In [2]:
```
import os
print(os.getcwd())  # This shows the current folder
```

```
C:\Users\Administrator
```

In [3]:
```
import os

# Create the directory if it doesn't exist
directory = 'C:\\Users\\Administrator\\NEWPROJECT'
try:
    os.makedirs(directory, exist_ok=True)
    print(f"Directory created or already exists at: {directory}")
except Exception as e:
    print(f"Error creating directory: {e}")

# Now try to change to that directory
try:
    os.chdir(directory)
    print(f"Current working directory: {os.getcwd()}")
except Exception as e:
    print(f"Error changing directory: {e}")
```

```
Directory created or already exists at: C:\Users\Administrator\NEWPROJECT
Current working directory: C:\Users\Administrator\NEWPROJECT
```

River Data Collection

In [5]:
```
def continuous_monitoring(interval_minutes=15):
    while True:
        print(f"\nFetching data at {datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        river_data = main()
        print("Waiting for next update...")
        time.sleep(interval_minutes * 60)  # Convert minutes to seconds

# Run continuous monitoring
# continuous_monitoring()  # Uncomment this line to start continuous monitoring
```

In [6]:
```
import requests
import pandas as pd
from datetime import datetime

def get_station_data(station_id):
    """
    Fetch real-time river level data for a specific station and show collection
    """
    base_url = f"https://environment.data.gov.uk/flood-monitoring/id/stations/{s

    try:
        response = requests.get(base_url)
        response.raise_for_status()

        data = response.json()
        readings = data.get('items', [])

        if readings:
            df = pd.DataFrame(readings)
            # Convert dateTime to datetime object
            df['dateTime'] = pd.to_datetime(df['dateTime'])
```

```
            # Print date range for this station
            print(f"\nStation {station_id} data collection period:")
            print(f"Earliest reading: {df['dateTime'].min()}")
            print(f"Latest reading: {df['dateTime'].max()}")
            print(f"Total readings: {len(df)}")

            # Rename columns for clarity
            df = df.rename(columns={'value': 'water_level'})
            # Add station ID column
            df['station_id'] = station_id
            return df

        return None

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data for station {station_id}: {e}")
        return None

def main():
    # List of station IDs in Greater Manchester
    station_ids = ['690203', '690510', '690160']

    # Create empty DataFrame to store all data
    all_data = pd.DataFrame()

    # Fetch data for each station
    for station_id in station_ids:
        print(f"\nFetching data for station {station_id}...")
        station_data = get_station_data(station_id)
        if station_data is not None:
            all_data = pd.concat([all_data, station_data], ignore_index=True)

    if not all_data.empty:
        print("\nOverall dataset summary:")
        print(f"Total readings across all stations: {len(all_data)}")
        print("\nDate range for entire dataset:")
        print(f"Earliest reading: {all_data['dateTime'].min()}")
        print(f"Latest reading: {all_data['dateTime'].max()}")

        # Show readings per station
        print("\nReadings per station:")
        print(all_data.groupby('station_id').size())

        # Optional: Display first few rows of the data
        print("\nFirst few rows of the collected data:")
        print(all_data.head())

    return all_data

# Run the script
river_data = main()
```

```
Fetching data for station 690203...

Station 690203 data collection period:
Earliest reading: 2024-12-31 00:00:00+00:00
Latest reading: 2025-01-05 04:45:00+00:00
Total readings: 500

Fetching data for station 690510...

Station 690510 data collection period:
Earliest reading: 2024-12-31 00:00:00+00:00
Latest reading: 2025-01-05 04:45:00+00:00
Total readings: 500

Fetching data for station 690160...

Station 690160 data collection period:
Earliest reading: 2024-12-31 00:00:00+00:00
Latest reading: 2025-01-05 04:45:00+00:00
Total readings: 500

Overall dataset summary:
Total readings across all stations: 1500

Date range for entire dataset:
Earliest reading: 2024-12-31 00:00:00+00:00
Latest reading: 2025-01-05 04:45:00+00:00

Readings per station:
station_id
690160    500
690203    500
690510    500
dtype: int64

First few rows of the collected data:
                                                 @id  \
0  http://environment.data.gov.uk/flood-monitorin...
1  http://environment.data.gov.uk/flood-monitorin...
2  http://environment.data.gov.uk/flood-monitorin...
3  http://environment.data.gov.uk/flood-monitorin...
4  http://environment.data.gov.uk/flood-monitorin...

                   dateTime  \
0 2024-12-31 00:00:00+00:00
1 2024-12-31 00:15:00+00:00
2 2024-12-31 00:30:00+00:00
3 2024-12-31 00:45:00+00:00
4 2024-12-31 01:00:00+00:00

                                             measure  water_level station_id
0  http://environment.data.gov.uk/flood-monitorin...        0.206     690203
1  http://environment.data.gov.uk/flood-monitorin...        0.206     690203
2  http://environment.data.gov.uk/flood-monitorin...        0.206     690203
3  http://environment.data.gov.uk/flood-monitorin...        0.206     690203
4  http://environment.data.gov.uk/flood-monitorin...        0.206     690203
```

In [ ]:

```
import requests
import pandas as pd
from datetime import datetime, timezone
```

```
import time
import os

def get_latest_readings(station_id):
    """
    Fetch only the most recent reading for a specific station
    """
    base_url = f"https://environment.data.gov.uk/flood-monitoring/id/stations/{s
    params = {
        '_limit': 1,  # Get only the latest reading
        '_sorted': ''
    }

    try:
        response = requests.get(base_url, params=params)
        response.raise_for_status()

        data = response.json()
        reading = data.get('items', [])[0] if data.get('items') else None

        if reading:
            df = pd.DataFrame([reading])
            df['dateTime'] = pd.to_datetime(df['dateTime'])
            df = df.rename(columns={'value': 'water_level'})
            df['station_id'] = station_id
            return df
        return None

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data for station {station_id}: {e}")
        return None

def collect_data_continuously(project_path, interval_minutes=15):
    """
    Continuously collect data and save to the project folder
    """
    station_ids = ['690203', '690510', '690160']

    # Create data folder if it doesn't exist
    data_folder = os.path.join(project_path, 'river_data')
    os.makedirs(data_folder, exist_ok=True)

    # Path for the main data file
    data_file = os.path.join(data_folder, 'river_data_continuous.csv')

    # Load existing data if available
    try:
        all_data = pd.read_csv(data_file)
        all_data['dateTime'] = pd.to_datetime(all_data['dateTime'])
        print(f"Loaded existing data file with {len(all_data)} records")
    except FileNotFoundError:
        all_data = pd.DataFrame()
        print("Created new data file")

    print(f"\nStarting continuous data collection every {interval_minutes} minut
    print("Press Ctrl+C to stop the collection")

    try:
        while True:
            current_time = datetime.now(timezone.utc)
```

```
            print(f"\nFetching data at {current_time.strftime('%Y-%m-%d %H:%M:%S

            # Create empty DataFrame for new readings
            new_data = pd.DataFrame()

            # Fetch latest reading for each station
            for station_id in station_ids:
                station_data = get_latest_readings(station_id)
                if station_data is not None:
                    new_data = pd.concat([new_data, station_data], ignore_index=
                    print(f"Station {station_id} - Water Level: {station_data['w

            # Add new data to existing data
            if not new_data.empty:
                all_data = pd.concat([all_data, new_data], ignore_index=True)

                # Remove duplicates based on dateTime and station_id
                all_data = all_data.drop_duplicates(subset=['dateTime', 'station

                # Sort by dateTime
                all_data = all_data.sort_values('dateTime')

                # Save main CSV file
                all_data.to_csv(data_file, index=False)

                # Save daily backup file
                daily_backup_file = os.path.join(data_folder,
                                                f"river_data_{current_time.strfti
                all_data.to_csv(daily_backup_file, index=False)

                print(f"Data saved - Total records: {len(all_data)}")

            # Calculate wait time until next collection
            next_collection = current_time + pd.Timedelta(minutes=interval_minut
            sleep_seconds = (next_collection - datetime.now(timezone.utc)).total

            if sleep_seconds > 0:
                print(f"Waiting until {next_collection.strftime('%H:%M:%S UTC')}
                time.sleep(sleep_seconds)

    except KeyboardInterrupt:
        print("\nData collection stopped by user")
        print(f"Data saved to {data_file}")

# Set your project path
PROJECT_PATH = r"C:\Users\Administrator\NEWPROJECT"

# Start the continuous data collection
if __name__ == "__main__":
    collect_data_continuously(PROJECT_PATH, 15)  # Collect every 15 minutes
```

```
Loaded existing data file with 15 records

Starting continuous data collection every 15 minutes...
Press Ctrl+C to stop the collection

Fetching data at 2025-01-29 05:26:59 UTC
Station 690203 - Water Level: 0.314m at 2025-01-29T05:00:00.000000000
Station 690510 - Water Level: 1.2m at 2025-01-29T05:00:00.000000000
Station 690160 - Water Level: 0.447m at 2025-01-29T05:00:00.000000000
Data saved - Total records: 18
Waiting until 05:41:59 UTC for next collection...
```

In [ ]:
```
%run collect_river_data.py
```

In [ ]:
```
# In your Python console or Jupyter notebook:
import pandas as pd
df = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\river_data\river_data_conti
print(df)  # See all records
print(len(df))  # Number of records
print(df['dateTime'].min())  # Earliest reading
print(df['dateTime'].max())  # Latest reading
```

In [7]:
```
import requests
import pandas as pd
from datetime import datetime

# Define the target stations
stations = ['690203', '690510', '690160']
base_url = "https://environment.data.gov.uk/flood-monitoring/id/stations/{}/read

def fetch_rainfall_data(station_id):
    """Fetch real-time rainfall data for a given station ID."""
    url = base_url.format(station_id)

    try:
        response = requests.get(url)
        response.raise_for_status()  # Raise an error for bad responses (4xx, 5x
        data = response.json()

        # Extract latest reading
        if "items" in data and len(data["items"]) > 0:
            latest_reading = data["items"][0]  # Assuming the first item is the
            return {
                "station_id": station_id,
                "dateTime": latest_reading.get("dateTime", "N/A"),
                "value": latest_reading.get("value", "N/A"),
                "unit": latest_reading.get("measure", "N/A")
            }
        else:
            return {"station_id": station_id, "error": "No data available"}

    except requests.exceptions.RequestException as e:
        return {"station_id": station_id, "error": str(e)}

# Collect rainfall data for all stations
rainfall_data = [fetch_rainfall_data(station) for station in stations]

# Convert data to a DataFrame
df = pd.DataFrame(rainfall_data)
```

```python
# Save to CSV
df.to_csv("real_time_rainfall_data.csv", index=False)

# Print the collected data
print(df)
```

```
  station_id              dateTime  value  \
0     690203  2024-12-31T00:00:00Z  0.206
1     690510  2024-12-31T00:00:00Z  0.933
2     690160  2024-12-31T00:00:00Z  0.341

                                                unit
0  http://environment.data.gov.uk/flood-monitorin...
1  http://environment.data.gov.uk/flood-monitorin...
2  http://environment.data.gov.uk/flood-monitorin...
```

In [9]:
```python
import requests
import pandas as pd

# Define the target stations
stations = ['690203', '690510', '690160']
base_url = "https://environment.data.gov.uk/flood-monitoring/id/stations/{}/read

def fetch_unit_from_measure(measure_url):
    """Fetch the measurement unit from the given measure URL."""
    try:
        response = requests.get(measure_url)
        response.raise_for_status()
        data = response.json()

        # Extract unitName from API response
        return data.get("items", {}).get("unitName", "Unknown")
    except requests.exceptions.RequestException:
        return "Unknown"

def fetch_rainfall_data(station_id):
    """Fetch real-time rainfall data for a given station ID."""
    url = base_url.format(station_id)

    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()

        if "items" in data and len(data["items"]) > 0:
            latest_reading = data["items"][0]  # Latest reading
            measure_url = latest_reading.get("measure", "")

            # Fetch unit dynamically
            readable_unit = fetch_unit_from_measure(measure_url) if measure_url

            return {
                "station_id": station_id,
                "dateTime": latest_reading.get("dateTime", "N/A"),
                "value": latest_reading.get("value", "N/A"),
                "unit": readable_unit  # Correct unit
            }
        else:
            return {"station_id": station_id, "error": "No data available"}
```

```
    except requests.exceptions.RequestException as e:
        return {"station_id": station_id, "error": str(e)}

# Collect rainfall data for all stations
rainfall_data = [fetch_rainfall_data(station) for station in stations]

# Convert data to a DataFrame
df = pd.DataFrame(rainfall_data)

# Save to CSV
df.to_csv("real_time_rainfall_data.csv", index=False)

# Print the collected data
print(df)
```

```
  station_id              dateTime  value unit
0     690203  2024-12-31T00:00:00Z  0.206    m
1     690510  2024-12-31T00:00:00Z  0.933    m
2     690160  2024-12-31T00:00:00Z  0.341    m
```

Collecting Rainfall Data

In [11]:

```
import requests

url = "https://environment.data.gov.uk/flood-monitoring/id/stations?parameter=ra
response = requests.get(url)
stations = response.json()

# Print a few stations to check
for station in stations["items"][:10]:  # Show first 10 stations
    print(station["notation"], station["label"])
```

```
E7050 Rainfall station
4163 Day Brook
0890TH Lechlade
E1310 Weldon Flood Storage Reservoir
3680 Rainfall station
3275 Rainfall station
3167 Rainfall station
3307 Rainfall station
3404 Rainfall station
3014 Rainfall station
```

In [14]:

```
import requests
import pandas as pd

# Get all rainfall stations
url = "https://environment.data.gov.uk/flood-monitoring/id/stations?parameter=ra
response = requests.get(url)
data = response.json()

# Extract relevant details safely
stations = []
for station in data.get("items", []):
    stations.append({
        "id": station.get("notation", "Unknown"),
        "name": station.get("label", "Unknown"),
        "lat": station.get("lat", None),
        "lon": station.get("long", None),
```

```
        "region": station.get("catchmentName", "Unknown")
    })

# Convert to DataFrame
df = pd.DataFrame(stations)

# Approximate Manchester coordinates (latitude: ~53.5, longitude: ~-2.2)
manchester_lat_min, manchester_lat_max = 53.3, 53.7
manchester_lon_min, manchester_lon_max = -2.5, -1.9

# Filter stations near Manchester
df_manchester = df[
    (df["lat"].between(manchester_lat_min, manchester_lat_max, inclusive="both"))
    (df["lon"].between(manchester_lon_min, manchester_lon_max, inclusive="both"))
]

print(df_manchester)

# Save to CSV
df_manchester.to_csv("manchester_rainfall_stations.csv", index=False)
```

```
           id              name        lat       lon   region
62     564769  Rainfall station  53.300189 -2.155251  Unknown
126    564154  Rainfall station  53.497891 -2.499671  Unknown
156    077800  Rainfall station  53.636457 -2.005248  Unknown
183    077836  Rainfall station  53.657107 -1.925056  Unknown
184    078530  Rainfall station  53.592394 -1.929702  Unknown
221    559586  Rainfall station  53.534889 -2.012778  Unknown
239   559100R  Rainfall station  53.459370 -1.934440  Unknown
402    561299  Rainfall station  53.657120 -2.049131  Unknown
403    560943  Rainfall station  53.697953 -2.353609  Unknown
490    575935  Rainfall station  53.694788 -2.486856  Unknown
493    562417  Rainfall station  53.609580 -2.440532  Unknown
507    558491  Rainfall station  53.389250 -1.919511  Unknown
544    559969  Rainfall station  53.459311 -2.134747  Unknown
589    562992  Rainfall station  53.535499 -2.263235  Unknown
629    561613  Rainfall station  53.663285 -2.180798  Unknown
681    560557  Rainfall station  53.431001 -2.352893  Unknown
805    563599  Rainfall station  53.462415 -2.368215  Unknown
824    562656  Rainfall station  53.539768 -2.350780  Unknown
861    562811  Rainfall station  53.560873 -2.141109  Unknown
913   562992_  Rainfall station  53.535499 -2.263235  Unknown
938    563170  Rainfall station  53.534780 -2.162144  Unknown
988  Egerton1  Rainfall station  53.636515 -2.448375  Unknown
1006   558975  Rainfall station  53.463871 -1.947989  Unknown
```

In [18]:

```
import requests
import pandas as pd
from geopy.distance import geodesic

# Target river stations
river_station_ids = ["690203", "690510", "690160"]

# API URL template for river stations
river_url = "https://environment.data.gov.uk/flood-monitoring/id/stations/{}"

# Get river station locations
river_stations = []
for station in river_station_ids:
    response = requests.get(river_url.format(station))
```

```
    if response.status_code == 200:
        data = response.json().get("items", {})
        river_stations.append({
            "id": station,
            "name": data.get("label", "Unknown"),
            "lat": data.get("lat", None),
            "lon": data.get("long", None)
        })

# Convert to DataFrame
df_river = pd.DataFrame(river_stations)
print(df_river)
```

```
       id                  name        lat       lon
0  690203              Rochdale  53.611067 -2.178685
1  690510  Manchester Racecourse  53.499526 -2.271756
2  690160           Bury Ground  53.598766 -2.305182
```

Locating Rainfall Stations close to River Stations

In [17]:
```
pip install geopy
```

```
Collecting geopy
  Downloading geopy-2.4.1-py3-none-any.whl.metadata (6.8 kB)
Collecting geographiclib<3,>=1.52 (from geopy)
  Downloading geographiclib-2.0-py3-none-any.whl.metadata (1.4 kB)
Downloading geopy-2.4.1-py3-none-any.whl (125 kB)
Downloading geographiclib-2.0-py3-none-any.whl (40 kB)
Installing collected packages: geographiclib, geopy
Successfully installed geographiclib-2.0 geopy-2.4.1
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```
# List of known rainfall stations (with lat/lon from previous results)
rainfall_stations = [
    {"id": "564769", "lat": 53.300189, "lon": -2.155251},
    {"id": "564154", "lat": 53.497891, "lon": -2.499671},
    {"id": "077800", "lat": 53.636457, "lon": -2.005248},
    {"id": "077836", "lat": 53.657107, "lon": -1.925056},
    {"id": "078530", "lat": 53.592394, "lon": -1.929702},
    {"id": "559586", "lat": 53.534889, "lon": -2.012778},
    {"id": "562992", "lat": 53.535499, "lon": -2.263235},
]

# Define max distance (in km)
MAX_DISTANCE = 10

# Find closest rainfall stations
nearby_stations = []
for river in df_river.itertuples():
    river_location = (river.lat, river.lon)
    for rainfall in rainfall_stations:
        rainfall_location = (rainfall["lat"], rainfall["lon"])
        distance = geodesic(river_location, rainfall_location).km
        if distance <= MAX_DISTANCE:
            nearby_stations.append(rainfall["id"])

# Remove duplicates
nearby_stations = list(set(nearby_stations))
print("Nearby Rainfall Stations:", nearby_stations)
```

```
Nearby Rainfall Stations: ['562992']
```

Locating rainfall stations close to the 3 river stations

In [22]:
```
import requests
import pandas as pd
import math
from datetime import datetime, timezone

def get_all_rainfall_stations():
    """
    Get all stations that measure rainfall
    """
    url = "http://environment.data.gov.uk/flood-monitoring/id/stations"
    params = {
        'parameter': 'rainfall',
        '_limit': 10000  # Get all stations
    }

    response = requests.get(url, params=params)
    if response.status_code == 200:
        data = response.json()
        return data.get('items', [])
    return []

def get_river_station_location(station_id):
    """
    Get coordinates of a river station
    """
    url = f"http://environment.data.gov.uk/flood-monitoring/id/stations/{station
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        station = data.get('items', {})
        return {
            'id': station_id,
            'name': station.get('label', ''),
            'lat': station.get('lat'),
            'long': station.get('long')
        }
    return None

def calculate_distance(lat1, lon1, lat2, lon2):
    """
    Calculate distance between two points in kilometers
    """
    R = 6371  # Earth's radius in km

    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = (math.sin(dlat/2) * math.sin(dlat/2) +
         math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) *
         math.sin(dlon/2) * math.sin(dlon/2))
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    return R * c

def find_nearest_rainfall_stations():
    """
    Find rainfall stations nearest to our river stations
```

```
    """
    river_stations = ['690203', '690510', '690160']
    river_locations = []

    print("Getting river station locations...")
    for station_id in river_stations:
        location = get_river_station_location(station_id)
        if location:
            river_locations.append(location)
            print(f"\nRiver Station: {location['name']}")
            print(f"Location: {location['lat']}, {location['long']}")

    print("\nFetching all rainfall stations...")
    rainfall_stations = get_all_rainfall_stations()
    print(f"Found {len(rainfall_stations)} rainfall stations")

    nearest_stations = {}
    max_distance = 10  # Maximum distance in kilometers

    print("\nFinding nearest rainfall stations...")
    for river in river_locations:
        nearest = []
        print(f"\nNearest rainfall stations to {river['name']}:")

        for rainfall in rainfall_stations:
            if rainfall.get('lat') and rainfall.get('long'):
                distance = calculate_distance(
                    river['lat'], river['long'],
                    rainfall['lat'], rainfall['long']
                )

                if distance <= max_distance:
                    nearest.append({
                        'station_id': rainfall['stationReference'],
                        'name': rainfall['label'],
                        'distance': distance,
                        'lat': rainfall['lat'],
                        'long': rainfall['long']
                    })

        # Sort by distance and get the closest stations
        nearest = sorted(nearest, key=lambda x: x['distance'])
        nearest_stations[river['id']] = nearest[:3]  # Get top 3 nearest station

        # Print the results
        if nearest:
            for station in nearest:
                print(f"- {station['name']}")
                print(f"  Distance: {station['distance']:.2f} km")
                print(f"  ID: {station['station_id']}")
                print(f"  Location: {station['lat']}, {station['long']}")
        else:
            print("No rainfall stations found within 10km")

    return nearest_stations

if __name__ == "__main__":
    print("Finding nearest rainfall stations to river monitoring points...")
    nearest_stations = find_nearest_rainfall_stations()
```

```
Finding nearest rainfall stations to river monitoring points...
Getting river station locations...

River Station: Rochdale
Location: 53.611067, -2.178685

River Station: Manchester Racecourse
Location: 53.499526, -2.271756

River Station: Bury Ground
Location: 53.598766, -2.305182

Fetching all rainfall stations...
Found 1012 rainfall stations

Finding nearest rainfall stations...

Nearest rainfall stations to Rochdale:
- Rainfall station
  Distance: 5.81 km
  ID: 561613
  Location: 53.663285, -2.180798
- Rainfall station
  Distance: 6.11 km
  ID: 562811
  Location: 53.560873, -2.141109
- Rainfall station
  Distance: 8.55 km
  ID: 563170
  Location: 53.53478, -2.162144
- Rainfall station
  Distance: 9.96 km
  ID: 561299
  Location: 53.65712, -2.049131

Nearest rainfall stations to Manchester Racecourse:
- Rainfall station
  Distance: 4.04 km
  ID: 562992
  Location: 53.535499, -2.263235
- Rainfall station
  Distance: 4.04 km
  ID: 562992_
  Location: 53.535499, -2.263235
- Rainfall station
  Distance: 6.88 km
  ID: 562656
  Location: 53.539768, -2.35078
- Rainfall station
  Distance: 7.60 km
  ID: 563599
  Location: 53.462415, -2.368215
- Rainfall station
  Distance: 8.24 km
  ID: 563170
  Location: 53.53478, -2.162144
- Rainfall station
  Distance: 9.32 km
  ID: 560557
  Location: 53.431001, -2.352893
```

```
Nearest rainfall stations to Bury Ground:
- Rainfall station
  Distance: 7.22 km
  ID: 562656
  Location: 53.539768, -2.35078
- Rainfall station
  Distance: 7.56 km
  ID: 562992
  Location: 53.535499, -2.263235
- Rainfall station
  Distance: 7.56 km
  ID: 562992_
  Location: 53.535499, -2.263235
- Rainfall station
  Distance: 9.01 km
  ID: 562417
  Location: 53.60958, -2.440532
```

Extracting Monthly Weather Data for Manchester, Bury, and Rochdale.

