes))

    # ✅ Dynamic clustering for accuracy
    num_clusters = max(len(disaster_latlon) // 5000, 1)
    kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
    kmeans.fit(disaster_latlon)
    clustered_anomalies = kmeans.cluster_centers_

    print(f"Total clustered disaster-prone points: {len(clustered_anomalies)}")
else:
    clustered_anomalies = []

# --- Step 4: Find the Most Relevant Disaster Type for Each Cluster ---
def find_most_relevant_disaster(lat, lon, disaster_df, k=3):
    """Finds the most relevant disaster type using weighted distance-based matching."""
    if disaster_df.empty:
        return "Flood"

    disaster_coords = disaster_df[[lat_col, lon_col]].values
    tree = cKDTree(disaster_coords)
    dist, idx = tree.query([lat, lon], k=min(k, len(disaster_coords)))

    nearest_disasters = disaster_df.iloc[idx][type_col]
    weights = 1 / (dist + 1e-6)  # Avoid division by zero
    weighted_disaster_counts = {}

    for disaster, weight in zip(nearest_disasters, weights):
        weighted_disaster_counts[disaster] = weighted_disaster_counts.get(disaster, 0) + weight

    return max(weighted_disaster_counts, key=weighted_disaster_counts.get)

# --- Step 5: Create the Map ---
m = folium.Map(location=[(min_latitude + max_latitude) / 2, (min_longitude + max_longitude) / 2], zoom_start=8)
marker_cluster = MarkerCluster().add_to(m)

# ✅ Plot clustered disaster-prone areas with disaster type labels
```

```
    for lat, lon in clustered_anomalies:
        nearest_disaster = find_most_relevant_disaster(lat, lon, filtered_disaster_data)

        folium.CircleMarker(
            location=[lat, lon],
            radius=5,
            color='red',
            fill=True,
            fill_color='red',
            fill_opacity=0.6,
            popup=f"Predicted Disaster: {nearest_disaster}"
        ).add_to(marker_cluster)

    print(f"Plotted {len(clustered_anomalies)} disaster-prone areas.")

    # ✅ Add ALL historical disasters across India
    for index, row in disaster_data.iterrows():
```