In [32]:
```
import pandas as pd
import os
import numpy as np

def clean_met_office_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WE

    # Read the files
    temp_df = pd.read_excel(os.path.join(project_path, 'TEMPERATURE.xlsx'))
    precip_df = pd.read_excel(os.path.join(project_path, 'PRECIPITATION.xlsx'))

    # Debug: Print first few rows to understand data structure
    print("Temperature DataFrame First Few Rows:")
    print(temp_df.head())
    print("\nPrecipitation DataFrame First Few Rows:")
    print(precip_df.head())

    # Define stations with January temperatures
    stations = {
        'manchester': {
            'name': 'MANCHESTER RACECOURSE',
            'grid_id': 'AX-71',
            'temp_start': 1,
            'precip_start': 0,
            'january_temp': 5.0  # Adding January temperature
        },
        'bury': {
            'name': 'BURY MANCHESTER',
            'grid_id': 'AX-70',
            'temp_start': 17,
            'precip_start': 16,
            'january_temp': 3.8  # Adding January temperature
        },
        'rochdale': {
            'name': 'ROCHDALE',
            'grid_id': 'AY-70',
            'temp_start': 34,
```

```
            'precip_start': 31,
            'january_temp': 3.6  # Adding January temperature
        }
    }
    months = ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', 'December']

    all_data = []

    for station in stations.values():
        try:
            # Debug: Print exact locations we're trying to extract from
            print(f"\nProcessing {station['name']}:")
            print("Temperature slice:", temp_df.iloc[station['temp_start']+4:sta
            print("Precipitation slice:", precip_df.iloc[station['precip_start']

            # Create monthly data including January
            monthly_data = pd.DataFrame({
                'Month': months,
                'Temperature_C': [
                    station['january_temp'],  # January temperature
                    *[float(x) if pd.notna(x) and x != '' else np.nan
                      for x in temp_df.iloc[station['temp_start']+4:station['tem
                ],
                'Precipitation_mm': [
                    float(precip_df.iloc[station['precip_start']+2, 1]),  # Janu
                    *[float(x) if pd.notna(x) and x != '' else np.nan
                      for x in precip_df.iloc[station['precip_start']+3:station[
                ]
            })

            monthly_data['Station'] = station['name']
            monthly_data['Grid_ID'] = station['grid_id']
            all_data.append(monthly_data)

        except Exception as e:
            print(f"Error processing {station['name']}: {e}")
            # Optionally, you can raise the exception to stop processing
            # raise

    if not all_data:
        raise ValueError("No data could be processed")

    # Combine all stations
    combined_data = pd.concat(all_data, ignore_index=True)

    # Ensure months are in correct order
    combined_data['Month'] = pd.Categorical(combined_data['Month'], categories=m
    combined_data = combined_data.sort_values(['Station', 'Month']).reset_index(

    # Print summary
    print("\nProcessed Weather Data Summary (1991-2020):")
    print("========================================")
    print("Monthly averages from HadUK gridded data")
    print("Temperature: °C (12km British National Grid)")
    print("Precipitation: mm (2km British National Grid)")

    for station in stations.values():
        print(f"\n{station['name']} (Grid ID: {station['grid_id']}):")
        station_data = combined_data[combined_data['Station'] == station['name']
```

```
        pd.set_option('display.float_format', '{:.1f}'.format)
        print(station_data[['Month', 'Temperature_C', 'Precipitation_mm']])

    return combined_data

# Run the processing
cleaned_data = clean_met_office_data()
```

```
Temperature DataFrame First Few Rows:
           TEMPERATION Unnamed: 1
0                  NaN        NaN
1  MANCHESTER RECOURSE        NaN
2              GRID_ID      AX-71
3          tas Jaunary          5
4         tas February        5.4

Precipitation DataFrame First Few Rows:
    PRECIPITATION DATA Unnamed: 1
0  MANCHESTER RECOURSE        NaN
1              GRID_ID      AX-71
2           pr January         90
3          pr February         76
4             pr March         66

Processing MANCHESTER RACECOURSE:
Temperature slice: [7, 9.4, 12.4, 15, 16.8, 16.5, 14.2, 11, 7.6, 5.3, nan]
Precipitation slice: [90, 76, 66, 59, 64, 77, 84, 85, 85, 101, 97, 108, nan]

Processing BURY MANCHESTER:
Temperature slice: [4.1, 5.7, 8.1, 11, 13.6, 15.5, 15.2, 12.9, 9.7, 6.5, 4.1]
Precipitation slice: [131, 112, 95, 79, 83, 93, 100, 111, 110, 134, 138, 157, na
n]

Processing ROCHDALE:
Temperature slice: [3.9, 5.4, 7.9, 10.7, 13.4, 15.3, 15.1, 12.8, 9.6, 6.2, 4]
Precipitation slice: ['AY-70', 131, 110, 96, 77, 77, 92, 105, 110, 109, 130, 136,
154]
Error processing ROCHDALE: could not convert string to float: 'AY-70'

Processed Weather Data Summary (1991-2020):
==========================================
Monthly averages from HadUK gridded data
Temperature: °C (12km British National Grid)
Precipitation: mm (2km British National Grid)

MANCHESTER RACECOURSE (Grid ID: AX-71):
        Month  Temperature_C  Precipitation_mm
12    January            5.0              90.0
13   February            7.0              76.0
14      March            9.4              66.0
15      April           12.4              59.0
16        May           15.0              64.0
17       June           16.8              77.0
18       July           16.5              84.0
19     August           14.2              85.0
20  September           11.0              85.0
21    October            7.6             101.0
22   November            5.3              97.0
23   December            NaN             108.0

BURY MANCHESTER (Grid ID: AX-70):
        Month  Temperature_C  Precipitation_mm
0     January            3.8             131.0
1    February            4.1             112.0
2       March            5.7              95.0
3       April            8.1              79.0
4         May           11.0              83.0
5        June           13.6              93.0
```

```
6         July          15.5           100.0
7       August          15.2           111.0
8    September          12.9           110.0
9      October           9.7           134.0
10    November           6.5           138.0
11    December           4.1           157.0

ROCHDALE (Grid ID: AY-70):
Empty DataFrame
Columns: [Month, Temperature_C, Precipitation_mm]
Index: []
```

DATA CLEANING AND PREPROCESSING

In [33]:

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

def clean_met_office_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WE

    # Read the files
    temp_df = pd.read_excel(os.path.join(project_path, 'TEMPERATURE.xlsx'))
    precip_df = pd.read_excel(os.path.join(project_path, 'PRECIPITATION.xlsx'))

    # Define stations with January temperatures
    stations = {
        'manchester': {
            'name': 'MANCHESTER RACECOURSE',
            'grid_id': 'AX-71',
            'temp_start': 1,
            'precip_start': 0,
            'january_temp': 5.0,
            'january_precip': 90.0
        },
        'bury': {
            'name': 'BURY MANCHESTER',
            'grid_id': 'AX-70',
            'temp_start': 17,
            'precip_start': 16,
            'january_temp': 3.8,
            'january_precip': 131.0
        },
        'rochdale': {
            'name': 'ROCHDALE',
            'grid_id': 'AY-70',
            'temp_start': 34,
            'precip_start': 31,
            'january_temp': 3.6,
            'january_precip': 131.0  # Manually added from the data
        }
    }
    months = ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', 'December']

    all_data = []

    for station in stations.values():
        # Adjusting row indices based on actual data structure
```

```
        try:
            monthly_data = pd.DataFrame({
                'Month': months,
                'Temperature_C': [
                    station['january_temp'],  # January temperature
                    *[float(x) if pd.notna(x) and x != '' else np.nan
                      for x in temp_df.iloc[station['temp_start']+4:station['tem
                ],
                'Precipitation_mm': [
                    station['january_precip'],  # January precipitation
                    *[float(x) if pd.notna(x) and x != '' and x != station['grid
                      for x in precip_df.iloc[station['precip_start']+3:station[
                ]
            })

            monthly_data['Station'] = station['name']
            monthly_data['Grid_ID'] = station['grid_id']
            all_data.append(monthly_data)

        except Exception as e:
            print(f"Error processing {station['name']}: {e}")

    if not all_data:
        raise ValueError("No data could be processed")

    # Combine all stations
    combined_data = pd.concat(all_data, ignore_index=True)

    # Ensure months are in correct order
    combined_data['Month'] = pd.Categorical(combined_data['Month'], categories=m
    combined_data = combined_data.sort_values(['Station', 'Month']).reset_index(

    # Print summary
    print("\nProcessed Weather Data Summary (1991-2020):")
    print("==========================================")
    print("Monthly averages from HadUK gridded data")
    print("Temperature: °C (12km British National Grid)")
    print("Precipitation: mm (2km British National Grid)")

    for station in stations.values():
        print(f"\n{station['name']} (Grid ID: {station['grid_id']}):")
        station_data = combined_data[combined_data['Station'] == station['name']
        pd.set_option('display.float_format', '{:.1f}'.format)
        print(station_data[['Month', 'Temperature_C', 'Precipitation_mm']])

    return combined_data

# Run the processing
cleaned_data = clean_met_office_data()
```

```
Processed Weather Data Summary (1991-2020):
==========================================
Monthly averages from HadUK gridded data
Temperature: °C (12km British National Grid)
Precipitation: mm (2km British National Grid)

MANCHESTER RACECOURSE (Grid ID: AX-71):
        Month  Temperature_C  Precipitation_mm
12    January            5.0              90.0
13   February            7.0              76.0
14      March            9.4              66.0
15      April           12.4              59.0
16        May           15.0              64.0
17       June           16.8              77.0
18       July           16.5              84.0
19     August           14.2              85.0
20  September           11.0              85.0
21    October            7.6             101.0
22   November            5.3              97.0
23   December            NaN             108.0

BURY MANCHESTER (Grid ID: AX-70):
        Month  Temperature_C  Precipitation_mm
0     January            3.8             131.0
1    February            4.1             112.0
2       March            5.7              95.0
3       April            8.1              79.0
4         May           11.0              83.0
5        June           13.6              93.0
6        July           15.5             100.0
7      August           15.2             111.0
8   September           12.9             110.0
9     October            9.7             134.0
10   November            6.5             138.0
11   December            4.1             157.0

ROCHDALE (Grid ID: AY-70):
        Month  Temperature_C  Precipitation_mm
24    January            3.6             131.0
25   February            3.9             131.0
26      March            5.4             110.0
27      April            7.9              96.0
28        May           10.7              77.0
29       June           13.4              77.0
30       July           15.3              92.0
31     August           15.1             105.0
32  September           12.8             110.0
33    October            9.6             109.0
34   November            6.2             130.0
35   December            4.0             136.0
```

In [34]:

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

def clean_met_office_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WE

    # Read the files
    temp_df = pd.read_excel(os.path.join(project_path, 'TEMPERATURE.xlsx'))
```

```
    precip_df = pd.read_excel(os.path.join(project_path, 'PRECIPITATION.xlsx'))

    # Define stations with January temperatures
    stations = {
        'manchester': {
            'name': 'MANCHESTER RACECOURSE',
            'grid_id': 'AX-71',
            'temp_start': 1,
            'precip_start': 0,
            'january_temp': 5.0,
            'january_precip': 90.0
        },
        'bury': {
            'name': 'BURY MANCHESTER',
            'grid_id': 'AX-70',
            'temp_start': 17,
            'precip_start': 16,
            'january_temp': 3.8,
            'january_precip': 131.0
        },
        'rochdale': {
            'name': 'ROCHDALE',
            'grid_id': 'AY-70',
            'temp_start': 34,
            'precip_start': 31,
            'january_temp': 3.6,
            'january_precip': 131.0  # Manually added from the data
        }
    }
    months = ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', 'December']

    all_data = []

    for station in stations.values():
        # Adjusting row indices based on actual data structure
        try:
            monthly_data = pd.DataFrame({
                'Month': months,
                'Temperature_C': [
                    station['january_temp'],  # January temperature
                    *[float(x) if pd.notna(x) and x != '' else np.nan
                      for x in temp_df.iloc[station['temp_start']+4:station['tem
                ],
                'Precipitation_mm': [
                    station['january_precip'],  # January precipitation
                    *[float(x) if pd.notna(x) and x != '' and x != station['grid
                      for x in precip_df.iloc[station['precip_start']+3:station[
                ]
            })

            monthly_data['Station'] = station['name']
            monthly_data['Grid_ID'] = station['grid_id']
            all_data.append(monthly_data)

        except Exception as e:
            print(f"Error processing {station['name']}: {e}")

    if not all_data:
        raise ValueError("No data could be processed")
```

```
    # Combine all stations
    combined_data = pd.concat(all_data, ignore_index=True)

    # Ensure months are in correct order
    combined_data['Month'] = pd.Categorical(combined_data['Month'], categories=m
    combined_data = combined_data.sort_values(['Station', 'Month']).reset_index(

    # Print summary
    print("\nProcessed Weather Data Summary (1991-2020):")
    print("===========================================")
    print("Monthly averages from HadUK gridded data")
    print("Temperature: °C (12km British National Grid)")
    print("Precipitation: mm (2km British National Grid)")

    for station in stations.values():
        print(f"\n{station['name']} (Grid ID: {station['grid_id']}):")
        station_data = combined_data[combined_data['Station'] == station['name']]
        pd.set_option('display.float_format', '{:.1f}'.format)
        print(station_data[['Month', 'Temperature_C', 'Precipitation_mm']])

    # Save to CSV
    output_path = os.path.join(project_path, 'cleaned_data', 'monthly_weather_da

    # Create the directory if it doesn't exist
    os.makedirs(os.path.dirname(output_path), exist_ok=True)

    # Save the DataFrame to CSV
    combined_data.to_csv(output_path, index=False)
    print(f"\nData saved to {output_path}")

    return combined_data

# Run the processing
cleaned_data = clean_met_office_data()
```

```
Processed Weather Data Summary (1991-2020):
==========================================
Monthly averages from HadUK gridded data
Temperature: °C (12km British National Grid)
Precipitation: mm (2km British National Grid)

MANCHESTER RACECOURSE (Grid ID: AX-71):
        Month  Temperature_C  Precipitation_mm
12    January            5.0              90.0
13   February            7.0              76.0
14      March            9.4              66.0
15      April           12.4              59.0
16        May           15.0              64.0
17       June           16.8              77.0
18       July           16.5              84.0
19     August           14.2              85.0
20  September           11.0              85.0
21    October            7.6             101.0
22   November            5.3              97.0
23   December            NaN             108.0

BURY MANCHESTER (Grid ID: AX-70):
        Month  Temperature_C  Precipitation_mm
0     January            3.8             131.0
1    February            4.1             112.0
2       March            5.7              95.0
3       April            8.1              79.0
4         May           11.0              83.0
5        June           13.6              93.0
6        July           15.5             100.0
7      August           15.2             111.0
8   September           12.9             110.0
9     October            9.7             134.0
10   November            6.5             138.0
11   December            4.1             157.0

ROCHDALE (Grid ID: AY-70):
        Month  Temperature_C  Precipitation_mm
24    January            3.6             131.0
25   February            3.9             131.0
26      March            5.4             110.0
27      April            7.9              96.0
28        May           10.7              77.0
29       June           13.4              77.0
30       July           15.3              92.0
31     August           15.1             105.0
32  September           12.8             110.0
33    October            9.6             109.0
34   November            6.2             130.0
35   December            4.0             136.0

Data saved to C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WEATHER\cl
eaned_data\monthly_weather_data.csv
```

In [35]:

```python
import os
import pandas as pd

# Set the path to the historical data directory
historical_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HIS
```

```
# List all files and directories in the historical data folder
print("Contents of Historical Data Directory:")
for root, dirs, files in os.walk(historical_path):
    level = root.replace(historical_path, '').count(os.sep)
    indent = ' ' * 4 * level
    print(f"{indent}{os.path.basename(root)}/")
    subindent = ' ' * 4 * (level + 1)
    for file in files:
        print(f"{subindent}{file}")
```

```
Contents of Historical Data Directory:
HISTORICAL DATA/
    BURY STATION/
        69044_cdr.csv
        69044_gdf.csv
        BURY RIVER PEAK.xlsx
        ~$BURY RIVER PEAK.xlsx
    MANCHESTER RACECOURSE/
        MACHESTER RACECOURSE RIVER PEAK.xlsx
        ~$MACHESTER RACECOURSE RIVER PEAK.xlsx
    ROCHDALE/
        69803_cdr.csv
        69803_gdf.csv
        ROCHDALE RIVER PEAK.xlsx
        ~$ROCHDALE RIVER PEAK.xlsx
```

In [37]:

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

def clean_river_peak_data(file_path):
    """
    Clean and process river peak flow data from Excel files
    """
    # Read the Excel file, skipping initial empty rows
    df = pd.read_excel(file_path, header=None)

    # Find the header row (the row with 'Rank' and 'Water Year')
    header_row = df[df.iloc[:, 0].isin(['Rank'])].index[0]

    # Re-read the file with the correct header
    df = pd.read_excel(file_path, header=header_row)

    # Clean up the columns
    df.columns = [col.strip() if isinstance(col, str) else f'Unnamed_{i}' for i,

    # Select and rename relevant columns
    columns_to_keep = {
        'Rank': 'Rank',
        'Water Year': 'Water_Year',
        'Date': 'Date',
        'Time': 'Time',
        'Stage (m)': 'Stage_m',
        'Flow (m3/s)': 'Flow_m3s',
        'Source': 'Source',
        'Ref': 'Reference',
        'Comments': 'Comments'
    }

    # Select and rename columns that exist in the dataframe
```

```
    selected_columns = {col: alias for col, alias in columns_to_keep.items() if
    df_clean = df[list(selected_columns.keys())].copy()
    df_clean.rename(columns=selected_columns, inplace=True)

    # Convert Date to datetime
    if 'Date' in df_clean.columns:
        df_clean['Date'] = pd.to_datetime(df_clean['Date'], errors='coerce')

    # Convert numeric columns
    numeric_columns = ['Rank', 'Stage_m', 'Flow_m3s']
    for col in numeric_columns:
        if col in df_clean.columns:
            df_clean[col] = pd.to_numeric(df_clean[col], errors='coerce')

    return df_clean

def process_all_peak_flow_files():
    """
    Process peak flow data for all stations
    """
    historical_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION

    # Define the peak flow files for each station
    peak_files = {
        'Bury': os.path.join(historical_path, 'BURY STATION', 'BURY RIVER PEAK.x
        'Manchester': os.path.join(historical_path, 'MANCHESTER RACECOURSE', 'MA
        'Rochdale': os.path.join(historical_path, 'ROCHDALE', 'ROCHDALE RIVER PE
    }

    # Store processed dataframes
    processed_data = {}

    # Process each file
    for station, file_path in peak_files.items():
        try:
            df_clean = clean_river_peak_data(file_path)

            # Add station name column
            df_clean['Station'] = station

            # Store processed dataframe
            processed_data[station] = df_clean

            # Print summary for each station
            print(f"\n{station} Station Peak Flow Data:")
            print("=" * (len(station) + 20))
            print(f"Total records: {len(df_clean)}")
            print("\nData Summary:")
            print(df_clean.describe())

            # Print date range
            if 'Date' in df_clean.columns:
                print("\nDate Range:")
                print(f"Earliest date: {df_clean['Date'].min()}")
                print(f"Latest date: {df_clean['Date'].max()}")

        except Exception as e:
            print(f"Error processing {station} station data: {e}")

    # Combine all stations' data
```

```
    combined_data = pd.concat(processed_data.values(), ignore_index=True)

    # Save to CSV
    output_path = os.path.join(historical_path, 'processed_peak_flow_data.csv')
    combined_data.to_csv(output_path, index=False)
    print("\nCombined data saved to:", output_path)

    return processed_data, combined_data

# Run the processing
station_data, combined_data = process_all_peak_flow_files()
```

```
Bury Station Peak Flow Data:
========================
Total records: 52

Data Summary:
       Rank                           Date  Stage_m  Flow_m3s  Comments
count  50.0                             51     51.0      51.0       0.0
mean   25.5  1998-01-25 01:24:42.352941184      1.4     115.9       NaN
min     1.0            1973-01-12 00:00:00      1.1      51.5       NaN
25%    13.2            1985-05-28 12:00:00      1.3      84.6       NaN
50%    25.5            1998-01-08 00:00:00      1.4     112.9       NaN
75%    37.8            2010-04-26 12:00:00      1.5     125.6       NaN
max    50.0            2023-07-23 00:00:00      2.2     283.6       NaN
std    14.6                            NaN      0.2      43.6       NaN

Date Range:
Earliest date: 1973-01-12 00:00:00
Latest date: 2023-07-23 00:00:00

Manchester Station Peak Flow Data:
==============================
Total records: 83

Data Summary:
       Rank                           Date  Stage_m  Flow_m3s
count  82.0                             82     82.0      82.0
mean   41.4  1982-08-01 18:43:54.146341440      3.5     279.4
min     1.0            1941-10-24 00:00:00      2.5     135.0
25%    21.2            1962-04-11 18:00:00      3.1     217.3
50%    41.5            1982-06-08 12:00:00      3.5     273.5
75%    61.8            2002-11-10 06:00:00      3.8     327.3
max    81.0            2023-01-10 00:00:00      5.7     560.0
std    23.8                            NaN      0.6      87.4

Date Range:
Earliest date: 1941-10-24 00:00:00
Latest date: 2023-01-10 00:00:00

Rochdale Station Peak Flow Data:
============================
Total records: 32

Data Summary:
       Rank                           Date  Stage_m  Flow_m3s  Comments
count  30.0                             31     31.0      31.0       0.0
mean   15.5  2008-02-06 15:29:01.935483904      1.4      46.4       NaN
min     1.0            1993-09-13 00:00:00      0.8      18.0       NaN
25%     8.2            2000-10-08 12:00:00      1.3      38.1       NaN
50%    15.0            2008-01-21 00:00:00      1.4      44.7       NaN
75%    22.8            2015-08-13 00:00:00      1.5      51.3       NaN
max    30.0            2023-07-23 00:00:00      2.2      92.8       NaN
std     8.8                            NaN      0.3      15.0       NaN

Date Range:
Earliest date: 1993-09-13 00:00:00
Latest date: 2023-07-23 00:00:00

Combined data saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION
\HISTORICAL DATA\processed_peak_flow_data.csv
```

In [39]:
```
import pandas as pd
import os

def detailed_csv_analysis(filepath):
    # Read the entire CSV file
    df = pd.read_csv(filepath)

    print(f"\nDetailed Analysis of {os.path.basename(filepath)}:")
    print("=" * 50)

    # Print full dataframe to see the structure
    print("\nFull DataFrame:")
    print(df)

    # Attempt to reshape the data
    # Group by 'file' and 'timestamp'
    grouped = df.groupby(['file', 'timestamp'])

    print("\nGrouped Data Structure:")
    for (file, timestamp), group in grouped:
        print(f"\nFile: {file}, Timestamp: {timestamp}")
        print("Group contents:")
        print(group)
        break  # Just show the first group to understand structure

    # Find unique entries for each category
    print("\nUnique Entries:")
    for column in df.columns:
        print(f"{column} unique entries:")
        print(df[column].unique())

# Paths to the CSV files
bury_station_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\H
cdr_file = os.path.join(bury_station_path, '69044_cdr.csv')
gdf_file = os.path.join(bury_station_path, '69044_gdf.csv')

# Analyze both files
print("Analyzing CDR File:")
detailed_csv_analysis(cdr_file)

print("\n\nAnalyzing GDF File:")
detailed_csv_analysis(gdf_file)
```

```
Analyzing CDR File:

Detailed Analysis of 69044_cdr.csv:
==================================================

Full DataFrame:
             file      timestamp             2025-01-30T20:48:59
0        database             id                  nrfa-public-31
1        database           name  UK National River Flow Archive
2         station             id                           69044
3         station           name           Irwell at Bury Ground
4         station  gridReference                    SD7998711393
...           ...            ...                             ...
20833  2017-12-27          0.000                            3000
20834  2017-12-28          2.400                            3000
20835  2017-12-29         18.100                            3000
20836  2017-12-30          5.800                            3000
20837  2017-12-31          7.100                            3000

[20838 rows x 3 columns]

Grouped Data Structure:

File: 1961-01-01, Timestamp: 9.400
Group contents:
          file timestamp 2025-01-30T20:48:59
19  1961-01-01     9.400                1000

Unique Entries:
file unique entries:
['database' 'station' 'dataType' ... '2017-12-29' '2017-12-30'
 '2017-12-31']
timestamp unique entries:
['id' 'name' 'gridReference' 'descriptionSummary' 'descriptionGeneral'
 'descriptionStationHydrometry' 'descriptionFlowRecord'
 'descriptionCatchment' 'descriptionFlowRegime' 'parameter' 'units'
 'period' 'measurementType' 'first' 'last' '9.400' '13.700' '3.000'
 '0.100' '13.000' '0.000' '16.900' '7.500' '5.700' '3.600' '16.700'
 '5.000' '0.300' '0.800' '9.200' '17.200' '14.300' '2.000' '0.200' '9.900'
 '4.600' '12.600' '3.400' '7.200' '10.200' '6.900' '1.700' '14.400'
 '11.500' '8.500' '11.100' '3.300' '7.300' '0.700' '3.900' '0.400' '9.500'
 '1.600' '7.000' '3.200' '0.600' '1.100' '0.500' '1.000' '1.500' '1.200'
 '5.600' '20.700' '5.800' '5.400' '10.400' '3.500' '7.600' '6.000' '5.500'
 '15.200' '4.400' '10.100' '10.500' '2.300' '8.600' '10.600' '2.100'
 '15.100' '1.300' '28.700' '3.800' '2.900' '1.900' '8.200' '6.600'
 '29.100' '6.800' '10.000' '12.100' '6.700' '38.500' '28.600' '39.500'
 '1.400' '4.500' '0.900' '4.700' '9.600' '27.100' '15.000' '11.700'
 '8.400' '2.600' '11.200' '1.800' '4.000' '13.400' '19.700' '11.900'
 '5.100' '5.300' '20.300' '6.300' '4.300' '15.700' '2.800' '34.900'
 '7.800' '3.700' '13.600' '18.200' '4.100' '2.200' '7.700' '37.400'
 '27.900' '11.600' '22.600' '21.400' '11.800' '20.500' '15.600' '12.700'
 '12.500' '4.200' '23.500' '2.500' '22.800' '9.300' '2.700' '17.900'
 '11.000' '6.100' '41.300' '3.100' '24.100' '17.700' '14.800' '13.300'
 '20.900' '2.400' '6.400' '30.500' '9.700' '13.500' '19.800' '15.300'
 '4.900' '35.100' '42.400' '19.400' '32.500' '6.200' '22.400' '4.800'
 '17.000' '21.000' '10.300' '8.000' '18.000' '17.800' '16.200' '14.600'
 '18.600' '7.100' '6.500' '7.400' '18.500' '10.700' '8.100' '23.300'
 '15.900' '19.600' '17.600' '14.000' '14.100' '8.900' '13.900' '5.900'
 '31.500' '39.900' '9.100' '25.400' '8.300' '11.400' '26.400' '31.400'
 '9.800' '16.000' '26.300' '20.100' '13.200' '11.300' '23.000' '36.800'
```

```
 '49.600' '5.200' '19.100' '21.300' '17.500' '18.900' '12.000' '44.500'
 '32.600' '49.100' '26.900' '22.300' '15.500' '16.100' '8.800' '24.700'
 '12.200' '19.300' '22.100' '8.700' '10.900' '16.500' '16.300' '13.800'
 '25.000' '28.800' '21.500' '10.800' '23.600' '17.400' '23.100' '14.500'
 '14.700' '19.500' '27.300' '30.900' '36.100' '14.200' '44.900' '16.800'
 '24.000' '12.900' '36.900' '13.100' '22.000' '38.700' '12.400' '7.900'
 '33.700' '39.300' '9.000' '21.700' '12.300' '23.200' '28.100' '26.500'
 '51.300' '36.700' '22.200' '20.200' '22.900' '31.800' '34.000' '26.200'
 '41.900' '20.600' '30.200' '17.300' '31.100' '32.000' '18.700' '16.400'
 '12.800' '34.800' '33.900' '79.500' '24.200' '47.900' '15.400' '30.300'
 '26.100' '19.200' '40.700' '16.600' '30.100' '31.900' '33.400' '20.800'
 '21.200' '25.900' '14.900' '31.200' '21.900' '25.100' '25.700' '35.200'
 '21.800' '29.400' '33.100' '28.200' '22.500' '29.900' '15.800' '28.400'
 '58.200' '20.000' '27.700' '18.300' '17.100' '32.300' '34.600' '27.400'
 '29.000' '24.800' '27.200' '21.100' '34.300' '23.400' '36.400' '25.300'
 '18.400' '24.900' '43.100' '25.800' '23.800' '56.200' '30.800' '33.500'
 '23.900' '36.200' '28.300' '23.700' '29.500' '31.300' '28.900' '32.400'
 '21.600' '35.800' '25.500' '24.500' '26.600' '40.800' '32.200' '39.800'
 '47.400' '32.900' '24.300' '57.900' '35.900' '28.500' '18.100' '20.400'
 '24.400' '35.300' '47.500' '25.600' '24.600' '38.200' '36.300' '51.100'
 '28.000' '39.600' '32.100' '33.300' '18.800' '38.900' '57.500' '19.000'
 '34.400' '27.000' '40.000' '26.000' '27.800' '35.700' '29.800' '29.700'
 '31.700' '26.700' '25.200' '29.300' '30.400' '43.600' '40.100' '19.900'
 '31.000' '40.500' '30.700' '33.800' '38.100' '33.600' '41.800' '27.600'
 '37.900' '37.800' '42.100' '42.200' '44.200' '41.000' '35.400' '46.700'
 '34.100' '22.700' '39.000' '37.300' '34.700' '33.000' '29.600' '37.500'
 '64.400' '44.400' '30.600' '48.800' '46.400' '34.500' '29.200' '79.000'
 '35.000' '30.000']
2025-01-30T20:48:59 unique entries:
['nrfa-public-31' 'UK National River Flow Archive' '69044'
 'Irwell at Bury Ground' 'SD7998711393'
 'Velocity area station with broad-crested weir as control. Replaced Bury Bridge
(69035) in 1995.'
 'Velocity area station, 22m wide section with good approach, opened November 199
5 with a pre-existing curved broad-crested mill weir as the control. Replaced Bur
y Bridge (69035), 1.5km downstream; Kirkless Brook enters between the two station
s. Bury Bridge and Bury Grounds overlapped from November 1995 to March 1998, and
included high flows in November 1996. Stages are closely correlated.'
 "The curved weir is 28m wide, crest is in poor condition. Weir doesn't drown due
to 3m drop over crest. The inlet pipe was extended to prevent silt blockage, and
the level re-surveyed. Level readings prior to August 1998 were then lowered by
0.022m to make them consistent with later records. High flow gaugings carried out
by cableway. A stage-stage relationship was derived to generate equivalent stages
at Bury Grounds for the period of record at Bury Bridge, and the present Bury Gro
unds rating applied. One peak flow rating applied across period of record, derive
d from current meter gaugings."
 'POT and AMAX data are presented for Bury Grounds including the period of record
at Bury Bridge. Full period of record peak flow data reviewed and released in Sep
tember 2019 (WINFAP Files v8).'
 'Geology is post-glacial deposits over predominantly Millstone Grit, with some C
oal Measures. A moderately urbanised catchment with steep moorland headwaters in
the south Pennines; includes urban areas of Bury and Rawtenstall. No catchment ch
anges known.'
 'Runoff influenced by storage reservoirs (Haslingden Grane system and Ogden, Clo
wbridge), abstractions and effluent returns.'
 'cdr' 'Catchment Daily Rainfall' 'Rainfall' 'mm' 'day (P1D)'
 'Accumulation' '1961-01-01' '2017-12-31' '1000' '2000' '3000' '4000']


Analyzing GDF File:
```

```
Detailed Analysis of 69044_gdf.csv:
==================================================

Full DataFrame:
             file      timestamp            2025-01-30T20:48:40
0        database             id                 nrfa-public-31
1        database           name  UK National River Flow Archive
2         station             id                          69044
3         station           name          Irwell at Bury Ground
4         station  gridReference                   SD7998711393
...           ...            ...                            ...
10189  2023-09-26          2.439                            NaN
10190  2023-09-27          2.769                            NaN
10191  2023-09-28          2.562                            NaN
10192  2023-09-29          2.277                            NaN
10193  2023-09-30          6.730                            NaN

[10194 rows x 3 columns]

Grouped Data Structure:

File: 1995-11-22, Timestamp: 0.897
Group contents:
          file timestamp 2025-01-30T20:48:40
19  1995-11-22     0.897                 NaN

Unique Entries:
file unique entries:
['database' 'station' 'dataType' ... '2023-09-28' '2023-09-29'
 '2023-09-30']
timestamp unique entries:
['id' 'name' 'gridReference' ... '3.602' '2.439' '2.277']
2025-01-30T20:48:40 unique entries:
['nrfa-public-31' 'UK National River Flow Archive' '69044'
 'Irwell at Bury Ground' 'SD7998711393'
 'Velocity area station with broad-crested weir as control. Replaced Bury Bridge
(69035) in 1995.'
 'Velocity area station, 22m wide section with good approach, opened November 199
5 with a pre-existing curved broad-crested mill weir as the control. Replaced Bur
y Bridge (69035), 1.5km downstream; Kirkless Brook enters between the two station
s. Bury Bridge and Bury Grounds overlapped from November 1995 to March 1998, and
included high flows in November 1996. Stages are closely correlated.'
 "The curved weir is 28m wide, crest is in poor condition. Weir doesn't drown due
to 3m drop over crest. The inlet pipe was extended to prevent silt blockage, and
the level re-surveyed. Level readings prior to August 1998 were then lowered by
0.022m to make them consistent with later records. High flow gaugings carried out
by cableway. A stage-stage relationship was derived to generate equivalent stages
at Bury Grounds for the period of record at Bury Bridge, and the present Bury Gro
unds rating applied. One peak flow rating applied across period of record, derive
d from current meter gaugings."
 'POT and AMAX data are presented for Bury Grounds including the period of record
at Bury Bridge. Full period of record peak flow data reviewed and released in Sep
tember 2019 (WINFAP Files v8).'
 'Geology is post-glacial deposits over predominantly Millstone Grit, with some C
oal Measures. A moderately urbanised catchment with steep moorland headwaters in
the south Pennines; includes urban areas of Bury and Rawtenstall. No catchment ch
anges known.'
 'Runoff influenced by storage reservoirs (Haslingden Grane system and Ogden, Clo
wbridge), abstractions and effluent returns.'
```

```
'gdf' 'Gauged Daily Flow' 'Flow' 'm3/s' 'day (P1D)' 'Mean' '1995-11-22'
'2023-09-30' nan 'M']
```

In [47]:

```python
import pandas as pd
import os
import matplotlib.pyplot as plt

def process_nrfa_data(filepath):
    """
    Process NRFA data files with comprehensive error handling
    """
    print(f"\nProcessing file: {os.path.basename(filepath)}")
    print("=" * 50)

    # Read the entire file
    df = pd.read_csv(filepath)

    # Print full dataframe details
    print("\nFull DataFrame Structure:")
    print(df)

    print("\nColumn Names:")
    print(df.columns)

    print("\nFirst few rows:")
    print(df.head())

    # Investigate data rows
    print("\nData Rows Investigation:")
    for index, row in df.iterrows():
        print(f"\nRow {index}:")
        print(row)

        # Try to find actual data rows
        if isinstance(row.iloc[0], str) and row.iloc[0] not in ['database', 'sta
            print("Potential data row found!")
            break

def analyze_bury_station_data():
    """
    Analyze all data files in the Bury Station folder
    """
    # Path to Bury Station folder
    bury_station_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTI

    # Files to process (only CSV files)
    data_files = [f for f in os.listdir(bury_station_path) if f.endswith('.csv')

    # Process each file
    processed_data = {}
    for filename in data_files:
        filepath = os.path.join(bury_station_path, filename)
        processed_data[filename] = process_nrfa_data(filepath)

    return processed_data

# Run the analysis
bury_data = analyze_bury_station_data()
```

```
Processing file: 69044_cdr.csv
==================================================

Full DataFrame Structure:
             file      timestamp             2025-01-30T20:48:59
0        database             id                 nrfa-public-31
1        database           name  UK National River Flow Archive
2         station             id                          69044
3         station           name            Irwell at Bury Ground
4         station  gridReference                   SD7998711393
...           ...            ...                            ...
20833  2017-12-27          0.000                           3000
20834  2017-12-28          2.400                           3000
20835  2017-12-29         18.100                           3000
20836  2017-12-30          5.800                           3000
20837  2017-12-31          7.100                           3000

[20838 rows x 3 columns]

Column Names:
Index(['file', 'timestamp', '2025-01-30T20:48:59'], dtype='object')

First few rows:
       file      timestamp             2025-01-30T20:48:59
0  database             id                 nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                          69044
3   station           name            Irwell at Bury Ground
4   station  gridReference                   SD7998711393

Data Rows Investigation:

Row 0:
file                           database
timestamp                            id
2025-01-30T20:48:59    nrfa-public-31
Name: 0, dtype: object

Row 1:
file                                           database
timestamp                                          name
2025-01-30T20:48:59    UK National River Flow Archive
Name: 1, dtype: object

Row 2:
file                    station
timestamp                    id
2025-01-30T20:48:59      69044
Name: 2, dtype: object

Row 3:
file                                  station
timestamp                                name
2025-01-30T20:48:59    Irwell at Bury Ground
Name: 3, dtype: object

Row 4:
file                           station
timestamp                gridReference
2025-01-30T20:48:59     SD7998711393
```

```
Name: 4, dtype: object

Row 5:
file                                                          station
timestamp                                          descriptionSummary
2025-01-30T20:48:59    Velocity area station with broad-crested weir ...
Name: 5, dtype: object

Row 6:
file                                                          station
timestamp                                          descriptionGeneral
2025-01-30T20:48:59    Velocity area station, 22m wide section with g...
Name: 6, dtype: object

Row 7:
file                                                          station
timestamp                                descriptionStationHydrometry
2025-01-30T20:48:59    The curved weir is 28m wide, crest is in poor ...
Name: 7, dtype: object

Row 8:
file                                                          station
timestamp                                       descriptionFlowRecord
2025-01-30T20:48:59    POT and AMAX data are presented for Bury Groun...
Name: 8, dtype: object

Row 9:
file                                                          station
timestamp                                        descriptionCatchment
2025-01-30T20:48:59    Geology is post-glacial deposits over predomin...
Name: 9, dtype: object

Row 10:
file                                                          station
timestamp                                       descriptionFlowRegime
2025-01-30T20:48:59    Runoff influenced by storage reservoirs (Hasli...
Name: 10, dtype: object

Row 11:
file                   dataType
timestamp                    id
2025-01-30T20:48:59         cdr
Name: 11, dtype: object
Potential data row found!

Processing file: 69044_gdf.csv
==================================================

Full DataFrame Structure:
             file      timestamp            2025-01-30T20:48:40
0        database             id                 nrfa-public-31
1        database           name  UK National River Flow Archive
2         station             id                          69044
3         station           name            Irwell at Bury Ground
4         station  gridReference                    SD7998711393
...           ...            ...                            ...
10189  2023-09-26          2.439                            NaN
10190  2023-09-27          2.769                            NaN
10191  2023-09-28          2.562                            NaN
10192  2023-09-29          2.277                            NaN
```

```
10193  2023-09-30          6.730                             NaN

[10194 rows x 3 columns]

Column Names:
Index(['file', 'timestamp', '2025-01-30T20:48:40'], dtype='object')

First few rows:
       file      timestamp             2025-01-30T20:48:40
0  database             id                  nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                           69044
3   station           name           Irwell at Bury Ground
4   station  gridReference                    SD7998711393

Data Rows Investigation:

Row 0:
file                            database
timestamp                             id
2025-01-30T20:48:40    nrfa-public-31
Name: 0, dtype: object

Row 1:
file                                       database
timestamp                                      name
2025-01-30T20:48:40    UK National River Flow Archive
Name: 1, dtype: object

Row 2:
file                   station
timestamp                   id
2025-01-30T20:48:40      69044
Name: 2, dtype: object

Row 3:
file                                 station
timestamp                               name
2025-01-30T20:48:40    Irwell at Bury Ground
Name: 3, dtype: object

Row 4:
file                         station
timestamp              gridReference
2025-01-30T20:48:40     SD7998711393
Name: 4, dtype: object

Row 5:
file                                                              station
timestamp                                              descriptionSummary
2025-01-30T20:48:40    Velocity area station with broad-crested weir ...
Name: 5, dtype: object

Row 6:
file                                                              station
timestamp                                              descriptionGeneral
2025-01-30T20:48:40    Velocity area station, 22m wide section with g...
Name: 6, dtype: object

Row 7:
```

```
file                                                               station
timestamp                                      descriptionStationHydrometry
2025-01-30T20:48:40    The curved weir is 28m wide, crest is in poor ...
Name: 7, dtype: object

Row 8:
file                                                               station
timestamp                                             descriptionFlowRecord
2025-01-30T20:48:40    POT and AMAX data are presented for Bury Groun...
Name: 8, dtype: object

Row 9:
file                                                               station
timestamp                                              descriptionCatchment
2025-01-30T20:48:40    Geology is post-glacial deposits over predomin...
Name: 9, dtype: object

Row 10:
file                                                               station
timestamp                                             descriptionFlowRegime
2025-01-30T20:48:40    Runoff influenced by storage reservoirs (Hasli...
Name: 10, dtype: object

Row 11:
file                   dataType
timestamp                    id
2025-01-30T20:48:40         gdf
Name: 11, dtype: object
Potential data row found!
```

In [51]:

```python
import os

# Path to the Bury Station folder
bury_station_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\H

# List all files in the directory
print("Files in the Bury Station folder:")
all_files = os.listdir(bury_station_path)
for file in all_files:
    print(f"- {file}")

# Print full file paths
print("\nFull file paths:")
for file in all_files:
    full_path = os.path.join(bury_station_path, file)
    print(f"- {full_path}")
    # Print file size
    print(f"  Size: {os.path.getsize(full_path)} bytes")
```

```
Files in the Bury Station folder:
- 69044_cdr.csv
- 69044_gdf.csv
- BURY RIVER PEAK.xlsx
- ~$BURY RIVER PEAK.xlsx

Full file paths:
- C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\BURY S
TATION\69044_cdr.csv
  Size: 426866 bytes
- C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\BURY S
TATION\69044_gdf.csv
  Size: 186697 bytes
- C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\BURY S
TATION\BURY RIVER PEAK.xlsx
  Size: 12466 bytes
- C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\BURY S
TATION\~$BURY RIVER PEAK.xlsx
  Size: 165 bytes
```

In [53]:

```
import os
import pandas as pd

def explore_historical_data():
    base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTO
    stations = {
        'bury': 'BURY STATION',
        'manchester': 'MANCHESTER RACECOURSE',
        'rochdale': 'ROCHDALE'
    }

    # Check contents of one station folder
    bury_path = os.path.join(base_path, stations['bury'])
    print("\nContents of Bury Station folder:")
    for file in os.listdir(bury_path):
        print(f"- {file}")
        file_path = os.path.join(bury_path, file)
        if file.endswith('.csv'):
            df = pd.read_csv(file_path, nrows=5)
            print("\nFirst 5 rows:")
            print(df)
            print("\nColumns:")
            print(df.columns.tolist())

explore_historical_data()
```

```
Contents of Bury Station folder:
- 69044_cdr.csv

First 5 rows:
       file      timestamp              2025-01-30T20:48:59
0  database             id                   nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                            69044
3   station           name             Irwell at Bury Ground
4   station  gridReference                     SD7998711393

Columns:
['file', 'timestamp', '2025-01-30T20:48:59']
- 69044_gdf.csv

First 5 rows:
       file      timestamp              2025-01-30T20:48:40
0  database             id                   nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                            69044
3   station           name             Irwell at Bury Ground
4   station  gridReference                     SD7998711393

Columns:
['file', 'timestamp', '2025-01-30T20:48:40']
- BURY RIVER PEAK.xlsx
- ~$BURY RIVER PEAK.xlsx
```

In [59]:

```python
import pandas as pd
import os

def process_historical_data():
    base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTO

    def process_peak_data(df):
        # Find the row containing column headers
        header_row = df[df.iloc[:,1] == 'Water Year'].index[0]

        # Get data after headers
        data = df.iloc[header_row+1:].copy()
        data.columns = df.iloc[header_row]

        # Clean up columns
        cols = ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Ratin
        return data[cols].dropna(subset=['Date'])

    stations = {
        'rochdale': {'folder': 'ROCHDALE', 'id': '69803', 'has_cdr_gdf': True},
        'manchester': {'folder': 'MANCHESTER RACECOURSE', 'id': '69023', 'has_cd
        'bury': {'folder': 'BURY STATION', 'id': '69044', 'has_cdr_gdf': True}
    }

    for station, info in stations.items():
        print(f"\nProcessing {station.upper()}:")
        station_path = os.path.join(base_path, info['folder'])

        # Process CDR and GDF if available
        if info['has_cdr_gdf']:
            cdr = pd.read_csv(os.path.join(station_path, f"{info['id']}_cdr.csv"
            gdf = pd.read_csv(os.path.join(station_path, f"{info['id']}_gdf.csv"
```

```
            print(f"CDR records: {len(cdr)}")
            print(f"GDF records: {len(gdf)}")

        # Process peak data
        peak_files = [f for f in os.listdir(station_path) if 'PEAK' in f.upper()
                     and f.endswith('.xlsx') and not f.startswith('~')]
        if peak_files:
            df = pd.read_excel(os.path.join(station_path, peak_files[0]))
            peak_data = process_peak_data(df)
            print(f"\nPeak flow records: {len(peak_data)}")
            print("\nSample peak data:")
            print(peak_data.head())

process_historical_data()
```

```
Processing ROCHDALE:
CDR records: 750
GDF records: 11193

Peak flow records: 31

Sample peak data:
3 Water Year                 Date      Time Stage (m) Flow (m3/s)    Rating
5  1992-1993  1993-09-13 00:00:00  11:30:00       0.9        21.1  In Range
6  1993-1994  1993-12-08 00:00:00  23:45:00       1.3        38.3  In Range
7  1994-1995  1995-01-31 00:00:00  23:15:00       1.6        56.7  In Range
8  1995-1996  1996-02-18 00:00:00  03:15:00       0.8        18.0  In Range
9  1996-1997  1996-11-06 00:00:00  02:15:00       1.2        36.3  In Range

Processing MANCHESTER:

Peak flow records: 82

Sample peak data:
1 Water Year                 Date      Time Stage (m) Flow (m3/s)   Rating
3  1941-1942  1941-10-24 00:00:00  00:00:00       3.5         269  Extrap.
4  1942-1943  1942-10-17 00:00:00  00:00:00       3.2         223  Extrap.
5  1943-1944  1944-01-23 00:00:00  00:00:00       4.1         374  Extrap.
6  1944-1945  1945-02-02 00:00:00  00:00:00       3.9         339  Extrap.
7  1945-1946  1946-09-20 00:00:00  00:00:00       5.3         500  Extrap.

Processing BURY:
CDR records: 20840
GDF records: 10190

Peak flow records: 51

Sample peak data:
2 Water Year                 Date      Time Stage (m) Flow (m3/s)    Rating
4  1972-1973  1973-01-12 00:00:00  00:00:00       1.3        78.1       NaN
5  1973-1974  1974-02-11 00:00:00  00:00:00       1.5       118.0       NaN
6  1974-1975  1975-01-21 00:00:00  00:00:00       1.4       113.4       NaN
7  1975-1976  1976-01-02 00:00:00  17:45:00       1.5       116.9  In Range
8  1976-1977  1977-09-30 00:00:00  20:00:00       1.3        78.6  In Range
```

In [62]:

```
import os

def find_rochdale_files(base_path):
    cdr_files = []
    gdf_files = []
```

```
    # Search patterns
    cdr_pattern = '69803_cdr.csv'
    gdf_pattern = '69803_gdf.csv'

    # Walk through all directories and subdirectories
    for root, dirs, files in os.walk(base_path):
        for file in files:
            if file == cdr_pattern:
                cdr_files.append(os.path.join(root, file))
            elif file == gdf_pattern:
                gdf_files.append(os.path.join(root, file))

    return cdr_files, gdf_files

# Base path to search
base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA

# Find Rochdale CDR and GDF files
cdr_files, gdf_files = find_rochdale_files(base_path)

print("Rochdale CDR files:")
for file in cdr_files:
    print(file)

print("\nRochdale GDF files:")
for file in gdf_files:
    print(file)
```

```
Rochdale CDR files:
C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\ROCHDALE
\69803_cdr.csv

Rochdale GDF files:
C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\ROCHDALE
\69803_gdf.csv
```

In [63]:

```
import os
import pandas as pd

folder_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORI

# List files
print("Files in the folder:")
files = os.listdir(folder_path)
for file in files:
    print(file)

# If there's a CSV, read and display basic info
csv_files = [f for f in files if f.endswith('.csv')]
if csv_files:
    df = pd.read_csv(os.path.join(folder_path, csv_files[0]))
    print("\nDataFrame Info:")
    print(df.info())
```

```
Files in the folder:
peak_flow_data.csv

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   HISTORICAL DATA  32 non-null     object
 1   Unnamed: 1       32 non-null     object
 2   Unnamed: 2       32 non-null     object
 3   Unnamed: 3       32 non-null     object
 4   Unnamed: 4       32 non-null     object
 5   Unnamed: 5       32 non-null     object
 6   Unnamed: 6       32 non-null     object
 7   Unnamed: 7       32 non-null     object
 8   Unnamed: 8       32 non-null     object
 9   Unnamed: 9       1 non-null      object
dtypes: object(10)
memory usage: 2.7+ KB
None
```

In [64]:

```
import os

def check_station_folder(path):
    print(f"Contents of {os.path.basename(path)}:")
    files = os.listdir(path)
    for file in files:
        print(f"- {file}")
        file_path = os.path.join(path, file)
        if os.path.isfile(file_path):
            # For CSV and Excel files, print first line
            if file.endswith('.csv'):
                with open(file_path, 'r') as f:
                    print(f"  First line: {f.readline().strip()}")
            elif file.endswith('.xlsx'):
                print("  (Excel file - cannot preview content)")

# Paths for each station
stations = [
    r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\B
    r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\M
    r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DATA\R
]

for station_path in stations:
    check_station_folder(station_path)
    print("\n")
```

```
Contents of BURY STATION:
- 69044_cdr.csv
  First line: file,timestamp,2025-01-30T20:48:59
- 69044_gdf.csv
  First line: file,timestamp,2025-01-30T20:48:40
- BURY RIVER PEAK.xlsx
  (Excel file - cannot preview content)
- ~$BURY RIVER PEAK.xlsx
  (Excel file - cannot preview content)


Contents of MANCHESTER RACECOURSE:
- MACHESTER RACECOURSE RIVER PEAK.xlsx
  (Excel file - cannot preview content)
- ~$MACHESTER RACECOURSE RIVER PEAK.xlsx
  (Excel file - cannot preview content)


Contents of ROCHDALE:
- 69803_cdr.csv
  First line: file,timestamp,2025-01-30T20:38:35
- 69803_gdf.csv
  First line: file,timestamp,2025-01-30T20:33:30
- ROCHDALE RIVER PEAK.xlsx
  (Excel file - cannot preview content)
- ~$ROCHDALE RIVER PEAK.xlsx
  (Excel file - cannot preview content)
```

In [75]:

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

class HistoricalDataProcessor:
    def __init__(self, base_path):
        self.base_path = base_path
        self.stations = {
            'bury': {
                'folder': 'BURY STATION',
                'id': '69044',
                'has_cdr_gdf': True
            },
            'manchester': {
                'folder': 'MANCHESTER RACECOURSE',
                'id': '69023',
                'has_cdr_gdf': False
            },
            'rochdale': {
                'folder': 'ROCHDALE',
                'id': '69803',
                'has_cdr_gdf': True
            }
        }

    def process_peak_data(self, file_path):
        """Process peak flow data from Excel"""
        try:
            # Read Excel file, handling potential header issues
            df = pd.read_excel(file_path, header=None)
```

```
            # Find the row with column headers
            header_row = df[df.iloc[:,1] == 'Water Year'].index[0]

            # Extract data and set correct headers
            data = df.iloc[header_row+1:].copy()
            data.columns = df.iloc[header_row]

            # Select and clean relevant columns
            cols = ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'R
            processed_data = data[cols].dropna(subset=['Date'])

            # Convert Date to datetime
            processed_data['Date'] = pd.to_datetime(processed_data['Date'])

            # Convert Time to string if it's not already
            processed_data['Time'] = processed_data['Time'].astype(str)

            # Combine Date and Time
            processed_data['Datetime'] = pd.to_datetime(
                processed_data['Date'].dt.strftime('%Y-%m-%d') + ' ' +
                processed_data['Time'].str.strip(),
                errors='coerce'
            )

            return processed_data
        except Exception as e:
            print(f"Error processing peak data for {file_path}: {e}")
            return None

    def process_cdr_gdf(self, df):
        """
        Process CDR and GDF dataframes:
        - Keep only necessary columns
        - Replace 'M' values with missing (NaN)
        - Clean and format data
        """
        # For CDR data (Catchment Daily Rainfall)
        if 'dataType name Catchment Daily Rainfall' in df.columns:
            # Focus on date and rainfall columns
            rainfall_columns = [col for col in df.columns if 'Rainfall' in col]
            date_column = [col for col in df.columns if 'date' in col.lower()]

            # Select relevant columns
            if date_column and rainfall_columns:
                df = df[date_column + rainfall_columns]

        # For GDF data (Gauged Daily Flow)
        elif 'dataType name Gauged Daily Flow' in df.columns:
            # Focus on date and flow columns
            flow_columns = [col for col in df.columns if 'Flow' in col]
            date_column = [col for col in df.columns if 'date' in col.lower()]

            # Select relevant columns
            if date_column and flow_columns:
                df = df[date_column + flow_columns]

        # Replace 'M' with NaN in numeric columns
        numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
        for col in numeric_columns:
```

```
            df[col] = df[col].replace('M', np.nan)
            df[col] = pd.to_numeric(df[col], errors='coerce')

        return df

    def process_station_data(self, station_name):
        """Process data for a specific station"""
        station_info = self.stations[station_name]
        station_path = os.path.join(self.base_path, station_info['folder'])

        results = {
            'station': station_name.upper(),
            'cdr': None,
            'gdf': None,
            'peak_flow': None
        }

        # Process CDR and GDF if available
        if station_info['has_cdr_gdf']:
            try:
                # Read CDR and process
                cdr = pd.read_csv(os.path.join(station_path, f"{station_info['id
                results['cdr'] = self.process_cdr_gdf(cdr)

                # Read GDF and process
                gdf = pd.read_csv(os.path.join(station_path, f"{station_info['id
                results['gdf'] = self.process_cdr_gdf(gdf)
            except Exception as e:
                print(f"Error reading CDR/GDF for {station_name}: {e}")

        # Process peak flow data (existing code remains the same)
        peak_files = [f for f in os.listdir(station_path) if 'PEAK' in f.upper()
                     and f.endswith('.xlsx') and not f.startswith('~')]
        if peak_files:
            peak_file_path = os.path.join(station_path, peak_files[0])
            results['peak_flow'] = self.process_peak_data(peak_file_path)

        return results

    def process_all_stations(self):
        """Process data for all stations"""
        all_station_data = {}
        for station in self.stations:
            all_station_data[station] = self.process_station_data(station)
        return all_station_data

    def save_processed_data(self, processed_data, output_base_path):
        """Save processed data to CSV files"""
        # Ensure output directory exists
        os.makedirs(output_base_path, exist_ok=True)

        for station, data in processed_data.items():
            station_output_path = os.path.join(output_base_path, station)
            os.makedirs(station_output_path, exist_ok=True)

            # Save CDR
            if data['cdr'] is not None:
                try:
                    cdr_path = os.path.join(station_output_path, f"{station}_cdr
                    data['cdr'].to_csv(cdr_path, index=False)
```

```
                except PermissionError:
                    print(f"Permission denied: Cannot save CDR for {station}. Cl
                except Exception as e:
                    print(f"Error saving CDR for {station}: {e}")

            # Save GDF
            if data['gdf'] is not None:
                try:
                    gdf_path = os.path.join(station_output_path, f"{station}_gdf
                    data['gdf'].to_csv(gdf_path, index=False)
                except PermissionError:
                    print(f"Permission denied: Cannot save GDF for {station}. Cl
                except Exception as e:
                    print(f"Error saving GDF for {station}: {e}")

            # Save Peak Flow
            if data['peak_flow'] is not None:
                try:
                    peak_path = os.path.join(station_output_path, f"{station}_pe
                    data['peak_flow'].to_csv(peak_path, index=False)
                except PermissionError:
                    print(f"Permission denied: Cannot save peak flow for {static
                except Exception as e:
                    print(f"Error saving peak flow for {station}: {e}")

def main():
    # Base path for historical data
    base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTO

    # Output path for processed data
    output_base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTIO

    # Create processor
    processor = HistoricalDataProcessor(base_path)

    # Process all stations
    processed_data = processor.process_all_stations()

    # Save processed data
    processor.save_processed_data(processed_data, output_base_path)

    # Print summary
    for station, data in processed_data.items():
        print(f"\n{station.upper()} Station Summary:")
        for key, df in data.items():
            if df is not None:
                print(f"{key.upper()} Records: {len(df)}")

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

```
BURY Station Summary:
STATION Records: 4
CDR Records: 20840
GDF Records: 10190
PEAK_FLOW Records: 51

MANCHESTER Station Summary:
STATION Records: 10
PEAK_FLOW Records: 82

ROCHDALE Station Summary:
STATION Records: 8
CDR Records: 750
GDF Records: 11193
PEAK_FLOW Records: 31
```

DATA CLEANING AND STANDARDIZATION

In [76]:

```
import pandas as pd
import os

def standardize_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT"

    # 1. Real-time data
    print("\nCHECKING REAL-TIME DATA:")
    combined_path = os.path.join(project_path, 'combined_data')
    recent_file = sorted([f for f in os.listdir(combined_path) if f.endswith('.c
    realtime_df = pd.read_csv(os.path.join(combined_path, recent_file))
    print("\nReal-time data structure:")
    print(realtime_df.head())

    # 2. Historical NRFA Data for each station
    stations = {
        'ROCHDALE': {'id': '69803', 'peak_file': 'ROCHDALE RIVER PEAK.xlsx'},
        'MANCHESTER RACECOURSE': {'id': '69023', 'peak_file': 'MANCHESTER RIVER
        'BURY': {'id': '69044', 'peak_file': 'BURY RIVER PEAK.xlsx'}
    }

    for station_name, info in stations.items():
        print(f"\nCHECKING {station_name} HISTORICAL DATA:")
        station_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'HIS

        # Peak Flow Data
        peak_file = os.path.join(station_path, info['peak_file'])
        if os.path.exists(peak_file):
            peak_df = pd.read_excel(peak_file)
            print(f"\nPeak Flow data structure for {station_name}:")
            print(peak_df.head())

        # CDR & GDF Data (if available)
        cdr_file = os.path.join(station_path, f"{info['id']}_cdr.csv")
        gdf_file = os.path.join(station_path, f"{info['id']}_gdf.csv")

        if os.path.exists(cdr_file):
            cdr_df = pd.read_csv(cdr_file)
            print(f"\nCDR data available for {station_name}")

        if os.path.exists(gdf_file):
```

```
            gdf_df = pd.read_csv(gdf_file)
            print(f"\nGDF data available for {station_name}")

    # 3. Weather Data
    print("\nCHECKING WEATHER DATA:")
    weather_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'WEATHER
    temp_df = pd.read_excel(os.path.join(weather_path, 'TEMPERATURE.xlsx'))
    precip_df = pd.read_excel(os.path.join(weather_path, 'PRECIPITATION.xlsx'))
    print("\nTemperature data structure:")
    print(temp_df.head())
    print("\nPrecipitation data structure:")
    print(precip_df.head())

    # Summary of data types and formats
    print("\nDATA STANDARDIZATION NEEDED:")
    print("1. Real-time data: 15-minute intervals")
    print("2. Historical Peak Flow data: Event-based")
    print("3. CDR data: Daily rainfall")
    print("4. GDF data: Daily flow")
    print("5. Weather data: Monthly averages")

standardize_data()
```

```
CHECKING REAL-TIME DATA:

Real-time data structure:
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0          0.3  2025-01-31T19:15:00Z       0.0  2025-01-31T19:15:00Z
1          1.1  2025-01-31T19:15:00Z       0.0  2025-01-31T19:15:00Z
2          0.4  2025-01-31T19:15:00Z       0.0  2025-01-31T19:15:00Z

           location_name  river_station_id  rainfall_station_id
0               Rochdale            690203               561613
1  Manchester Racecourse            690510               562992
2            Bury Ground            690160               562656

CHECKING ROCHDALE HISTORICAL DATA:

Peak Flow data structure for ROCHDALE:
            HISTORICAL DATA  Unnamed: 1 Unnamed: 2 Unnamed: 3 Unnamed: 4  \
0                       NaN         NaN        NaN        NaN        NaN
1  69803 - Roch at Rochdale         NaN        NaN        NaN        NaN
2                       NaN         NaN        NaN        NaN        NaN
3                      Rank  Water Year       Date       Time  Stage (m)
4                       NaN         NaN        NaN        NaN        NaN

    Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9
0          NaN        NaN        NaN        NaN        NaN
1          NaN        NaN        NaN        NaN        NaN
2          NaN        NaN        NaN        NaN        NaN
3  Flow (m3/s)     Rating     Source        Ref   Comments
4          NaN        NaN        NaN        NaN        NaN

CDR data available for ROCHDALE

GDF data available for ROCHDALE

CHECKING MANCHESTER RACECOURSE HISTORICAL DATA:

CHECKING BURY HISTORICAL DATA:

CHECKING WEATHER DATA:

Temperature data structure:
                                          TEMPERATION Unnamed: 1
0  WEBSITE: https://climate-themetoffice.hub.arcg...        NaN
1                                                NaN        NaN
2                                                NaN        NaN
3                              MANCHESTER RACECOURSE        NaN
4                                            GRID_ID      AX-71

Precipitation data structure:
    PRECIPITATION DATA Unnamed: 1
0  MANCHESTER RECOURSE        NaN
1              GRID_ID      AX-71
2           pr January         90
3          pr February         76
4             pr March         66

DATA STANDARDIZATION NEEDED:
1. Real-time data: 15-minute intervals
2. Historical Peak Flow data: Event-based
3. CDR data: Daily rainfall
```

4. GDF data: Daily flow
5. Weather data: Monthly averages

In [79]:
```
import pandas as pd
import os

def clean_data_sources():
    project_path = r"C:\Users\Administrator\NEWPROJECT"

    def clean_peak_flows(df, station_name):
        """Clean peak flow data"""
        # Find header row and correct columns
        for idx, row in df.iterrows():
            if 'Water Year' in str(row.values):
                header_row = idx
                # Get data after header
                data = df.iloc[header_row+1:].copy()
                # Get column names from header row
                columns = df.iloc[header_row]
                data.columns = columns

                # Select and rename relevant columns
                data = data[['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m
                data.columns = ['Water_Year', 'Date', 'Time', 'Stage_m', 'Flow_m

                # Add station information
                data['Station'] = station_name

                # Convert date and numeric columns
                data['Date'] = pd.to_datetime(data['Date'])
                data['Stage_m'] = pd.to_numeric(data['Stage_m'], errors='coerce'
                data['Flow_m3s'] = pd.to_numeric(data['Flow_m3s'], errors='coerc

                # Remove rows with NaN dates
                data = data.dropna(subset=['Date', 'Flow_m3s'])

                return data[['Station', 'Water_Year', 'Date', 'Time', 'Stage_m',

        return None

    stations = {
        'Rochdale': {
            'folder': 'ROCHDALE',
            'peak_file': 'ROCHDALE RIVER PEAK.xlsx'
        },
        'Manchester Racecourse': {
            'folder': 'MANCHESTER RACECOURSE',
            'peak_file': 'MANCHESTER RIVER PEAK.xlsx'
        },
        'Bury Ground': {
            'folder': 'BURY STATION',
            'peak_file': 'BURY RIVER PEAK.xlsx'
        }
    }

    # Create output directory
    output_dir = os.path.join(project_path, 'cleaned_data')
    os.makedirs(output_dir, exist_ok=True)

    # Process Peak Flow Data
```

```
    all_peak_flows = []

    for station_name, info in stations.items():
        station_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'HIS
                                    info['folder'])
        peak_file = os.path.join(station_path, info['peak_file'])

        if os.path.exists(peak_file):
            print(f"\nProcessing Peak Flow data for {station_name}")
            try:
                peak_df = pd.read_excel(peak_file)
                cleaned_peaks = clean_peak_flows(peak_df, station_name)
                if cleaned_peaks is not None:
                    all_peak_flows.append(cleaned_peaks)
                    print(f"Records found: {len(cleaned_peaks)}")
                    print("\nSample data:")
                    print(cleaned_peaks.head())
            except Exception as e:
                print(f"Error processing {station_name}: {str(e)}")

    # Combine all peak flows
    if all_peak_flows:
        combined_peaks = pd.concat(all_peak_flows, ignore_index=True)

        # Sort by date
        combined_peaks = combined_peaks.sort_values(['Station', 'Date'])

        # Save to CSV
        output_file = os.path.join(output_dir, 'cleaned_peak_flows.csv')
        combined_peaks.to_csv(output_file, index=False)

        print("\nFinal Combined Peak Flow Data Summary:")
        print("=====================================")
        for station in combined_peaks['Station'].unique():
            station_data = combined_peaks[combined_peaks['Station'] == station]
            print(f"\n{station}:")
            print(f"Total records: {len(station_data)}")
            print(f"Date range: {station_data['Date'].min()} to {station_data['D
            print(f"Maximum flow: {station_data['Flow_m3s'].max():.1f} m3/s")
            print(f"Maximum stage: {station_data['Stage_m'].max():.2f} m")

clean_data_sources()
```

```
Processing Peak Flow data for Rochdale
Records found: 31

Sample data:
    Station Water_Year       Date      Time  Stage_m  Flow_m3s    Rating  \
5  Rochdale  1992-1993 1993-09-13  11:30:00      0.9      21.1  In Range
6  Rochdale  1993-1994 1993-12-08  23:45:00      1.3      38.3  In Range
7  Rochdale  1994-1995 1995-01-31  23:15:00      1.6      56.7  In Range
8  Rochdale  1995-1996 1996-02-18  03:15:00      0.8      18.0  In Range
9  Rochdale  1996-1997 1996-11-06  02:15:00      1.2      36.3  In Range

            Source
5  Digital Archive
6  Digital Archive
7  Digital Archive
8  Digital Archive
9  Digital Archive

Processing Peak Flow data for Bury Ground
Records found: 51

Sample data:
       Station Water_Year       Date      Time  Stage_m  Flow_m3s    Rating  \
3  Bury Ground  1972-1973 1973-01-12  00:00:00      1.3      78.1       NaN
4  Bury Ground  1973-1974 1974-02-11  00:00:00      1.5     118.0       NaN
5  Bury Ground  1974-1975 1975-01-21  00:00:00      1.4     113.4       NaN
6  Bury Ground  1975-1976 1976-01-02  17:45:00      1.5     116.9  In Range
7  Bury Ground  1976-1977 1977-09-30  20:00:00      1.3      78.6  In Range

                                         Source
3                               Digital Archive
4                               Digital Archive
5                               Digital Archive
6  Estimated stage data from Bury Bridge (69035)
7  Estimated stage data from Bury Bridge (69035)

Final Combined Peak Flow Data Summary:
=====================================

Bury Ground:
Total records: 51
Date range: 1973-01-12 00:00:00 to 2023-07-23 00:00:00
Maximum flow: 283.6 m3/s
Maximum stage: 2.18 m

Rochdale:
Total records: 31
Date range: 1993-09-13 00:00:00 to 2023-07-23 00:00:00
Maximum flow: 92.8 m3/s
Maximum stage: 2.22 m
```

In [88]:

```
import pandas as pd
import os

def clean_data_sources():
    project_path = r"C:\Users\Administrator\NEWPROJECT"

    stations = {
        'Rochdale': {
            'folder': 'ROCHDALE',
```

```
            'peak_file': 'ROCHDALE RIVER PEAK.xlsx'
        },
        'Manchester Racecourse': {
            'folder': 'MANCHESTER RACECOURSE',  # Updated path
            'peak_file': 'MANCHESTER RACECOURSE RIVER PEAK.xlsx'  # Updated file
        },
        'Bury Ground': {
            'folder': 'BURY STATION',
            'peak_file': 'BURY RIVER PEAK.xlsx'
        }
    }

    def clean_peak_flows(df, station_name):
        """Clean peak flow data"""
        # Find header row and correct columns
        header_row = None
        for idx, row in df.iterrows():
            if any('Water Year' in str(val) for val in row.values):
                header_row = idx
                break

        if header_row is None:
            print(f"Could not find header row for {station_name}")
            return None

        # Get data after header
        data = df.iloc[header_row+1:].copy()
        # Get column names from header row
        columns = df.iloc[header_row]
        data.columns = columns

        print(f"\nColumns found for {station_name}:")
        print(columns.tolist())

        # Select and rename relevant columns
        data = data[['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', '
        data.columns = ['Water_Year', 'Date', 'Time', 'Stage_m', 'Flow_m3s', 'Ra

        # Add station information
        data['Station'] = station_name

        # Convert date and numeric columns
        data['Date'] = pd.to_datetime(data['Date'])
        data['Stage_m'] = pd.to_numeric(data['Stage_m'], errors='coerce')
        data['Flow_m3s'] = pd.to_numeric(data['Flow_m3s'], errors='coerce')

        # Remove rows with NaN dates or flows
        data = data.dropna(subset=['Date', 'Flow_m3s'])

        return data[['Station', 'Water_Year', 'Date', 'Time', 'Stage_m', 'Flow_m

    # Create output directory
    output_dir = os.path.join(project_path, 'cleaned_data')
    os.makedirs(output_dir, exist_ok=True)

    # Process Peak Flow Data
    all_peak_flows = []

    for station_name, info in stations.items():
        station_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'HIS
```

```
                                         info['folder'])
        peak_file = os.path.join(station_path, info['peak_file'])

        if os.path.exists(peak_file):
            print(f"\nProcessing Peak Flow data for {station_name}")
            try:
                peak_df = pd.read_excel(peak_file)
                cleaned_peaks = clean_peak_flows(peak_df, station_name)
                if cleaned_peaks is not None:
                    all_peak_flows.append(cleaned_peaks)
                    print(f"\nRecords found: {len(cleaned_peaks)}")
                    print("\nSample data:")
                    print(cleaned_peaks.head())
            except Exception as e:
                print(f"Error processing {station_name}: {str(e)}")
        else:
            print(f"\nFile not found: {peak_file}")

    # Combine all peak flows
    if all_peak_flows:
        combined_peaks = pd.concat(all_peak_flows, ignore_index=True)

        # Sort by date
        combined_peaks = combined_peaks.sort_values(['Station', 'Date'])

        # Save to CSV
        output_file = os.path.join(output_dir, 'cleaned_peak_flows.csv')
        combined_peaks.to_csv(output_file, index=False)

        print("\nFinal Combined Peak Flow Data Summary:")
        print("=====================================")
        for station in combined_peaks['Station'].unique():
            station_data = combined_peaks[combined_peaks['Station'] == station]
            print(f"\n{station}:")
            print(f"Total records: {len(station_data)}")
            print(f"Date range: {station_data['Date'].min()} to {station_data['D
            print(f"Maximum flow: {station_data['Flow_m3s'].max():.1f} m3/s")
            print(f"Maximum stage: {station_data['Stage_m'].max():.2f} m")

clean_data_sources()
```

```
Processing Peak Flow data for Rochdale

Columns found for Rochdale:
['Rank', 'Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Sou
rce', 'Ref', 'Comments']

Records found: 31

Sample data:
    Station Water_Year       Date      Time  Stage_m  Flow_m3s    Rating  \
5  Rochdale  1992-1993 1993-09-13  11:30:00      0.9      21.1  In Range
6  Rochdale  1993-1994 1993-12-08  23:45:00      1.3      38.3  In Range
7  Rochdale  1994-1995 1995-01-31  23:15:00      1.6      56.7  In Range
8  Rochdale  1995-1996 1996-02-18  03:15:00      0.8      18.0  In Range
9  Rochdale  1996-1997 1996-11-06  02:15:00      1.2      36.3  In Range

            Source
5  Digital Archive
6  Digital Archive
7  Digital Archive
8  Digital Archive
9  Digital Archive

File not found: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORIC
AL DATA\MANCHESTER RACECOURSE\MANCHESTER RACECOURSE RIVER PEAK.xlsx

Processing Peak Flow data for Bury Ground

Columns found for Bury Ground:
['Rank', 'Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Sou
rce', 'Ref', 'Comments']

Records found: 51

Sample data:
       Station Water_Year       Date      Time  Stage_m  Flow_m3s    Rating  \
3  Bury Ground  1972-1973 1973-01-12  00:00:00      1.3      78.1       NaN
4  Bury Ground  1973-1974 1974-02-11  00:00:00      1.5     118.0       NaN
5  Bury Ground  1974-1975 1975-01-21  00:00:00      1.4     113.4       NaN
6  Bury Ground  1975-1976 1976-01-02  17:45:00      1.5     116.9  In Range
7  Bury Ground  1976-1977 1977-09-30  20:00:00      1.3      78.6  In Range

                                      Source
3                            Digital Archive
4                            Digital Archive
5                            Digital Archive
6  Estimated stage data from Bury Bridge (69035)
7  Estimated stage data from Bury Bridge (69035)

Final Combined Peak Flow Data Summary:
=====================================

Bury Ground:
Total records: 51
Date range: 1973-01-12 00:00:00 to 2023-07-23 00:00:00
Maximum flow: 283.6 m3/s
Maximum stage: 2.18 m

Rochdale:
Total records: 31
```

```
Date range: 1993-09-13 00:00:00 to 2023-07-23 00:00:00
Maximum flow: 92.8 m3/s
Maximum stage: 2.22 m
```

In [95]:

```python
import pandas as pd
import os
import traceback

def clean_data_sources():
    project_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HI

    stations = {
        'Manchester Racecourse': {
            'folder': 'manchester',
            'peak_file': 'manchester_peak_flow.csv'
        },
        'Rochdale': {
            'folder': 'rochdale',
            'peak_file': 'rochdale_peak_flow.csv'
        },
        'Bury Ground': {
            'folder': 'bury',
            'peak_file': 'bury_peak_flow.csv'
        }
    }

    # Create output directory
    output_dir = os.path.join(r"C:\Users\Administrator\NEWPROJECT", 'cleaned_dat
    os.makedirs(output_dir, exist_ok=True)

    # Process Peak Flow Data
    all_peak_flows = []

    for station_name, info in stations.items():
        station_path = os.path.join(project_path, info['folder'])
        peak_file = os.path.join(station_path, info['peak_file'])

        print(f"\n--- Checking {station_name} ---")
        print("Full file path:", peak_file)
        print("File exists:", os.path.exists(peak_file))

        if os.path.exists(peak_file):
            try:
                # Use read_csv instead of read_excel
                peak_df = pd.read_csv(peak_file)
                print(f"File read successfully. Shape: {peak_df.shape}")
                print("\nFirst few rows:")
                print(peak_df.head())

                # Add station column
                peak_df['Station'] = station_name

                all_peak_flows.append(peak_df)
                print(f"\nRecords found: {len(peak_df)}")
            except Exception as e:
                print("Error processing file:")
                print(traceback.format_exc())
        else:
            print(f"ERROR: File does not exist at {peak_file}")
```

```
    # Combine and save data
    if all_peak_flows:
        combined_peaks = pd.concat(all_peak_flows, ignore_index=True)
        output_file = os.path.join(output_dir, 'combined_peak_flows.csv')
        combined_peaks.to_csv(output_file, index=False)
        print("\nData saved to:", output_file)

        # Print summary
        print("\nFinal Combined Peak Flow Data Summary:")
        print("=====================================")
        for station in combined_peaks['Station'].unique():
            station_data = combined_peaks[combined_peaks['Station'] == station]
            print(f"\n{station}:")
            print(f"Total records: {len(station_data)}")

clean_data_sources()
```

```
--- Checking Manchester Racecourse ---
Full file path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORIC
AL DATA\processed\manchester\manchester_peak_flow.csv
File exists: True
File read successfully. Shape: (82, 7)

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)   Rating  \
0  1941-1942  1941-10-24  00:00:00        3.5        269.0  Extrap.
1  1942-1943  1942-10-17  00:00:00        3.2        223.0  Extrap.
2  1943-1944  1944-01-23  00:00:00        4.1        374.0  Extrap.
3  1944-1945  1945-02-02  00:00:00        3.9        339.0  Extrap.
4  1945-1946  1946-09-20  00:00:00        5.3        500.0  Extrap.

              Datetime
0  1941-10-24 00:00:00
1  1942-10-17 00:00:00
2  1944-01-23 00:00:00
3  1945-02-02 00:00:00
4  1946-09-20 00:00:00

Records found: 82

--- Checking Rochdale ---
Full file path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORIC
AL DATA\processed\rochdale\rochdale_peak_flow.csv
File exists: True
File read successfully. Shape: (31, 7)

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1992-1993  1993-09-13  11:30:00        0.9         21.1  In Range
1  1993-1994  1993-12-08  23:45:00        1.3         38.3  In Range
2  1994-1995  1995-01-31  23:15:00        1.6         56.7  In Range
3  1995-1996  1996-02-18  03:15:00        0.8         18.0  In Range
4  1996-1997  1996-11-06  02:15:00        1.2         36.3  In Range

              Datetime
0  1993-09-13 11:30:00
1  1993-12-08 23:45:00
2  1995-01-31 23:15:00
3  1996-02-18 03:15:00
4  1996-11-06 02:15:00

Records found: 31

--- Checking Bury Ground ---
Full file path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORIC
AL DATA\processed\bury\bury_peak_flow.csv
File exists: True
File read successfully. Shape: (51, 7)

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1972-1973  1973-01-12  00:00:00        1.3         78.1       NaN
1  1973-1974  1974-02-11  00:00:00        1.5        118.0       NaN
2  1974-1975  1975-01-21  00:00:00        1.4        113.4       NaN
3  1975-1976  1976-01-02  17:45:00        1.5        116.9  In Range
4  1976-1977  1977-09-30  20:00:00        1.3         78.6  In Range
```

```
                Datetime
0  1973-01-12 00:00:00
1  1974-02-11 00:00:00
2  1975-01-21 00:00:00
3  1976-01-02 17:45:00
4  1977-09-30 20:00:00

Records found: 51

Data saved to: C:\Users\Administrator\NEWPROJECT\cleaned_data\combined_peak_flow
s.csv

Final Combined Peak Flow Data Summary:
=====================================

Manchester Racecourse:
Total records: 82

Rochdale:
Total records: 31

Bury Ground:
Total records: 51
```

In [96]:

```
import os

base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA

def list_files_recursively(base_path):
    for root, dirs, files in os.walk(base_path):
        level = root.replace(base_path, '').count(os.sep)
        indent = ' ' * 4 * level
        print(f"{indent}{os.path.basename(root)}/")
        subindent = ' ' * 4 * (level + 1)
        for file in files:
            print(f"{subindent}{file}")

list_files_recursively(base_path)
```

```
processed/
    bury/
        bury_cdr.csv
        bury_gdf.csv
        bury_peak_flow.csv
    cleaned_data/
    manchester/
        manchester_peak_flow.csv
    rochdale/
        rochdale_cdr.csv
        rochdale_gdf.csv
        rochdale_peak_flow.csv
```

In [97]:

```
import pandas as pd
import os

def inspect_csv_files():
    base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTO

    files_to_check = [
        os.path.join(base_path, 'bury', 'bury_cdr.csv'),
```

```
        os.path.join(base_path, 'bury', 'bury_gdf.csv'),
        os.path.join(base_path, 'rochdale', 'rochdale_cdr.csv'),
        os.path.join(base_path, 'rochdale', 'rochdale_gdf.csv'),
        os.path.join(base_path, 'manchester', 'manchester_peak_flow.csv'),
        os.path.join(base_path, 'bury', 'bury_peak_flow.csv'),
        os.path.join(base_path, 'rochdale', 'rochdale_peak_flow.csv')
    ]

    for file_path in files_to_check:
        print(f"\n--- Inspecting {os.path.basename(file_path)} ---")
        try:
            # Read the CSV file
            df = pd.read_csv(file_path)

            # Basic information
            print(f"Full Path: {file_path}")
            print(f"Total Records: {len(df)}")

            # Columns
            print("\nColumns:")
            print(df.columns.tolist())

            # Data types
            print("\nData Types:")
            print(df.dtypes)

            # First few rows
            print("\nFirst 5 Rows:")
            print(df.head())

            # Basic statistics for numeric columns
            numeric_cols = df.select_dtypes(include=['float64', 'int64']).column
            if len(numeric_cols) > 0:
                print("\nNumeric Columns Statistics:")
                print(df[numeric_cols].describe())

            # Date range if applicable
            date_cols = df.select_dtypes(include=['datetime64', 'object']).colum
            for col in date_cols:
                if 'date' in col.lower() or 'time' in col.lower():
                    try:
                        date_ser = pd.to_datetime(df[col], errors='coerce')
                        print(f"\nDate Range for {col}:")
                        print(f"Earliest: {date_ser.min()}")
                        print(f"Latest: {date_ser.max()}")
                    except Exception as date_err:
                        print(f"Could not process dates in {col}: {date_err}")

        except Exception as e:
            print(f"Error reading {os.path.basename(file_path)}: {str(e)}")

# Run the inspection
inspect_csv_files()
```

```
--- Inspecting bury_cdr.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\bury\bury_cdr.csv
Total Records: 20840

Columns:
['file', 'timestamp', '2025-01-30T20:48:59']

Data Types:
file                   object
timestamp              object
2025-01-30T20:48:59    object
dtype: object

First 5 Rows:
       file      timestamp             2025-01-30T20:48:59
0  database             id                 nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                          69044
3   station           name           Irwell at Bury Ground
4   station  gridReference                   SD7998711393

Date Range for timestamp:
Earliest: NaT
Latest: NaT

--- Inspecting bury_gdf.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\bury\bury_gdf.csv
Total Records: 10190

Columns:
['file', 'timestamp', '2025-01-30T20:48:40']

Data Types:
file                   object
timestamp              object
2025-01-30T20:48:40    object
dtype: object

First 5 Rows:
       file      timestamp             2025-01-30T20:48:40
0  database             id                 nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                          69044
3   station           name           Irwell at Bury Ground
4   station  gridReference                   SD7998711393
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
```

```
Date Range for timestamp:
Earliest: NaT
Latest: NaT

--- Inspecting rochdale_cdr.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\rochdale\rochdale_cdr.csv
Total Records: 750

Columns:
['file', 'timestamp', '2025-01-30T20:38:35']

Data Types:
file                   object
timestamp              object
2025-01-30T20:38:35    object
dtype: object

First 5 Rows:
       file      timestamp             2025-01-30T20:38:35
0  database             id                  nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                           69803
3   station           name                Roch at Rochdale
4   station  gridReference                        SD882127

Date Range for timestamp:
Earliest: NaT
Latest: NaT

--- Inspecting rochdale_gdf.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\rochdale\rochdale_gdf.csv
Total Records: 11193

Columns:
['file', 'timestamp', '2025-01-30T20:33:30']

Data Types:
file                   object
timestamp              object
2025-01-30T20:33:30    object
dtype: object

First 5 Rows:
       file      timestamp             2025-01-30T20:33:30
0  database             id                  nrfa-public-31
1  database           name  UK National River Flow Archive
2   station             id                           69803
3   station           name                Roch at Rochdale
4   station  gridReference                        SD882127
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
```

```
Date Range for timestamp:
Earliest: NaT
Latest: NaT

--- Inspecting manchester_peak_flow.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\manchester\manchester_peak_flow.csv
Total Records: 82

Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Data Types:
Water Year      object
Date            object
Time            object
Stage (m)      float64
Flow (m3/s)    float64
Rating          object
Datetime        object
dtype: object

First 5 Rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)   Rating  \
0  1941-1942  1941-10-24  00:00:00        3.5        269.0  Extrap.
1  1942-1943  1942-10-17  00:00:00        3.2        223.0  Extrap.
2  1943-1944  1944-01-23  00:00:00        4.1        374.0  Extrap.
3  1944-1945  1945-02-02  00:00:00        3.9        339.0  Extrap.
4  1945-1946  1946-09-20  00:00:00        5.3        500.0  Extrap.

              Datetime
0  1941-10-24 00:00:00
1  1942-10-17 00:00:00
2  1944-01-23 00:00:00
3  1945-02-02 00:00:00
4  1946-09-20 00:00:00

Numeric Columns Statistics:
       Stage (m)  Flow (m3/s)
count       82.0         82.0
mean         3.5        279.4
std          0.6         87.4
min          2.5        135.0
25%          3.1        217.3
50%          3.5        273.5
75%          3.8        327.3
max          5.7        560.0

Date Range for Date:
Earliest: 1941-10-24 00:00:00
Latest: 2023-01-10 00:00:00

Date Range for Time:
Earliest: 2025-01-31 00:00:00
Latest: 2025-01-31 23:00:00

Date Range for Datetime:
Earliest: 1941-10-24 00:00:00
Latest: 2023-01-10 16:15:00
```

```
--- Inspecting bury_peak_flow.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\bury\bury_peak_flow.csv
Total Records: 51

Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Data Types:
Water Year      object
Date            object
Time            object
Stage (m)      float64
Flow (m3/s)    float64
Rating          object
Datetime        object
dtype: object

First 5 Rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1972-1973  1973-01-12  00:00:00        1.3         78.1       NaN
1  1973-1974  1974-02-11  00:00:00        1.5        118.0       NaN
2  1974-1975  1975-01-21  00:00:00        1.4        113.4       NaN
3  1975-1976  1976-01-02  17:45:00        1.5        116.9  In Range
4  1976-1977  1977-09-30  20:00:00        1.3         78.6  In Range

              Datetime
0  1973-01-12 00:00:00
1  1974-02-11 00:00:00
2  1975-01-21 00:00:00
3  1976-01-02 17:45:00
4  1977-09-30 20:00:00

Numeric Columns Statistics:
       Stage (m)  Flow (m3/s)
count       51.0         51.0
mean         1.4        115.9
std          0.2         43.6
min          1.1         51.5
25%          1.3         84.6
50%          1.4        112.9
75%          1.5        125.6
max          2.2        283.6

Date Range for Date:
Earliest: 1973-01-12 00:00:00
Latest: 2023-07-23 00:00:00

Date Range for Time:
Earliest: 2025-01-31 00:00:00
Latest: 2025-01-31 23:30:00

Date Range for Datetime:
Earliest: 1973-01-12 00:00:00
Latest: 2023-07-23 12:15:00

--- Inspecting rochdale_peak_flow.csv ---
Full Path: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DA
TA\processed\rochdale\rochdale_peak_flow.csv
Total Records: 31
```

```
Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Data Types:
Water Year      object
Date            object
Time            object
Stage (m)      float64
Flow (m3/s)    float64
Rating          object
Datetime        object
dtype: object

First 5 Rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1992-1993  1993-09-13  11:30:00        0.9         21.1  In Range
1  1993-1994  1993-12-08  23:45:00        1.3         38.3  In Range
2  1994-1995  1995-01-31  23:15:00        1.6         56.7  In Range
3  1995-1996  1996-02-18  03:15:00        0.8         18.0  In Range
4  1996-1997  1996-11-06  02:15:00        1.2         36.3  In Range

              Datetime
0  1993-09-13 11:30:00
1  1993-12-08 23:45:00
2  1995-01-31 23:15:00
3  1996-02-18 03:15:00
4  1996-11-06 02:15:00

Numeric Columns Statistics:
       Stage (m)  Flow (m3/s)
count       31.0         31.0
mean         1.4         46.4
std          0.3         15.0
min          0.8         18.0
25%          1.3         38.1
50%          1.4         44.7
75%          1.5         51.3
max          2.2         92.8

Date Range for Date:
Earliest: 1993-09-13 00:00:00
Latest: 2023-07-23 00:00:00

Date Range for Time:
Earliest: 2025-01-31 00:00:00
Latest: 2025-01-31 23:45:00

Date Range for Datetime:
Earliest: 1993-09-13 11:30:00
Latest: 2023-07-23 12:15:00
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\2544550040.py:50: UserW
arning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please s
pecify a format.
  date_ser = pd.to_datetime(df[col], errors='coerce')
```

In [100…

```
import pandas as pd
import os
import matplotlib.pyplot as plt

def process_daily_data(base_path):
    stations = {
        'Bury': {
            'gdf_file': 'bury_gdf.csv',
            'cdr_file': 'bury_cdr.csv',
            'station_id': '69044'
        },
        'Rochdale': {
            'gdf_file': 'rochdale_gdf.csv',
            'cdr_file': 'rochdale_cdr.csv',
            'station_id': '69803'
        }
    }

    for station_name, files in stations.items():
        print(f"\n--- {station_name} Station Analysis ---")

        # Process Gauged Daily Flow (GDF)
        gdf_path = os.path.join(base_path, station_name.lower(), files['gdf_file
        if os.path.exists(gdf_path):
            try:
                # Read the entire CSV
                gdf_df = pd.read_csv(gdf_path)

                # Print full dataframe to understand structure
                print("\nGDF Data Structure:")
                print(gdf_df)

                # Print column names
                print("\nColumns:")
                print(gdf_df.columns)

            except Exception as e:
                print(f"Error processing GDF for {station_name}: {e}")

        # Process Catchment Daily Rainfall (CDR)
        cdr_path = os.path.join(base_path, station_name.lower(), files['cdr_file
        if os.path.exists(cdr_path):
            try:
                # Read the entire CSV
```

```
                cdr_df = pd.read_csv(cdr_path)

                # Print full dataframe to understand structure
                print("\nCDR Data Structure:")
                print(cdr_df)

                # Print column names
                print("\nColumns:")
                print(cdr_df.columns)

            except Exception as e:
                print(f"Error processing CDR for {station_name}: {e}")

# Base path for processed historical data
base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA

# Run the analysis
process_daily_data(base_path)
```

```
--- Bury Station Analysis ---

GDF Data Structure:
            file      timestamp             2025-01-30T20:48:40
0       database             id                  nrfa-public-31
1       database           name  UK National River Flow Archive
2        station             id                           69044
3        station           name            Irwell at Bury Ground
4        station  gridReference                     SD7998711393
...          ...            ...                             ...
10185  26/09/2023          2.439                             NaN
10186  27/09/2023          2.769                             NaN
10187  28/09/2023          2.562                             NaN
10188  29/09/2023          2.277                             NaN
10189  30/09/2023           6.73                             NaN

[10190 rows x 3 columns]

Columns:
Index(['file', 'timestamp', '2025-01-30T20:48:40'], dtype='object')

CDR Data Structure:
            file      timestamp             2025-01-30T20:48:59
0       database             id                  nrfa-public-31
1       database           name  UK National River Flow Archive
2        station             id                           69044
3        station           name            Irwell at Bury Ground
4        station  gridReference                     SD7998711393
...          ...            ...                             ...
20835  27/12/2017              0                            3000
20836  28/12/2017            2.4                            3000
20837  29/12/2017           18.1                            3000
20838  30/12/2017            5.8                            3000
20839  31/12/2017            7.1                            3000

[20840 rows x 3 columns]

Columns:
Index(['file', 'timestamp', '2025-01-30T20:48:59'], dtype='object')

--- Rochdale Station Analysis ---

GDF Data Structure:
            file      timestamp             2025-01-30T20:33:30
0       database             id                  nrfa-public-31
1       database           name  UK National River Flow Archive
2        station             id                           69803
3        station           name                Roch at Rochdale
4        station  gridReference                        SD882127
...          ...            ...                             ...
11188  2023-09-26          1.198                             NaN
11189  2023-09-27          1.181                             NaN
11190  2023-09-28          1.037                             NaN
11191  2023-09-29          0.916                             NaN
11192  2023-09-30          3.654                             NaN

[11193 rows x 3 columns]

Columns:
Index(['file', 'timestamp', '2025-01-30T20:33:30'], dtype='object')
```

```
CDR Data Structure:
           file      timestamp           2025-01-30T20:38:35
0      database             id                nrfa-public-31
1      database           name  UK National River Flow Archive
2       station             id                         69803
3       station           name            Roch at Rochdale
4       station  gridReference                       SD882127
..          ...            ...                           ...
745  2017-12-27          0.000                          2000
746  2017-12-28          3.400                          2000
747  2017-12-29         17.200                          2000
748  2017-12-30          4.800                          2000
749  2017-12-31          5.700                          2000

[750 rows x 3 columns]

Columns:
Index(['file', 'timestamp', '2025-01-30T20:38:35'], dtype='object')
```

PREPROCESSING RAINFALL AND PEAK FLOW HISTORICAL DATA

In [105…

```
import pandas as pd
import os

def process_nrfa_data(base_path):
    stations = {
        'Bury': {
            'gdf_file': 'bury_gdf.csv',
            'cdr_file': 'bury_cdr.csv',
            'station_id': '69044'
        },
        'Rochdale': {
            'gdf_file': 'rochdale_gdf.csv',
            'cdr_file': 'rochdale_cdr.csv',
            'station_id': '69803'
        }
    }

    for station_name, files in stations.items():
        print(f"\n--- {station_name} Station Analysis ---")

        # Process Gauged Daily Flow (GDF)
        gdf_path = os.path.join(base_path, station_name.lower(), files['gdf_file
        if os.path.exists(gdf_path):
            try:
                # Read the entire CSV
                gdf_df = pd.read_csv(gdf_path)

                # Try different date filtering strategies
                gdf_data_dd_mm_yyyy = gdf_df[gdf_df['file'].str.contains(r'^\d{2
                gdf_data_yyyy_mm_dd = gdf_df[gdf_df['file'].str.contains(r'^\d{4

                # Process dd/mm/yyyy format
                if not gdf_data_dd_mm_yyyy.empty:
                    gdf_data_dd_mm_yyyy.columns = ['Date', 'Flow', 'Extra']
                    gdf_data_dd_mm_yyyy['Date'] = pd.to_datetime(gdf_data_dd_mm_
                    gdf_data_dd_mm_yyyy['Flow'] = pd.to_numeric(gdf_data_dd_mm_y
```

```
                gdf_data_dd_mm_yyyy = gdf_data_dd_mm_yyyy.dropna(subset=['Da

            # Process yyyy-mm-dd format
            if not gdf_data_yyyy_mm_dd.empty:
                gdf_data_yyyy_mm_dd.columns = ['Date', 'Flow', 'Extra']
                gdf_data_yyyy_mm_dd['Date'] = pd.to_datetime(gdf_data_yyyy_m
                gdf_data_yyyy_mm_dd['Flow'] = pd.to_numeric(gdf_data_yyyy_mm
                gdf_data_yyyy_mm_dd = gdf_data_yyyy_mm_dd.dropna(subset=['Da

            # Combine or select the non-empty dataframe
            gdf_data = gdf_data_dd_mm_yyyy if not gdf_data_dd_mm_yyyy.empty

            # Basic analysis
            print("\nGauged Daily Flow Data:")
            print(f"Total records: {len(gdf_data)}")
            print(f"Date range: {gdf_data['Date'].min()} to {gdf_data['Date'
            print("\nFlow Statistics:")
            print(gdf_data['Flow'].describe())

            # Save processed data
            output_path = os.path.join(base_path, 'cleaned_data', f'{station
            os.makedirs(os.path.dirname(output_path), exist_ok=True)
            gdf_data.to_csv(output_path, index=False)
            print(f"\nProcessed data saved to: {output_path}")

        except Exception as e:
            print(f"Error processing GDF for {station_name}: {e}")

    # Process Catchment Daily Rainfall (CDR)
    cdr_path = os.path.join(base_path, station_name.lower(), files['cdr_file
    if os.path.exists(cdr_path):
        try:
            # Read the entire CSV
            cdr_df = pd.read_csv(cdr_path)

            # Try different date filtering strategies
            cdr_data_dd_mm_yyyy = cdr_df[cdr_df['file'].str.contains(r'^\d{2
            cdr_data_yyyy_mm_dd = cdr_df[cdr_df['file'].str.contains(r'^\d{4

            # Process dd/mm/yyyy format
            if not cdr_data_dd_mm_yyyy.empty:
                cdr_data_dd_mm_yyyy.columns = ['Date', 'Rainfall', 'Extra']
                cdr_data_dd_mm_yyyy['Date'] = pd.to_datetime(cdr_data_dd_mm_
                cdr_data_dd_mm_yyyy['Rainfall'] = pd.to_numeric(cdr_data_dd_
                cdr_data_dd_mm_yyyy = cdr_data_dd_mm_yyyy.dropna(subset=['Da

            # Process yyyy-mm-dd format
            if not cdr_data_yyyy_mm_dd.empty:
                cdr_data_yyyy_mm_dd.columns = ['Date', 'Rainfall', 'Extra']
                cdr_data_yyyy_mm_dd['Date'] = pd.to_datetime(cdr_data_yyyy_m
                cdr_data_yyyy_mm_dd['Rainfall'] = pd.to_numeric(cdr_data_yyy
                cdr_data_yyyy_mm_dd = cdr_data_yyyy_mm_dd.dropna(subset=['Da

            # Combine or select the non-empty dataframe
            cdr_data = cdr_data_dd_mm_yyyy if not cdr_data_dd_mm_yyyy.empty

            # Basic analysis
            print("\nCatchment Daily Rainfall Data:")
            print(f"Total records: {len(cdr_data)}")
            print(f"Date range: {cdr_data['Date'].min()} to {cdr_data['Date'
```

```
                print("\nRainfall Statistics:")
                print(cdr_data['Rainfall'].describe())

                # Save processed data
                output_path = os.path.join(base_path, 'cleaned_data', f'{station
                os.makedirs(os.path.dirname(output_path), exist_ok=True)
                cdr_data.to_csv(output_path, index=False)
                print(f"\nProcessed data saved to: {output_path}")

            except Exception as e:
                print(f"Error processing CDR for {station_name}: {e}")

# Base path for processed historical data
base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA

# Run the analysis
process_nrfa_data(base_path)
```

```
--- Bury Station Analysis ---

Gauged Daily Flow Data:
Total records: 9928
Date range: 1995-11-22 00:00:00 to 2023-09-30 00:00:00

Flow Statistics:
count   9928.0
mean       3.9
std        5.4
min        0.4
25%        1.2
50%        2.1
75%        4.1
max      117.0
Name: Flow, dtype: float64

Processed data saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION
\HISTORICAL DATA\processed\cleaned_data\bury_daily_flow.csv

Catchment Daily Rainfall Data:
Total records: 20819
Date range: 1961-01-01 00:00:00 to 2017-12-31 00:00:00

Rainfall Statistics:
count   20819.0
mean        3.8
std         6.2
min         0.0
25%         0.0
50%         0.9
75%         5.1
max        79.5
Name: Rainfall, dtype: float64

Processed data saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION
\HISTORICAL DATA\processed\cleaned_data\bury_daily_rainfall.csv

--- Rochdale Station Analysis ---

Gauged Daily Flow Data:
Total records: 11118
Date range: 1993-02-26 00:00:00 to 2023-09-30 00:00:00

Flow Statistics:
count   11118.0
mean        2.8
std         3.5
min         0.2
25%         0.8
50%         1.5
75%         3.3
max        50.4
Name: Flow, dtype: float64

Processed data saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION
\HISTORICAL DATA\processed\cleaned_data\rochdale_daily_flow.csv

Catchment Daily Rainfall Data:
Total records: 731
```

```
Date range: 2016-01-01 00:00:00 to 2017-12-31 00:00:00

Rainfall Statistics:
count    731.0
mean       3.8
std        5.8
min        0.0
25%        0.0
50%        0.9
75%        5.3
max       36.6
Name: Rainfall, dtype: float64

Processed data saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION
\HISTORICAL DATA\processed\cleaned_data\rochdale_daily_rainfall.csv
```

In [107…

```
import pandas as pd
import os

def clean_peak_flows(base_path):
    stations = {
        'Manchester Racecourse': {
            'folder': 'manchester',
            'peak_file': 'manchester_peak_flow.csv'
        },
        'Rochdale': {
            'folder': 'rochdale',
            'peak_file': 'rochdale_peak_flow.csv'
        },
        'Bury Ground': {
            'folder': 'bury',
            'peak_file': 'bury_peak_flow.csv'
        }
    }

    # Create output directory
    output_dir = os.path.join(base_path, 'cleaned_data')
    os.makedirs(output_dir, exist_ok=True)

    # Process Peak Flow Data
    for station_name, info in stations.items():
        station_path = os.path.join(base_path, info['folder'])
        peak_file = os.path.join(station_path, info['peak_file'])

        if os.path.exists(peak_file):
            print(f"\nProcessing Peak Flow data for {station_name}")
            try:
                peak_df = pd.read_csv(peak_file)

                # Convert date columns
                peak_df['Date'] = pd.to_datetime(peak_df['Date'])
                peak_df['Datetime'] = pd.to_datetime(peak_df['Datetime'])

                # Rename and reorder columns for consistency
                peak_df = peak_df[['Water Year', 'Date', 'Time', 'Stage (m)', 'F

                # Basic analysis
                print(f"Total records: {len(peak_df)}")
                print(f"Date range: {peak_df['Date'].min()} to {peak_df['Date'].
                print("\nFlow Statistics:")
```

```
                print(peak_df['Flow (m3/s)'].describe())

                # Save to cleaned data
                output_file = os.path.join(output_dir, f'{info["folder"]}_peak_f
                peak_df.to_csv(output_file, index=False)
                print(f"Saved to: {output_file}")

            except Exception as e:
                print(f"Error processing {station_name}: {str(e)}")

# Base path for processed historical data
base_path = r"C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA

# Run the processing
clean_peak_flows(base_path)
```

```
Processing Peak Flow data for Manchester Racecourse
Total records: 82
Date range: 1941-10-24 00:00:00 to 2023-01-10 00:00:00

Flow Statistics:
count     82.0
mean     279.4
std       87.4
min      135.0
25%      217.3
50%      273.5
75%      327.3
max      560.0
Name: Flow (m3/s), dtype: float64
Saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DAT
A\processed\cleaned_data\manchester_peak_flow.csv

Processing Peak Flow data for Rochdale
Total records: 31
Date range: 1993-09-13 00:00:00 to 2023-07-23 00:00:00

Flow Statistics:
count    31.0
mean     46.4
std      15.0
min      18.0
25%      38.1
50%      44.7
75%      51.3
max      92.8
Name: Flow (m3/s), dtype: float64
Saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DAT
A\processed\cleaned_data\rochdale_peak_flow.csv

Processing Peak Flow data for Bury Ground
Total records: 51
Date range: 1973-01-12 00:00:00 to 2023-07-23 00:00:00

Flow Statistics:
count     51.0
mean     115.9
std       43.6
min       51.5
25%       84.6
50%      112.9
75%      125.6
max      283.6
Name: Flow (m3/s), dtype: float64
Saved to: C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICAL DAT
A\processed\cleaned_data\bury_peak_flow.csv
```

Processing Real Time Data for Rainfall and River Flow

In [ ]:

```python
import os
import pandas as pd

# Path to combined data
combined_data_path = r'C:\Users\Administrator\NEWPROJECT\combined_data'
```

```
# Function to inspect CSV files in detail
def detailed_csv_inspection(directory):
    print("=" * 50)
    print(f"Inspecting CSV files in: {directory}")
    print("=" * 50)

    # List all files in the directory
    files = [f for f in os.listdir(directory) if f.endswith('.csv')]

    if not files:
        print("No CSV files found in the directory.")
        return

    for filename in files:
        filepath = os.path.join(directory, filename)

        try:
            # Read the CSV file
            df = pd.read_csv(filepath)

            print("\n" + "=" * 40)
            print(f"File: {filename}")
            print("=" * 40)

            # Basic file information
            print(f"Total records: {len(df)}")

            # Column names and types
            print("\nColumns:")
            for col in df.columns:
                print(f"- {col}: {df[col].dtype}")

            # Check for datetime columns
            datetime_cols = df.select_dtypes(include=['datetime64']).columns
            if len(datetime_cols) == 0:
                # Try to identify potential datetime columns
                potential_datetime_cols = [
                    col for col in df.columns
                    if 'date' in col.lower() or 'time' in col.lower()
                ]
                if potential_datetime_cols:
                    print("\nPotential datetime columns (not parsed):")
                    for col in potential_datetime_cols:
                        print(f"- {col}")

            # First few rows
            print("\nFirst 5 rows:")
            print(df.head())

            # Basic statistics for numeric columns
            numeric_cols = df.select_dtypes(include=['float64', 'int64']).column
            if len(numeric_cols) > 0:
                print("\nNumeric Columns Statistics:")
                print(df[numeric_cols].describe())

            # Check for missing values
            missing_values = df.isnull().sum()
            print("\nMissing Values:")
            print(missing_values[missing_values > 0])
```

```
        except Exception as e:
            print(f"Error reading {filename}: {e}")

# Run the inspection
detailed_csv_inspection(combined_data_path)
```

In [109…

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

def clean_and_process_real_time_data(combined_data_dir):
    """
    Clean and process real-time data collection files
    """
    # Collect all CSV files
    csv_files = [f for f in os.listdir(combined_data_dir) if f.startswith('combi

    # List to store dataframes
    dataframes = []

    # Process each file
    for file in csv_files:
        file_path = os.path.join(combined_data_dir, file)
        try:
            # Read CSV file
            df = pd.read_csv(file_path)

            # Convert timestamps to datetime
            df['river_timestamp'] = pd.to_datetime(df['river_timestamp'], utc=Tr
            df['rainfall_timestamp'] = pd.to_datetime(df['rainfall_timestamp'],

            # Add file collection timestamp as a column
            df['collection_timestamp'] = pd.to_datetime(file.split('_')[2], form

            dataframes.append(df)
        except Exception as e:
            print(f"Error processing {file}: {e}")

    # Combine all dataframes
    if dataframes:
        combined_df = pd.concat(dataframes, ignore_index=True)

        # Data Cleaning Steps
        # 1. Remove duplicates
        combined_df.drop_duplicates(subset=['river_timestamp', 'location_name'],

        # 2. Handle missing values
        # Replace zero rainfall with NaN
        combined_df.loc[combined_df['rainfall'] == 0, 'rainfall'] = np.nan

        # 3. Data Type Conversion
        combined_df['river_level'] = pd.to_numeric(combined_df['river_level'], e
        combined_df['rainfall'] = pd.to_numeric(combined_df['rainfall'], errors=

        # 4. Sort by timestamp
        combined_df.sort_values('river_timestamp', inplace=True)

        # 5. Reset index
        combined_df.reset_index(drop=True, inplace=True)
```

```
        return combined_df

    return None

def integrate_with_historical_data(real_time_df, historical_peak_flow_files):
    """
    Integrate real-time data with historical peak flow data
    """
    # Load historical peak flow data for each station
    historical_dfs = {}
    for file in historical_peak_flow_files:
        station_name = file.split('_')[0].capitalize()
        historical_dfs[station_name] = pd.read_csv(file)
        # Convert historical data dates
        historical_dfs[station_name]['Date'] = pd.to_datetime(historical_dfs[sta

    # Integrate real-time data
    integrated_data = {}
    for station in real_time_df['location_name'].unique():
        # Real-time data for the station
        real_time_station_data = real_time_df[real_time_df['location_name'] == s

        # Historical data for the station
        historical_station_data = historical_dfs.get(station, pd.DataFrame())

        # Combine datasets
        if not historical_station_data.empty:
            # Rename columns to match
            historical_station_data = historical_station_data.rename(columns={
                'Date': 'river_timestamp',
                'Flow (m3/s)': 'historical_flow',
                'Stage (m)': 'historical_stage'
            })

            # Merge datasets
            merged_data = pd.merge_asof(
                real_time_station_data.sort_values('river_timestamp'),
                historical_station_data.sort_values('river_timestamp'),
                on='river_timestamp',
                direction='nearest'
            )

            integrated_data[station] = merged_data

    return integrated_data

def main_data_processing():
    # Directories and file paths
    combined_data_dir = r'C:\Users\Administrator\NEWPROJECT\combined_data'
    historical_data_dir = r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLEC
    output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data'

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Clean real-time data
    real_time_df = clean_and_process_real_time_data(combined_data_dir)

    if real_time_df is not None:
        # Save cleaned real-time data
```

```
        real_time_output_path = os.path.join(output_dir, 'cleaned_real_time_data
        real_time_df.to_csv(real_time_output_path, index=False)
        print(f"Cleaned real-time data saved to: {real_time_output_path}")

        # Historical peak flow files
        historical_peak_flow_files = [
            r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA
            r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA
            r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\HISTORICA
        ]

        # Integrate with historical data
        integrated_data = integrate_with_historical_data(real_time_df, historica

        # Save integrated data for each station
        for station, data in integrated_data.items():
            integrated_output_path = os.path.join(output_dir, f'{station.lower()
            data.to_csv(integrated_output_path, index=False)
            print(f"Integrated data for {station} saved to: {integrated_output_p

# Run the main processing
main_data_processing()
```

```
Cleaned real-time data saved to: C:\Users\Administrator\NEWPROJECT\processed_data
\cleaned_real_time_data.csv
```

In [111…

```
import os
import shutil
import pandas as pd

def organize_existing_cleaned_data():
    # Source directories
    historical_data_dir = r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLEC
    real_time_data_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data'

    # Unified output directory
    unified_output_dir = r'C:\Users\Administrator\NEWPROJECT\cleaned_data'

    # Create subdirectories
    river_historical_dir = os.path.join(unified_output_dir, 'river_data', 'histo
    river_realtime_dir = os.path.join(unified_output_dir, 'river_data', 'real_ti

    os.makedirs(river_historical_dir, exist_ok=True)
    os.makedirs(river_realtime_dir, exist_ok=True)

    # Move historical river data files
    historical_files = [
        'bury_daily_flow.csv',
        'bury_daily_rainfall.csv',
        'rochdale_daily_flow.csv',
        'rochdale_daily_rainfall.csv',
        'manchester_peak_flow.csv',
        'bury_peak_flow.csv',
        'rochdale_peak_flow.csv'
    ]

    for file in historical_files:
        source_path = os.path.join(historical_data_dir, file)
        dest_path = os.path.join(river_historical_dir, file)
```

```
        if os.path.exists(source_path):
            shutil.copy2(source_path, dest_path)
            print(f"Copied historical file: {file}")

    # Move real-time data files
    realtime_files = [
        'cleaned_real_time_data.csv'
    ]

    for file in realtime_files:
        source_path = os.path.join(real_time_data_dir, file)
        dest_path = os.path.join(river_realtime_dir, file)

        if os.path.exists(source_path):
            shutil.copy2(source_path, dest_path)
            print(f"Copied real-time file: {file}")

    print("\nData organization complete.")

# Run the data organization
organize_existing_cleaned_data()
```

```
Copied historical file: bury_daily_flow.csv
Copied historical file: bury_daily_rainfall.csv
Copied historical file: rochdale_daily_flow.csv
Copied historical file: rochdale_daily_rainfall.csv
Copied historical file: manchester_peak_flow.csv
Copied historical file: bury_peak_flow.csv
Copied historical file: rochdale_peak_flow.csv
Copied real-time file: cleaned_real_time_data.csv

Data organization complete.
```

In [113…

```
import os
import pandas as pd

# Base path for cleaned data
cleaned_data_base = r'C:\Users\Administrator\NEWPROJECT\cleaned_data'

# Function to explore datasets
def explore_cleaned_data(base_path):
    print("Cleaned Data Exploration:\n")

    # River Data - Historical
    historical_dir = os.path.join(base_path, 'river_data', 'historical')
    historical_files = os.listdir(historical_dir)

    print("Historical River Data Files:")
    for file in historical_files:
        file_path = os.path.join(historical_dir, file)
        df = pd.read_csv(file_path)
        print(f"\n{file}:")
        print(f"  - Total Records: {len(df)}")
        print(f"  - Columns: {list(df.columns)}")

        # Date range for time-series files
        if 'Date' in df.columns or 'date' in df.columns:
            date_col = 'Date' if 'Date' in df.columns else 'date'
            df[date_col] = pd.to_datetime(df[date_col])
            print(f"  - Date Range: {df[date_col].min()} to {df[date_col].max()}
```

```
    # Real-Time Data
    realtime_dir = os.path.join(base_path, 'river_data', 'real_time')
    realtime_files = os.listdir(realtime_dir)

    print("\nReal-Time River Data Files:")
    for file in realtime_files:
        file_path = os.path.join(realtime_dir, file)
        df = pd.read_csv(file_path)
        print(f"\n{file}:")
        print(f"  - Total Records: {len(df)}")
        print(f"  - Columns: {list(df.columns)}")

        # Timestamp range
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
        print(f"  - Timestamp Range: {df['river_timestamp'].min()} to {df['river

    # Weather Data
    weather_dir = os.path.join(base_path, 'weather_data')
    print("\nWeather Data Files:")
    for category in os.listdir(weather_dir):
        category_path = os.path.join(weather_dir, category)
        files = os.listdir(category_path)
        for file in files:
            file_path = os.path.join(category_path, file)
            df = pd.read_csv(file_path)
            print(f"\n{file}:")
            print(f"  - Total Records: {len(df)}")
            print(f"  - Columns: {list(df.columns)}")

    # Flood History Data
    flood_history_dir = os.path.join(base_path, 'flood_history')
    flood_files = os.listdir(flood_history_dir)

    print("\nFlood History Data Files:")
    for file in flood_files:
        file_path = os.path.join(flood_history_dir, file)
        df = pd.read_csv(file_path)
        print(f"\n{file}:")
        print(f"  - Total Records: {len(df)}")
        print(f"  - Columns: {list(df.columns)}")

# Run the exploration
explore_cleaned_data(cleaned_data_base)
```

```
Cleaned Data Exploration:

Historical River Data Files:

bury_daily_flow.csv:
  - Total Records: 9928
  - Columns: ['Date', 'Flow', 'Extra']
  - Date Range: 1995-11-22 00:00:00 to 2023-09-30 00:00:00

bury_daily_rainfall.csv:
  - Total Records: 20819
  - Columns: ['Date', 'Rainfall', 'Extra']
  - Date Range: 1961-01-01 00:00:00 to 2017-12-31 00:00:00

bury_peak_flow.csv:
  - Total Records: 51
  - Columns: ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating',
'Datetime']
  - Date Range: 1973-01-12 00:00:00 to 2023-07-23 00:00:00

manchester_peak_flow.csv:
  - Total Records: 82
  - Columns: ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating',
'Datetime']
  - Date Range: 1941-10-24 00:00:00 to 2023-01-10 00:00:00

rochdale_daily_flow.csv:
  - Total Records: 11118
  - Columns: ['Date', 'Flow', 'Extra']
  - Date Range: 1993-02-26 00:00:00 to 2023-09-30 00:00:00

rochdale_daily_rainfall.csv:
  - Total Records: 731
  - Columns: ['Date', 'Rainfall', 'Extra']
  - Date Range: 2016-01-01 00:00:00 to 2017-12-31 00:00:00

rochdale_peak_flow.csv:
  - Total Records: 31
  - Columns: ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating',
'Datetime']
  - Date Range: 1993-09-13 00:00:00 to 2023-07-23 00:00:00

Real-Time River Data Files:

cleaned_real_time_data.csv:
  - Total Records: 390
  - Columns: ['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp',
'location_name', 'river_station_id', 'rainfall_station_id', 'collection_timestam
p']
  - Timestamp Range: 2025-01-30 11:15:00+00:00 to 2025-01-31 21:00:00+00:00

Weather Data Files:

standardized_precipitation.csv:
  - Total Records: 42
  - Columns: ['precipitation_data', 'unnamed:_1']

standardized_temperature.csv:
  - Total Records: 43
  - Columns: ['temperation', 'unnamed:_1']
```

```
Flood History Data Files:

standardized_flood_history.csv:
  - Total Records: 500
  - Columns: ['label', 'description', 'river', 'area_type', 'quick_dial', 'count
y', 'source_file']
```

In [122…

```python
import pandas as pd
import os

def examine_weather_files():
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    weather_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'WEATHER

    # Read files
    temp_file = os.path.join(weather_path, 'TEMPERATURE.xlsx')
    precip_file = os.path.join(weather_path, 'PRECIPITATION.xlsx')

    temp_df = pd.read_excel(temp_file)
    precip_df = pd.read_excel(precip_file)

    print("Temperature Data Structure:")
    print("=========================")
    print(temp_df.head(20))
    print("\nShape:", temp_df.shape)

    print("\nPrecipitation Data Structure:")
    print("============================")
    print(precip_df.head(20))
    print("\nShape:", precip_df.shape)

examine_weather_files()
```

```
Temperature Data Structure:
=========================
                                           TEMPERATION Unnamed: 1
0   WEBSITE: https://climate-themetoffice.hub.arcg...        NaN
1                                                 NaN        NaN
2                                                 NaN        NaN
3                                 MANCHESTER RACECOURSE        NaN
4                                             GRID_ID      AX-71
5                                         tas Jaunary          5
6                                        tas February        5.4
7                                           tas March          7
8                                           tas April        9.4
9                                             tas May       12.4
10                                           tas June         15
11                                           tas July       16.8
12                                         tas August       16.5
13                                      tas September       14.2
14                                        tas October         11
15                                       tas November        7.6
16                                       tas December        5.3
17                                    BURY MANCHESTER        NaN
18                                            GRID_ID      AX-70
19                                        tas Jaunary        3.8

Shape: (48, 2)

Precipitation Data Structure:
============================
         PRECIPITATION DATA Unnamed: 1
0       MANCHESTER RECOURSE        NaN
1                   GRID_ID      AX-71
2                pr January         90
3               pr February         76
4                  pr March         66
5                  pr April         59
6                    pr May         64
7                   pr June         77
8                   pr July         84
9                 pr August         85
10             pr September         85
11               pr October        101
12              pr November         97
13              pr December        108
14                      NaN        NaN
15                      NaN        NaN
16  BURY GREATER MANCHESTER        NaN
17                  GRID_ID      AX-70
18               pr January        131
19              pr February        112

Shape: (46, 2)
```

In [123…

```
import pandas as pd
import os

def clean_weather_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    weather_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'WEATHER
    output_path = os.path.join(project_path, 'cleaned_data')
```

```
def process_temperature_data(df):
    """Process temperature data"""
    data = []
    current_station = None
    current_grid = None

    for idx, row in df.iterrows():
        if pd.notna(row['TEMPERATION']) and 'GRID_ID' not in str(row['TEMPER
            if 'MANCHESTER' in str(row['TEMPERATION']) or 'BURY' in str(row[
                current_station = row['TEMPERATION']
            elif 'GRID_ID' in str(row['Unnamed: 1']):
                current_grid = row['Unnamed: 1']
            elif 'tas' in str(row['TEMPERATION']):
                month = row['TEMPERATION'].replace('tas ', '')
                value = row['Unnamed: 1']
                data.append({
                    'Station': current_station,
                    'Grid_ID': current_grid,
                    'Month': month,
                    'Temperature_C': value,
                    'Parameter': 'Temperature',
                    'Unit': '°C',
                    'Grid': '12km BNG',
                    'Period': '1991-2020'
                })

    return pd.DataFrame(data)

def process_precipitation_data(df):
    """Process precipitation data"""
    data = []
    current_station = None
    current_grid = None

    for idx, row in df.iterrows():
        if pd.notna(row['PRECIPITATION DATA']):
            if 'MANCHESTER' in str(row['PRECIPITATION DATA']) or 'BURY' in s
                current_station = row['PRECIPITATION DATA']
            elif 'GRID_ID' in str(row['PRECIPITATION DATA']):
                current_grid = row['Unnamed: 1']
            elif 'pr' in str(row['PRECIPITATION DATA']):
                month = row['PRECIPITATION DATA'].replace('pr ', '')
                value = row['Unnamed: 1']
                data.append({
                    'Station': current_station,
                    'Grid_ID': current_grid,
                    'Month': month,
                    'Precipitation_mm': value,
                    'Parameter': 'Precipitation',
                    'Unit': 'mm',
                    'Grid': '2km BNG',
                    'Period': '1991-2020'
                })

    return pd.DataFrame(data)

# Read and process data
temp_df = pd.read_excel(os.path.join(weather_path, 'TEMPERATURE.xlsx'))
precip_df = pd.read_excel(os.path.join(weather_path, 'PRECIPITATION.xlsx'))
```

```
    temp_clean = process_temperature_data(temp_df)
    precip_clean = process_precipitation_data(precip_df)

    # Save cleaned data
    temp_clean.to_csv(os.path.join(output_path, 'cleaned_temperature.csv'), inde
    precip_clean.to_csv(os.path.join(output_path, 'cleaned_precipitation.csv'),

    # Print summary
    print("Weather Data Summary (1991-2020):")
    print("================================")
    print("\nTemperature Data:")
    print(f"Total records: {len(temp_clean)}")
    for station in temp_clean['Station'].unique():
        print(f"\n{station} (Grid: {temp_clean[temp_clean['Station'] == station]
        station_data = temp_clean[temp_clean['Station'] == station]
        print(station_data[['Month', 'Temperature_C']].to_string(index=False))

    print("\nPrecipitation Data:")
    print(f"Total records: {len(precip_clean)}")
    for station in precip_clean['Station'].unique():
        print(f"\n{station} (Grid: {precip_clean[precip_clean['Station'] == stat
        station_data = precip_clean[precip_clean['Station'] == station]
        print(station_data[['Month', 'Precipitation_mm']].to_string(index=False)

clean_weather_data()
```

```
Weather Data Summary (1991-2020):
================================

Temperature Data:
Total records: 36

MANCHESTER RACECOURSE (Grid: None):
    Month  Temperature_C
  Jaunary            5.0
 February            5.4
    March            7.0
    April            9.4
      May           12.4
     June           15.0
     July           16.8
   August           16.5
September           14.2
  October           11.0
 November            7.6
 December            5.3

BURY MANCHESTER (Grid: None):
    Month  Temperature_C
  Jaunary            3.8
 February            4.1
    March            5.7
    April            8.1
      May           11.0
     June           13.6
     July           15.5
   August           15.2
September           12.9
  October            9.7
 November            6.5
 December            4.1

ROCHDALE (Grid: None):
    Month  Temperature_C
  Jaunary            3.6
 February            3.9
    March            5.4
    April            7.9
      May           10.7
     June           13.4
     July           15.3
   August           15.1
September           12.8
  October            9.6
 November            6.2
 December            4.0

Precipitation Data:
Total records: 36

MANCHESTER RECOURSE (Grid: AX-71):
    Month  Precipitation_mm
  January                90
 February                76
    March                66
    April                59
```

```
      May                 64
     June                 77
     July                 84
   August                 85
September                 85
  October                101
 November                 97
 December                108

BURY GREATER MANCHESTER (Grid: AX-70):
    Month  Precipitation_mm
  January               131
 February               112
    March                95
    April                79
      May                83
     June                93
     July               100
   August               111
September               110
  October               134
 November               138
 December               157

ROCHDALE  (Grid: AY-70):
    Month  Precipitation_mm
  January               131
 February               110
    March                96
    April                77
      May                77
     June                92
     July               105
   August               110
September               109
  October               130
 November               136
 December               154
```

In [127…

```python
import pandas as pd
import os

def clean_weather_data():
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    weather_path = os.path.join(project_path, 'MANUAL DATA COLLECTION', 'WEATHER
    output_path = os.path.join(project_path, 'cleaned_data')

    # Define standard structure
    stations = {
        'MANCHESTER RACECOURSE': {'grid_id': 'AX-71', 'alias': ['MANCHESTER RECO
        'BURY MANCHESTER': {'grid_id': 'AX-70', 'alias': ['BURY GREATER MANCHEST
        'ROCHDALE': {'grid_id': 'AY-70', 'alias': []}
    }

    months = [
        'January', 'February', 'March', 'April', 'May', 'June',
        'July', 'August', 'September', 'October', 'November', 'December'
    ]

    def standardize_station_name(name):
```

```
    """Convert various station names to standard format"""
    for std_name, info in stations.items():
        if any(alias in name for alias in [std_name] + info['alias']):
            return std_name
    return name

def extract_data(df, data_type='temperature'):
    """Extract data from raw format"""
    data = []
    current_station = None
    current_grid = None

    col1_name = 'TEMPERATION' if data_type == 'temperature' else 'PRECIPITAT

    for idx, row in df.iterrows():
        col1 = str(row[col1_name]).strip()
        col2 = row['Unnamed: 1']

        if pd.notna(col1):
            if any(station in col1 or any(alias in col1 for alias in info['a
                   for station, info in stations.items()):
                current_station = standardize_station_name(col1)
            elif 'GRID_ID' in col1:
                current_grid = col2
            elif ('tas' in col1 and data_type == 'temperature') or ('pr' in
                month = col1.replace('tas ', '').replace('pr ', '').replace(
                if month in months:
                    data.append({
                        'Month': month,
                        'Station': current_station,
                        'Grid_ID': stations[current_station]['grid_id'],
                        'Value': float(col2),
                        'Grid': '12km BNG' if data_type == 'temperature' els
                        'Period': '1991-2020'
                    })
    return pd.DataFrame(data)

# Read and process data
temp_df = pd.read_excel(os.path.join(weather_path, 'TEMPERATURE.xlsx'))
precip_df = pd.read_excel(os.path.join(weather_path, 'PRECIPITATION.xlsx'))

# Clean and standardize data
temp_clean = extract_data(temp_df, 'temperature').rename(columns={'Value': '
precip_clean = extract_data(precip_df, 'precipitation').rename(columns={'Val

# Create combined weather data
weather_combined = pd.merge(
    temp_clean,
    precip_clean[['Month', 'Station', 'Precipitation_mm']],
    on=['Month', 'Station']
)

# Sort data
weather_combined = weather_combined.sort_values(['Month', 'Station']).reset_

# Save files
weather_combined.to_csv(os.path.join(output_path, 'cleaned_weather_combined.

# Print summary
print("Weather Data Summary (1991-2020):")
```

```
    print("================================")
    for month in months:
        print(f"\n{month}:")
        month_data = weather_combined[weather_combined['Month'] == month]
        print("\nStation               Grid_ID  Temp(°C)  Precip(mm)")
        print("---------------------------------------------------")
        for _, row in month_data.iterrows():
            print(f"{row['Station']:<20} {row['Grid_ID']}    {row['Temperature_C

clean_weather_data()
```

Weather Data Summary (1991-2020):
================================

January:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	3.8	131.0
MANCHESTER RACECOURSE	AX-71	5.0	90.0
ROCHDALE	AY-70	3.6	131.0

February:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	4.1	112.0
MANCHESTER RACECOURSE	AX-71	5.4	76.0
ROCHDALE	AY-70	3.9	110.0

March:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	5.7	95.0
MANCHESTER RACECOURSE	AX-71	7.0	66.0
ROCHDALE	AY-70	5.4	96.0

April:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	8.1	79.0
MANCHESTER RACECOURSE	AX-71	9.4	59.0
ROCHDALE	AY-70	7.9	77.0

May:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	11.0	83.0
MANCHESTER RACECOURSE	AX-71	12.4	64.0
ROCHDALE	AY-70	10.7	77.0

June:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	13.6	93.0
MANCHESTER RACECOURSE	AX-71	15.0	77.0
ROCHDALE	AY-70	13.4	92.0

July:

Station	Grid_ID	Temp(°C)	Precip(mm)
BURY MANCHESTER	AX-70	15.5	100.0
MANCHESTER RACECOURSE	AX-71	16.8	84.0
ROCHDALE	AY-70	15.3	105.0

August:

```
Station                Grid_ID  Temp(°C)  Precip(mm)
---------------------------------------------------
BURY MANCHESTER       AX-70      15.2      111.0
MANCHESTER RACECOURSE AX-71       16.5       85.0
ROCHDALE              AY-70      15.1      110.0

September:

Station                Grid_ID  Temp(°C)  Precip(mm)
---------------------------------------------------
BURY MANCHESTER       AX-70      12.9      110.0
MANCHESTER RACECOURSE AX-71       14.2       85.0
ROCHDALE              AY-70      12.8      109.0

October:

Station                Grid_ID  Temp(°C)  Precip(mm)
---------------------------------------------------
BURY MANCHESTER       AX-70       9.7      134.0
MANCHESTER RACECOURSE AX-71       11.0      101.0
ROCHDALE              AY-70       9.6      130.0

November:

Station                Grid_ID  Temp(°C)  Precip(mm)
---------------------------------------------------
BURY MANCHESTER       AX-70       6.5      138.0
MANCHESTER RACECOURSE AX-71        7.6       97.0
ROCHDALE              AY-70       6.2      136.0

December:

Station                Grid_ID  Temp(°C)  Precip(mm)
---------------------------------------------------
BURY MANCHESTER       AX-70       4.1      157.0
MANCHESTER RACECOURSE AX-71        5.3      108.0
ROCHDALE              AY-70       4.0      154.0
```

In [130…

```python
import pandas as pd
import json
import os
from datetime import datetime
import re

class FloodDataCleaner:
    def __init__(self, project_path):
        """Initialize with project path and Greater Manchester specifics"""
        self.project_path = project_path
        self.flood_path = os.path.join(project_path, 'flood_data')
        self.output_path = os.path.join(project_path, 'cleaned_data')

        # Define Greater Manchester specific information
        self.gm_areas = [
            'Manchester', 'Salford', 'Bolton', 'Bury', 'Rochdale',
            'Oldham', 'Tameside', 'Stockport', 'Trafford', 'Wigan'
        ]

        self.gm_rivers = [
            'River Irwell', 'River Roch', 'River Irk', 'River Medlock',
```

```
            'River Mersey', 'River Tame', 'River Croal', 'River Douglas',
            'Astley Brook', 'Hey Brook', 'Cringle Brook'
        ]

        # Create output directory if it doesn't exist
        os.makedirs(self.output_path, exist_ok=True)

    def clean_river_name(self, river):
        """Standardize river names"""
        if pd.isna(river):
            return None

        # Common corrections
        corrections = {
            'R. ': 'River ',
            'Riv. ': 'River ',
            'Rvr ': 'River '
        }

        cleaned = str(river).strip()
        for old, new in corrections.items():
            cleaned = cleaned.replace(old, new)

        # Handle multiple rivers in one field
        if ',' in cleaned:
            rivers = [r.strip() for r in cleaned.split(',')]
            return rivers[0]  # Take primary river

        return cleaned

    def is_greater_manchester_relevant(self, row):
        """Check if an area is relevant to Greater Manchester"""
        if pd.isna(row['label']) and pd.isna(row['description']):
            return False

        # Check various fields for Greater Manchester relevance
        text_to_check = ' '.join([
            str(row['label']),
            str(row['description']),
            str(row['river']),
            str(row['county'])
        ]).lower()

        # Check for GM areas
        if any(area.lower() in text_to_check for area in self.gm_areas):
            return True

        # Check for GM rivers
        if any(river.lower() in text_to_check for river in self.gm_rivers):
            return True

        return False

    def load_and_clean_json(self, filename):
        """Load and clean JSON flood data files"""
        try:
            with open(os.path.join(self.flood_path, filename), 'r') as f:
                data = json.load(f)

            # Convert JSON to DataFrame based on structure
```

```
            if isinstance(data, list):
                df = pd.json_normalize(data)
            else:
                df = pd.json_normalize([data])

            # Rename columns to match standard format
            column_mapping = {
                'floodArea.label': 'label',
                'floodArea.description': 'description',
                'floodArea.river': 'river',
                'floodArea.county': 'county',
                'floodArea.quickDial': 'quick_dial'
            }
            df = df.rename(columns=column_mapping)

            return df

        except Exception as e:
            print(f"Error processing {filename}: {e}")
            return None

    def determine_risk_level(self, description):
        """Determine flood risk level from description"""
        if pd.isna(description):
            return 'Unknown'

        description = description.lower()

        if any(word in description for word in ['severe', 'danger', 'extreme']):
            return 'High'
        elif any(word in description for word in ['warning', 'alert', 'caution']
            return 'Medium'
        elif any(word in description for word in ['monitoring', 'watch', 'possib
            return 'Low'
        else:
            return 'Standard'

    def assign_monitoring_station(self, row):
        """Assign relevant monitoring station(s) based on location"""
        text = f"{row['label']} {row['description']} {row['river']}".lower()

        stations = []
        if 'rochdale' in text or 'river roch' in text:
            stations.append('690203')  # Rochdale
        if 'manchester' in text or 'salford' in text:
            stations.append('690510')  # Manchester Racecourse
        if 'bury' in text or 'irwell' in text:
            stations.append('690160')  # Bury Ground

        return ','.join(stations) if stations else None

    def clean_flood_data(self):
        """Main cleaning function for all flood data"""
        print("Starting flood data cleaning process...")

        # Initialize empty list for all flood data
        all_data = []

        # Process each file in the flood data directory
        for filename in os.listdir(self.flood_path):
```

```
            if filename.endswith('.json'):
                df = self.load_and_clean_json(filename)
            elif filename.endswith('.csv'):
                try:
                    df = pd.read_csv(os.path.join(self.flood_path, filename))
                except Exception as e:
                    print(f"Error reading {filename}: {e}")
                    continue
            else:
                continue

            if df is not None:
                all_data.append(df)

        # Combine all data
        if not all_data:
            raise Exception("No data could be loaded")

        combined_df = pd.concat(all_data, ignore_index=True)

        # Clean and standardize
        combined_df['river'] = combined_df['river'].apply(self.clean_river_name)
        combined_df['risk_level'] = combined_df['description'].apply(self.determ
        combined_df['monitoring_stations'] = combined_df.apply(self.assign_monit

        # Filter for Greater Manchester relevance
        gm_df = combined_df[combined_df.apply(self.is_greater_manchester_relevan

        # Add metadata
        gm_df['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        gm_df['data_quality'] = gm_df.apply(
            lambda x: 'High' if pd.notna(x['river']) and pd.notna(x['monitoring_
            else 'Medium' if pd.notna(x['river']) or pd.notna(x['monitoring_stat
            else 'Low',
            axis=1
        )

        # Save cleaned data
        output_file = os.path.join(self.output_path, 'cleaned_flood_areas.csv')
        gm_df.to_csv(output_file, index=False)

        # Generate summary statistics
        summary = {
            'total_areas': len(gm_df),
            'rivers_covered': gm_df['river'].nunique(),
            'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
            'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stations
            'data_quality_distribution': gm_df['data_quality'].value_counts().tc
        }

        # Save summary
        summary_file = os.path.join(self.output_path, 'flood_data_summary.json')
        with open(summary_file, 'w') as f:
            json.dump(summary, f, indent=2)

        return gm_df, summary

def main():
    """Main function to run the cleaning process"""
    project_path = r"C:\Users\Administrator\NEWPROJECT"
```

```
    cleaner = FloodDataCleaner(project_path)

    try:
        df, summary = cleaner.clean_flood_data()
        print("\nCleaning completed successfully!")
        print("\nSummary of cleaned data:")
        print(json.dumps(summary, indent=2))

        print("\nSample of cleaned data:")
        print(df[['label', 'river', 'risk_level', 'monitoring_stations', 'data_q

    except Exception as e:
        print(f"Error during cleaning process: {e}")

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

```
Starting flood data cleaning process...

Cleaning completed successfully!

Summary of cleaned data:
{
  "total_areas": 52,
  "rivers_covered": 39,
  "high_risk_areas": 0,
  "areas_with_stations": 33,
  "data_quality_distribution": {
    "High": 32,
    "Medium": 20
  }
}

Sample of cleaned data:
                                                label                river  \
12  River Douglas at Wigan, between Scholes and Po...        River Douglas   
13                                Upper River Douglas        River Douglas   
15                             River Arrow and River Alne                Arrow   
17                               Upper Bristol Avon area  Bristol River Avon   
29  River Mersey at Fletcher Moss and Withington G...         River Mersey   

   risk_level monitoring_stations data_quality  
12   Standard                None       Medium  
13   Standard                None       Medium  
15   Standard              690510         High  
17   Standard              690160         High  
29   Standard              690160         High  
```

In [136…

```
import pandas as pd
import json
import os
from datetime import datetime
import re

class FloodDataCleaner:
    def __init__(self, project_path):
        """Initialize with project path and Greater Manchester specifics"""
        self.project_path = project_path
        self.flood_path = os.path.join(project_path, 'flood_data')
        self.output_path = os.path.join(project_path, 'cleaned_data')
```

```
        # Define Greater Manchester specific information
        self.gm_areas = {
            'primary': [
                'Manchester', 'Salford', 'Bury', 'Rochdale', 'Oldham',
                'Tameside', 'Stockport', 'Trafford', 'Bolton', 'Wigan'
            ],
            'secondary': [
                'Littleborough', 'Whitefield', 'Prestwich', 'Radcliffe',
                'Cheetham Hill', 'Blackley', 'Withington', 'Didsbury',
                'Fallowfield', 'Rusholme', 'Crumpsall', 'Middleton'
            ]
        }

        self.gm_rivers = {
            'primary': [
                'River Irwell', 'River Roch', 'River Irk', 'River Medlock',
                'River Mersey', 'River Tame', 'River Croal'
            ],
            'secondary': [
                'Irk', 'Medlock', 'Roch', 'Irwell', 'Mersey', 'Tame',
                'Croal', 'Cringle Brook', 'Chorlton Brook', 'Gore Brook'
            ]
        }

        # Define monitoring station catchments
        self.station_catchments = {
            '690203': {  # Rochdale
                'primary_areas': ['Rochdale', 'Littleborough', 'Heywood'],
                'rivers': ['River Roch'],
                'boundaries': ['North Manchester', 'East Manchester']
            },
            '690510': {  # Manchester Racecourse
                'primary_areas': ['Manchester', 'Salford', 'Cheetham Hill', 'Bla
                'rivers': ['River Irk', 'River Medlock', 'River Mersey'],
                'boundaries': ['Central Manchester', 'South Manchester']
            },
            '690160': {  # Bury Ground
                'primary_areas': ['Bury', 'Radcliffe', 'Whitefield'],
                'rivers': ['River Irwell'],
                'boundaries': ['North Manchester', 'Northwest Manchester']
            }
        }

        # Create output directory if it doesn't exist
        os.makedirs(self.output_path, exist_ok=True)

    def clean_river_name(self, river):
        """Standardize river names"""
        if pd.isna(river):
            return None

        # Common corrections
        corrections = {
            'R. ': 'River ',
            'Riv. ': 'River ',
            'Rvr ': 'River '
        }

        cleaned = str(river).strip()
```

```
        for old, new in corrections.items():
            cleaned = cleaned.replace(old, new)

        # Handle multiple rivers in one field
        if ',' in cleaned:
            rivers = [r.strip() for r in cleaned.split(',')]
            return rivers[0]  # Take primary river

        return cleaned

    def is_greater_manchester_relevant(self, row):
        """Check if an area is relevant to Greater Manchester"""
        if pd.isna(row['label']) and pd.isna(row['description']):
            return False

        # Create full text to check
        text_to_check = ' '.join([
            str(row.get('label', '')),
            str(row.get('description', '')),
            str(row.get('river', '')),
            str(row.get('county', ''))
        ]).lower()

        # Check for primary area matches
        has_primary_area = any(
            f" {area.lower()} " in f" {text_to_check} "
            for area in self.gm_areas['primary']
        )

        # Check for secondary area matches
        has_secondary_area = any(
            f" {area.lower()} " in f" {text_to_check} "
            for area in self.gm_areas['secondary']
        )

        # Check for primary river matches
        has_primary_river = any(
            f" {river.lower()} " in f" {text_to_check} "
            for river in self.gm_rivers['primary']
        )

        # Check for secondary river matches
        has_secondary_river = any(
            f" {river.lower()} " in f" {text_to_check} "
            for river in self.gm_rivers['secondary']
        )

        # Explicit Greater Manchester mention
        has_gm_mention = "greater manchester" in text_to_check

        # Decision logic:
        # 1. Must have either a primary area/river OR Greater Manchester mention
        # 2. If only secondary matches, must have both area and river
        primary_match = has_primary_area or has_primary_river or has_gm_mention
        secondary_match = has_secondary_area and has_secondary_river

        return primary_match or secondary_match

    def load_and_clean_json(self, filename):
        """Load and clean JSON flood data files"""
```

```
        try:
            with open(os.path.join(self.flood_path, filename), 'r') as f:
                data = json.load(f)

            # Convert JSON to DataFrame based on structure
            if isinstance(data, list):
                df = pd.json_normalize(data)
            else:
                df = pd.json_normalize([data])

            # Rename columns to match standard format
            column_mapping = {
                'floodArea.label': 'label',
                'floodArea.description': 'description',
                'floodArea.river': 'river',
                'floodArea.county': 'county',
                'floodArea.quickDial': 'quick_dial'
            }
            df = df.rename(columns=column_mapping)

            return df

        except Exception as e:
            print(f"Error processing {filename}: {e}")
            return None

    def determine_risk_level(self, row):
        """Determine flood risk level from description and other factors"""
        if pd.isna(row['description']):
            return 'Unknown'

        description = str(row['description']).lower()
        label = str(row['label']).lower()

        # Check for explicit risk indicators
        high_risk_phrases = [
            'severe flood', 'immediate action', 'flood warning',
            'danger to life', 'major flooding', 'high risk',
            'property flooding', 'flooding is expected'
        ]

        medium_risk_phrases = [
            'flood alert', 'rising levels', 'be prepared',
            'flooding possible', 'historical flooding',
            'river levels high', 'surface water'
        ]

        low_risk_phrases = [
            'monitoring', 'normal conditions',
            'routine monitoring', 'no immediate concern'
        ]

        # Check both description and label
        text_to_check = f"{description} {label}"

        # Check for high risk indicators
        if any(phrase in text_to_check for phrase in high_risk_phrases):
            return 'High'

        # Check for key infrastructure or vulnerable areas
```

```
        if any(term in text_to_check for term in ['hospital', 'school', 'care hc
            return 'High'

        # Check for medium risk indicators
        if any(phrase in text_to_check for phrase in medium_risk_phrases):
            return 'Medium'

        # Properties mentioned but no immediate risk
        if 'properties' in text_to_check and not any(phrase in text_to_check for
            return 'Medium'

        # Low risk if explicitly mentioned
        if any(phrase in text_to_check for phrase in low_risk_phrases):
            return 'Low'

        # Default to Medium if we're tracking it but no clear indicators
        return 'Medium'

    def assign_monitoring_station(self, row):
        """Assign relevant monitoring station(s) based on location and river"""
        try:
            text = ' '.join([
                str(row.get('label', '')),
                str(row.get('description', '')),
                str(row.get('river', ''))
            ]).lower()

            def check_catchment_match(catchment, text):
                """Helper to check how well a catchment matches the text"""
                score = 0
                # Check primary areas
                if any(area.lower() in text for area in catchment['primary_areas
                    score += 3
                # Check rivers
                if any(river.lower() in text for river in catchment['rivers']):
                    score += 2
                # Check boundaries
                if any(bound.lower() in text for bound in catchment['boundaries'
                    score += 1
                return score

            # Calculate match scores for each station
            station_scores = {
                station_id: check_catchment_match(catchment, text)
                for station_id, catchment in self.station_catchments.items()
            }

            # Special cases
            if 'crumpsall' in text or 'irk' in text:
                station_scores['690510'] += 2  # Boost Manchester Racecourse for
            if 'fletcher moss' in text or 'withington' in text or 'didsbury' in
                station_scores['690510'] += 2  # Boost Manchester Racecourse for
            if 'upper irwell' in text:
                station_scores['690160'] += 2  # Boost Bury Ground for upper Irw

            # Get best matching station if any score > 0
            best_station = max(station_scores.items(), key=lambda x: x[1])
            return best_station[0] if best_station[1] > 0 else None

        except Exception as e:
```

```
            print(f"Error assigning monitoring station: {e}")
            return None

    def clean_flood_data(self):
        """Main cleaning function for all flood data"""
        try:
            print("Starting flood data cleaning process...")

            # Initialize empty list for all flood data
            all_data = []

            # Process each file in the flood data directory
            for filename in os.listdir(self.flood_path):
                if filename.endswith('.json'):
                    df = self.load_and_clean_json(filename)
                elif filename.endswith('.csv'):
                    try:
                        df = pd.read_csv(os.path.join(self.flood_path, filename)
                    except Exception as e:
                        print(f"Error reading {filename}: {e}")
                        continue
                else:
                    continue

                if df is not None:
                    # Ensure required columns exist
                    required_columns = ['label', 'description', 'river']
                    for col in required_columns:
                        if col not in df.columns:
                            df[col] = None
                    all_data.append(df)

            # Combine all data
            if not all_data:
                raise Exception("No data could be loaded")

            combined_df = pd.concat(all_data, ignore_index=True)

            # Clean and standardize
            print("Cleaning river names...")
            combined_df['river'] = combined_df['river'].apply(self.clean_river_n

            # Filter for Greater Manchester relevance
            print("Filtering for Greater Manchester relevance...")
            gm_df = combined_df[combined_df.apply(self.is_greater_manchester_rel

            if len(gm_df) == 0:
                raise Exception("No Greater Manchester relevant data found after

            # Process GM relevant data
            print("Assigning risk levels...")
            gm_df['risk_level'] = gm_df.apply(self.determine_risk_level, axis=1)

            print("Assigning monitoring stations...")
            gm_df['monitoring_stations'] = gm_df.apply(self.assign_monitoring_st

            # Add metadata
            print("Adding metadata...")
            gm_df['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            gm_df['data_quality'] = gm_df.apply(
```

```
                lambda x: 'High' if pd.notna(x['river']) and pd.notna(x['monitor
                else 'Medium' if pd.notna(x['river']) or pd.notna(x['monitoring_
                else 'Low',
                axis=1
            )

            # Save cleaned data
            print("Saving cleaned data...")
            output_file = os.path.join(self.output_path, 'cleaned_flood_areas.cs
            gm_df.to_csv(output_file, index=False)

            # Generate summary statistics
            summary = {
                'total_areas': len(gm_df),
                'rivers_covered': gm_df['river'].nunique(),
                'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
                'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stat
                'data_quality_distribution': gm_df['data_quality'].value_counts(
            }

            # Save summary
            summary_file = os.path.join(self.output_path, 'flood_data_summary.js
            with open(summary_file, 'w') as f:
                json.dump(summary, f, indent=2)

            return gm_df, summary

        except Exception as e:
            print(f"Error during cleaning process: {e}")
            return None, None

        # Filter for Greater Manchester relevance
        gm_df = combined_df[combined_df.apply(self.is_greater_manchester_relevan

        # Add metadata
        gm_df['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        gm_df['data_quality'] = gm_df.apply(
            lambda x: 'High' if pd.notna(x['river']) and pd.notna(x['monitoring_
            else 'Medium' if pd.notna(x['river']) or pd.notna(x['monitoring_stat
            else 'Low',
            axis=1
        )

        # Save cleaned data
        output_file = os.path.join(self.output_path, 'cleaned_flood_areas.csv')
        gm_df.to_csv(output_file, index=False)

        # Generate summary statistics
        summary = {
            'total_areas': len(gm_df),
            'rivers_covered': gm_df['river'].nunique(),
            'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
            'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stations
            'data_quality_distribution': gm_df['data_quality'].value_counts().tc
        }

        # Save summary
        summary_file = os.path.join(self.output_path, 'flood_data_summary.json')
        with open(summary_file, 'w') as f:
            json.dump(summary, f, indent=2)
```

```python
        return gm_df, summary

def main():
    """Main function to run the cleaning process"""
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    cleaner = FloodDataCleaner(project_path)

    try:
        df, summary = cleaner.clean_flood_data()
        print("\nCleaning completed successfully!")
        print("\nSummary of cleaned data:")
        print(json.dumps(summary, indent=2))

        print("\nSample of cleaned data:")
        print(df[['label', 'river', 'risk_level', 'monitoring_stations', 'data_q

    except Exception as e:
        print(f"Error during cleaning process: {e}")

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

```
Starting flood data cleaning process...
Cleaning river names...
Filtering for Greater Manchester relevance...
Assigning risk levels...
Assigning monitoring stations...
Adding metadata...
Saving cleaned data...

Cleaning completed successfully!

Summary of cleaned data:
{
  "total_areas": 27,
  "rivers_covered": 17,
  "high_risk_areas": 1,
  "areas_with_stations": 15,
  "data_quality_distribution": {
    "High": 15,
    "Medium": 12
  }
}

Sample of cleaned data:
                                                label          river  \
12  River Douglas at Wigan, between Scholes and Po...  River Douglas
13                              Upper River Douglas  River Douglas
15                            River Arrow and River Alne          Arrow
29  River Mersey at Fletcher Moss and Withington G...   River Mersey
30  Middle River Mersey catchment including areas ...   River Mersey

   risk_level monitoring_stations data_quality
12     Medium                None       Medium
13     Medium                None       Medium
15     Medium              690510         High
29     Medium              690510         High
30     Medium              690510         High
```

In [141…

```
import pandas as pd
import json
import os
from datetime import datetime
import re

class FloodDataCleaner:
    def __init__(self, project_path):
        """Initialize with project path and Greater Manchester specifics"""
        self.project_path = project_path
        self.flood_path = os.path.join(project_path, 'flood_data')
        self.output_path = os.path.join(project_path, 'cleaned_data')

        # Define Greater Manchester specific information
        self.gm_areas = {
            'primary': [
                'Manchester', 'Salford', 'Bury', 'Rochdale', 'Oldham',
                'Tameside', 'Stockport', 'Trafford', 'Bolton'
            ],
            'secondary': [
                'Littleborough', 'Whitefield', 'Prestwich', 'Radcliffe',
                'Cheetham Hill', 'Blackley', 'Withington', 'Didsbury',
                'Fallowfield', 'Rusholme', 'Crumpsall', 'Middleton'
            ]
        }

        # Only include rivers we're actually monitoring
        self.monitored_rivers = {
            '690203': ['River Roch'],  # Rochdale
            '690510': ['River Irk', 'River Medlock', 'River Mersey'],  # Manches
            '690160': ['River Irwell']  # Bury Ground
        }

        # Create flat list of all monitored rivers
        self.all_monitored_rivers = list(set([
            river for rivers in self.monitored_rivers.values()
            for river in rivers
        ]))

        # Define monitoring station catchments
        self.station_catchments = {
            '690203': {  # Rochdale
                'primary_areas': ['Rochdale', 'Littleborough', 'Heywood'],
                'rivers': ['River Roch'],
                'boundaries': ['North Manchester', 'East Manchester']
            },
            '690510': {  # Manchester Racecourse
                'primary_areas': ['Manchester', 'Salford', 'Cheetham Hill', 'Bla
                                  'Withington', 'Didsbury', 'Fallowfield'],
                'rivers': ['River Irk', 'River Medlock', 'River Mersey'],
                'boundaries': ['Central Manchester', 'South Manchester']
            },
            '690160': {  # Bury Ground
                'primary_areas': ['Bury', 'Radcliffe', 'Whitefield', 'Prestwich'
                'rivers': ['River Irwell'],
                'boundaries': ['North Manchester', 'Northwest Manchester']
            }
        }
```

```
        # Create output directory if it doesn't exist
        os.makedirs(self.output_path, exist_ok=True)

    def clean_river_name(self, river):
        """Standardize river names"""
        if pd.isna(river):
            return None

        # Common corrections
        corrections = {
            'R. ': 'River ',
            'Riv. ': 'River ',
            'Rvr ': 'River '
        }

        cleaned = str(river).strip()
        for old, new in corrections.items():
            cleaned = cleaned.replace(old, new)

        # Handle multiple rivers in one field
        if ',' in cleaned:
            rivers = [r.strip() for r in cleaned.split(',')]
            return rivers[0]  # Take primary river

        return cleaned

    def is_greater_manchester_relevant(self, row):
        """Check if an area is relevant to Greater Manchester"""
        if pd.isna(row['label']) and pd.isna(row['description']):
            return False

        # Create full text to check
        text_to_check = ' '.join([
            str(row.get('label', '')),
            str(row.get('description', '')),
            str(row.get('river', '')),
            str(row.get('county', ''))
        ]).lower()

        # First check if the river is one we're monitoring
        river = str(row.get('river', '')).strip()
        if river and river not in self.all_monitored_rivers:
            return False

        # Check for primary area matches
        has_primary_area = any(
            f" {area.lower()} " in f" {text_to_check} "
            for area in self.gm_areas['primary']
        )

        # Check for secondary area matches
        has_secondary_area = any(
            f" {area.lower()} " in f" {text_to_check} "
            for area in self.gm_areas['secondary']
        )

        # Explicit Greater Manchester mention
        has_gm_mention = "greater manchester" in text_to_check

        # Must have either:
```

```
        # 1. A monitored river AND (primary area OR GM mention)
        # 2. A monitored river AND multiple secondary area mentions
        return (river in self.all_monitored_rivers) and (
            has_primary_area or
            has_gm_mention or
            (has_secondary_area and text_to_check.count(' manchester ') >= 2)
        )

    def load_and_clean_json(self, filename):
        """Load and clean JSON flood data files"""
        try:
            with open(os.path.join(self.flood_path, filename), 'r') as f:
                data = json.load(f)

            # Convert JSON to DataFrame based on structure
            if isinstance(data, list):
                df = pd.json_normalize(data)
            else:
                df = pd.json_normalize([data])

            # Rename columns to match standard format
            column_mapping = {
                'floodArea.label': 'label',
                'floodArea.description': 'description',
                'floodArea.river': 'river',
                'floodArea.county': 'county',
                'floodArea.quickDial': 'quick_dial'
            }
            df = df.rename(columns=column_mapping)

            return df

        except Exception as e:
            print(f"Error processing {filename}: {e}")
            return None

    def determine_risk_level(self, row):
        """Determine flood risk level from description and other factors"""
        if pd.isna(row['description']):
            return 'Unknown'

        description = str(row['description']).lower()
        label = str(row['label']).lower()

        # Check for explicit risk indicators
        high_risk_phrases = [
            'severe flood', 'immediate action', 'flood warning',
            'danger to life', 'major flooding', 'high risk',
            'property flooding', 'flooding is expected'
        ]

        medium_risk_phrases = [
            'flood alert', 'rising levels', 'be prepared',
            'flooding possible', 'historical flooding',
            'river levels high', 'surface water'
        ]

        low_risk_phrases = [
            'monitoring', 'normal conditions',
            'routine monitoring', 'no immediate concern'
```

```
        ]

        # Check both description and label
        text_to_check = f"{description} {label}"

        # Check for high risk indicators
        if any(phrase in text_to_check for phrase in high_risk_phrases):
            return 'High'

        # Check for key infrastructure or vulnerable areas
        if any(term in text_to_check for term in ['hospital', 'school', 'care hc
            return 'High'

        # Check for medium risk indicators
        if any(phrase in text_to_check for phrase in medium_risk_phrases):
            return 'Medium'

        # Properties mentioned but no immediate risk
        if 'properties' in text_to_check and not any(phrase in text_to_check for
            return 'Medium'

        # Low risk if explicitly mentioned
        if any(phrase in text_to_check for phrase in low_risk_phrases):
            return 'Low'

        # Default to Medium if we're tracking it but no clear indicators
        return 'Medium'

    def assign_monitoring_station(self, row):
        """Assign relevant monitoring station(s) based on location and river"""
        try:
            text = ' '.join([
                str(row.get('label', '')),
                str(row.get('description', '')),
                str(row.get('river', ''))
            ]).lower()

            def check_catchment_match(catchment, text):
                """Helper to check how well a catchment matches the text"""
                score = 0
                # Check primary areas
                if any(area.lower() in text for area in catchment['primary_areas
                    score += 3
                # Check rivers
                if any(river.lower() in text for river in catchment['rivers']):
                    score += 2
                # Check boundaries
                if any(bound.lower() in text for bound in catchment['boundaries'
                    score += 1
                return score

            # Calculate match scores for each station
            station_scores = {
                station_id: check_catchment_match(catchment, text)
                for station_id, catchment in self.station_catchments.items()
            }

            # Special cases
            if 'crumpsall' in text or 'irk' in text:
                station_scores['690510'] += 2  # Boost Manchester Racecourse for
```

```
            if 'fletcher moss' in text or 'withington' in text or 'didsbury' in 
                station_scores['690510'] += 2  # Boost Manchester Racecourse for
            if 'upper irwell' in text:
                station_scores['690160'] += 2  # Boost Bury Ground for upper Irw

            # Get best matching station if any score > 0
            best_station = max(station_scores.items(), key=lambda x: x[1])
            return best_station[0] if best_station[1] > 0 else None

        except Exception as e:
            print(f"Error assigning monitoring station: {e}")
            return None

    def clean_flood_data(self):
        """Main cleaning function for all flood data"""
        try:
            print("\nStarting flood data cleaning process...")
            print(f"Input directory: {self.flood_path}")
            print(f"Output directory: {self.output_path}\n")

            # Initialize empty list for all flood data
            all_data = []

            # Process each file in the flood data directory
            for filename in os.listdir(self.flood_path):
                print(f"Processing file: {filename}")
                if filename.endswith('.json'):
                    df = self.load_and_clean_json(filename)
                elif filename.endswith('.csv'):
                    try:
                        df = pd.read_csv(os.path.join(self.flood_path, filename)
                    except Exception as e:
                        print(f"Error reading {filename}: {e}")
                        continue
                else:
                    continue

                if df is not None:
                    # Ensure required columns exist
                    required_columns = ['label', 'description', 'river']
                    for col in required_columns:
                        if col not in df.columns:
                            df[col] = None
                    all_data.append(df)

            # Combine all data
            if not all_data:
                raise Exception("No data could be loaded")

            combined_df = pd.concat(all_data, ignore_index=True)
            print(f"\nTotal records loaded: {len(combined_df)}")

            # Clean and standardize
            print("\nCleaning river names...")
            combined_df['river'] = combined_df['river'].apply(self.clean_river_n

            # Filter for Greater Manchester relevance
            print("Filtering for Greater Manchester relevance...")
            gm_df = combined_df[combined_df.apply(self.is_greater_manchester_rel
            print(f"Found {len(gm_df)} relevant Greater Manchester records")
```

```
            if len(gm_df) == 0:
                raise Exception("No Greater Manchester relevant data found after

            # Process GM relevant data
            print("\nAssigning risk levels...")
            gm_df['risk_level'] = gm_df.apply(self.determine_risk_level, axis=1)

            print("Assigning monitoring stations...")
            gm_df['monitoring_stations'] = gm_df.apply(self.assign_monitoring_st

            # Add metadata
            print("Adding metadata...")
            gm_df['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            gm_df['data_quality'] = gm_df.apply(
                lambda x: 'High' if pd.notna(x['river']) and pd.notna(x['monitor
                else 'Medium' if pd.notna(x['river']) or pd.notna(x['monitoring_
                else 'Low',
                axis=1
            )

            # Create output directory if it doesn't exist
            os.makedirs(self.output_path, exist_ok=True)

            # Save cleaned data
            print("\nSaving cleaned data...")
            csv_path = os.path.join(self.output_path, 'cleaned_flood_areas.csv')
            gm_df.to_csv(csv_path, index=False)
            print(f"Saved CSV file to: {csv_path}")

            # Generate and save summary statistics
            summary = {
                'total_areas': len(gm_df),
                'rivers_covered': gm_df['river'].nunique(),
                'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
                'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stat
                'data_quality_distribution': gm_df['data_quality'].value_counts(
            }

            # Save summary
            summary_path = os.path.join(self.output_path, 'flood_data_summary.js
            with open(summary_path, 'w') as f:
                json.dump(summary, f, indent=2)
            print(f"Saved summary file to: {summary_path}")

            return gm_df, summary

        except Exception as e:
            print(f"\nError during cleaning process: {e}")
            return None, None

            # Save cleaned data
            print("Saving cleaned data...")
            output_file = os.path.join(self.output_path, 'cleaned_flood_areas.cs
            gm_df.to_csv(output_file, index=False)

            # Generate summary statistics
            summary = {
                'total_areas': len(gm_df),
                'rivers_covered': gm_df['river'].nunique(),
```

```
                'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
                'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stat
                'data_quality_distribution': gm_df['data_quality'].value_counts(
            }

            # Save summary
            summary_file = os.path.join(self.output_path, 'flood_data_summary.js
            with open(summary_file, 'w') as f:
                json.dump(summary, f, indent=2)

            return gm_df, summary

        except Exception as e:
            print(f"Error during cleaning process: {e}")
            return None, None

        # Filter for Greater Manchester relevance
        gm_df = combined_df[combined_df.apply(self.is_greater_manchester_relevan

        # Add metadata
        gm_df['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        gm_df['data_quality'] = gm_df.apply(
            lambda x: 'High' if pd.notna(x['river']) and pd.notna(x['monitoring_
            else 'Medium' if pd.notna(x['river']) or pd.notna(x['monitoring_stat
            else 'Low',
            axis=1
        )

        # Save cleaned data
        output_file = os.path.join(self.output_path, 'cleaned_flood_areas.csv')
        gm_df.to_csv(output_file, index=False)

        # Generate summary statistics
        summary = {
            'total_areas': len(gm_df),
            'rivers_covered': gm_df['river'].nunique(),
            'high_risk_areas': len(gm_df[gm_df['risk_level'] == 'High']),
            'areas_with_stations': len(gm_df[pd.notna(gm_df['monitoring_stations
            'data_quality_distribution': gm_df['data_quality'].value_counts().tc
        }

        # Save summary
        summary_file = os.path.join(self.output_path, 'flood_data_summary.json')
        with open(summary_file, 'w') as f:
            json.dump(summary, f, indent=2)

        return gm_df, summary

def main():
    """Main function to run the cleaning process"""
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    cleaner = FloodDataCleaner(project_path)

    try:
        df, summary = cleaner.clean_flood_data()
        print("\nCleaning completed successfully!")
        print("\nSummary of cleaned data:")
        print(json.dumps(summary, indent=2))

        print("\nSample of cleaned data:")
```

```
        print(df[['label', 'river', 'risk_level', 'monitoring_stations', 'data_q

    except Exception as e:
        print(f"Error during cleaning process: {e}")

def main():
    """Main function to run the cleaning process"""
    project_path = r"C:\Users\Administrator\NEWPROJECT"
    cleaner = FloodDataCleaner(project_path)

    try:
        df, summary = cleaner.clean_flood_data()
        print("\nCleaning completed successfully!")
        print("\nSummary of cleaned data:")
        print(json.dumps(summary, indent=2))

        print("\nSample of cleaned data:")
        print(df[['label', 'river', 'risk_level', 'monitoring_stations', 'data_q

        return df, summary

    except Exception as e:
        print(f"Error during cleaning process: {e}")
        return None, None

if __name__ == "__main__":
    df, summary = main()
    main()
```

```
Starting flood data cleaning process...
Input directory: C:\Users\Administrator\NEWPROJECT\flood_data
Output directory: C:\Users\Administrator\NEWPROJECT\cleaned_data

Processing file: flood_alerts_raw.json
Processing file: flood_areas.csv
Processing file: flood_areas_raw.json
Processing file: manchester_floods

Total records loaded: 502

Cleaning river names...
Filtering for Greater Manchester relevance...
Found 5 relevant Greater Manchester records

Assigning risk levels...
Assigning monitoring stations...
Adding metadata...

Saving cleaned data...
Saved CSV file to: C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_flood_a
reas.csv
Saved summary file to: C:\Users\Administrator\NEWPROJECT\cleaned_data\flood_data_
summary.json

Cleaning completed successfully!

Summary of cleaned data:
{
  "total_areas": 5,
  "rivers_covered": 3,
  "high_risk_areas": 1,
  "areas_with_stations": 5,
  "data_quality_distribution": {
    "High": 5
  }
}

Sample of cleaned data:
                                                 label         river  \
29   River Mersey at Fletcher Moss and Withington G...  River Mersey
52                     River Irk at Crumpsall Hospital     River Irk
53   Lower River Irwell catchment including areas i...  River Irwell
89   Upper River Irwell catchment with Oldham, Bolt...  River Irwell
122  River Mersey Uplands catchment including Hyde,...  River Mersey

    risk_level monitoring_stations data_quality
29      Medium              690510         High
52        High              690510         High
53      Medium              690510         High
89      Medium              690160         High
122     Medium              690510         High

Starting flood data cleaning process...
Input directory: C:\Users\Administrator\NEWPROJECT\flood_data
Output directory: C:\Users\Administrator\NEWPROJECT\cleaned_data

Processing file: flood_alerts_raw.json
Processing file: flood_areas.csv
Processing file: flood_areas_raw.json
```

```
Processing file: manchester_floods

Total records loaded: 502

Cleaning river names...
Filtering for Greater Manchester relevance...
Found 5 relevant Greater Manchester records

Assigning risk levels...
Assigning monitoring stations...
Adding metadata...

Saving cleaned data...
Saved CSV file to: C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_flood_a
reas.csv
Saved summary file to: C:\Users\Administrator\NEWPROJECT\cleaned_data\flood_data_
summary.json

Cleaning completed successfully!

Summary of cleaned data:
{
  "total_areas": 5,
  "rivers_covered": 3,
  "high_risk_areas": 1,
  "areas_with_stations": 5,
  "data_quality_distribution": {
    "High": 5
  }
}

Sample of cleaned data:
                                                 label         river  \
29   River Mersey at Fletcher Moss and Withington G...  River Mersey
52                       River Irk at Crumpsall Hospital     River Irk
53   Lower River Irwell catchment including areas i...  River Irwell
89   Upper River Irwell catchment with Oldham, Bolt...  River Irwell
122  River Mersey Uplands catchment including Hyde,...  River Mersey

    risk_level monitoring_stations data_quality
29      Medium              690510         High
52        High              690510         High
53      Medium              690510         High
89      Medium              690160         High
122     Medium              690510         High
```

Data Integration and Processing

In [144…

```python
import os

def list_historical_files():
    directory = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\hist

    print("Checking directory:", directory)
    print("\nFiles in directory:")

    if os.path.exists(directory):
        files = os.listdir(directory)
        for file in files:
```

```
            print(f"- {file}")
    else:
        print("Directory does not exist!")

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

```
Checking directory: C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\his
torical

Files in directory:
- bury_daily_flow.csv
- bury_daily_rainfall.csv
- bury_peak_flow.csv
- manchester_peak_flow.csv
- rochdale_daily_flow.csv
- rochdale_daily_rainfall.csv
- rochdale_peak_flow.csv
```

In [146…

```
import pandas as pd
import os

def inspect_csv_structure(directory):
    """
    Inspect the structure of all CSV files in the directory
    """
    for filename in os.listdir(directory):
        if filename.endswith('.csv'):
            file_path = os.path.join(directory, filename)
            print(f"\nFile: {filename}")
            print("-" * 50)

            # Read first few rows of the CSV
            df = pd.read_csv(file_path)

            # Display column names
            print("Columns:")
            for col in df.columns:
                print(f"- {col}")

            # Display first row as sample
            print("\nFirst row sample:")
            print(df.iloc[0])
            print("\n")

if __name__ == "__main__":
    directory = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\hist
    inspect_csv_structure(directory)
```

```
File: bury_daily_flow.csv
--------------------------------------------------
Columns:
- Date
- Flow
- Extra

First row sample:
Date     1995-11-22
Flow            0.9
Extra           NaN
Name: 0, dtype: object



File: bury_daily_rainfall.csv
--------------------------------------------------
Columns:
- Date
- Rainfall
- Extra

First row sample:
Date        1961-01-01
Rainfall           9.4
Extra             1000
Name: 0, dtype: object



File: bury_peak_flow.csv
--------------------------------------------------
Columns:
- Water Year
- Date
- Time
- Stage (m)
- Flow (m3/s)
- Rating
- Datetime

First row sample:
Water Year               1972-1973
Date                    1973-01-12
Time                      00:00:00
Stage (m)                      1.3
Flow (m3/s)                   78.1
Rating                         NaN
Datetime       1973-01-12 00:00:00
Name: 0, dtype: object



File: manchester_peak_flow.csv
--------------------------------------------------
Columns:
- Water Year
- Date
- Time
- Stage (m)
```

```
- Flow (m3/s)
- Rating
- Datetime

First row sample:
Water Year               1941-1942
Date                    1941-10-24
Time                      00:00:00
Stage (m)                      3.5
Flow (m3/s)                  269.0
Rating                     Extrap.
Datetime       1941-10-24 00:00:00
Name: 0, dtype: object



File: rochdale_daily_flow.csv
--------------------------------------------------
Columns:
- Date
- Flow
- Extra

First row sample:
Date     1993-02-26
Flow            1.3
Extra           NaN
Name: 0, dtype: object



File: rochdale_daily_rainfall.csv
--------------------------------------------------
Columns:
- Date
- Rainfall
- Extra

First row sample:
Date        2016-01-01
Rainfall           0.8
Extra             2000
Name: 0, dtype: object



File: rochdale_peak_flow.csv
--------------------------------------------------
Columns:
- Water Year
- Date
- Time
- Stage (m)
- Flow (m3/s)
- Rating
- Datetime

First row sample:
Water Year               1992-1993
Date                    1993-09-13
```

```
Time                      11:30:00
Stage (m)                      0.9
Flow (m3/s)                   21.1
Rating                    In Range
Datetime       1993-09-13 11:30:00
Name: 0, dtype: object
```

Purpose: Historical Data Preprocessing

Processes historical river data (flow, rainfall, peak flow). Standardizes, merges, analyzes, and exports cleaned data.

In [151…

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

class HistoricalDataProcessor:
    def __init__(self, data_directory):
        """
        Initialize the processor with the directory containing historical data f

        Args:
            data_directory (str): Path to the directory containing CSV files
        """
        self.data_directory = data_directory
        self.processed_data = {}

    def load_csv_files(self):
        """
        Load all CSV files from the specified directory.

        Returns:
            dict: Dictionary of DataFrames with filenames as keys
        """
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.

        for filename in csv_files:
            file_path = os.path.join(self.data_directory, filename)
            df = pd.read_csv(file_path)

            # Standardize column names and types
            self._standardize_dataframe(df, filename)

            self.processed_data[filename] = df

        return self.processed_data

    def _standardize_dataframe(self, df, filename):
        """
        Standardize DataFrame columns and types based on filename.

        Args:
            df (pd.DataFrame): Input DataFrame
            filename (str): Name of the source file
        """
        # Convert Date columns to datetime
```

```
        if 'Date' in df.columns:
            df['Date'] = pd.to_datetime(df['Date'])

        # Handle specific file types
        if 'daily_flow' in filename:
            df.rename(columns={'Flow': 'daily_flow'}, inplace=True)
            df['daily_flow'] = pd.to_numeric(df['daily_flow'], errors='coerce')

        elif 'daily_rainfall' in filename:
            df.rename(columns={'Rainfall': 'daily_rainfall'}, inplace=True)
            df['daily_rainfall'] = pd.to_numeric(df['daily_rainfall'], errors='c

        elif 'peak_flow' in filename:
            # Ensure consistent column names and types for peak flow data
            df.rename(columns={
                'Stage (m)': 'stage_meters',
                'Flow (m3/s)': 'peak_flow_cubic_meters'
            }, inplace=True)

            # Convert datetime columns
            if 'Datetime' in df.columns:
                df['Datetime'] = pd.to_datetime(df['Datetime'])

            # Convert numeric columns
            numeric_cols = ['stage_meters', 'peak_flow_cubic_meters']
            df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coe

    def combine_location_data(self):
        """
        Combine data for each location (Bury, Rochdale, Manchester).

        Returns:
            dict: Combined DataFrames for each location
        """
        locations = {
            'bury': ['bury_daily_flow.csv', 'bury_daily_rainfall.csv', 'bury_pea
            'rochdale': ['rochdale_daily_flow.csv', 'rochdale_daily_rainfall.csv
            'manchester': ['manchester_peak_flow.csv']
        }

        combined_data = {}

        for location, files in locations.items():
            location_dfs = [
                self.processed_data[file]
                for file in files
                if file in self.processed_data
            ]

            if location in ['bury', 'rochdale']:
                # For Bury and Rochdale, merge daily flow and rainfall
                if len(location_dfs) >= 2:
                    # Merge daily flow and daily rainfall on Date
                    merged_daily = pd.merge(
                        location_dfs[0], location_dfs[1],
                        on='Date', how='outer'
                    )

                    # If peak flow exists, merge it too
                    if len(location_dfs) > 2:
```

```
                    merged_daily = pd.merge(
                        merged_daily, location_dfs[2],
                        left_on='Date', right_on='Date',
                        how='outer'
                    )

                combined_data[location] = merged_daily

        elif location == 'manchester':
            # For Manchester, just use the peak flow data
            combined_data[location] = location_dfs[0]

    return combined_data

def calculate_statistical_baselines(self):
    """
    Calculate statistical baselines for each location.

    Returns:
        dict: Statistical summaries for each location
    """
    baselines = {}

    for location, df in self.combine_location_data().items():
        location_baseline = {
            'location': location,
            'total_records': len(df)
        }

        # Calculate baseline metrics for available columns
        numeric_columns = [
            col for col in df.columns
            if df[col].dtype in ['float64', 'int64'] and not col.startswith(
        ]

        for col in numeric_columns:
            location_baseline[f'{col}_mean'] = df[col].mean()
            location_baseline[f'{col}_median'] = df[col].median()
            location_baseline[f'{col}_std'] = df[col].std()
            location_baseline[f'{col}_min'] = df[col].min()
            location_baseline[f'{col}_max'] = df[col].max()

        # Temporal analysis
        if 'Date' in df.columns:
            location_baseline.update({
                'date_range_start': df['Date'].min(),
                'date_range_end': df['Date'].max(),
                'total_years': (df['Date'].max().year - df['Date'].min().yea
            })

        baselines[location] = location_baseline

    return baselines

def detect_seasonal_patterns(self):
    """
    Detect and analyze seasonal patterns in the data.

    Returns:
        dict: Seasonal pattern analysis for each location
```

```
        """
        seasonal_analysis = {}

        for location, df in self.combine_location_data().items():
            if 'Date' not in df.columns:
                continue

            # Ensure Date is datetime
            df['Date'] = pd.to_datetime(df['Date'])

            # Extract seasonal components
            seasonal_metrics = {}

            # Numeric columns to analyze
            numeric_columns = [
                col for col in df.columns
                if df[col].dtype in ['float64', 'int64'] and not col.startswith(
            ]

            for col in numeric_columns:
                # Group by month and calculate statistics
                monthly_stats = df.groupby(df['Date'].dt.month)[col].agg([
                    'mean', 'median', 'std', 'min', 'max'
                ]).rename(columns={
                    'mean': f'{col}_monthly_mean',
                    'median': f'{col}_monthly_median',
                    'std': f'{col}_monthly_std',
                    'min': f'{col}_monthly_min',
                    'max': f'{col}_monthly_max'
                })

                seasonal_metrics.update(monthly_stats.to_dict())

            # Seasonal variation calculation
            seasonal_analysis[location] = {
                'seasonal_metrics': seasonal_metrics
            }

        return seasonal_analysis

    def export_processed_data(self, output_directory):
        """
        Export processed data to CSV files.

        Args:
            output_directory (str): Directory to save processed data
        """
        os.makedirs(output_directory, exist_ok=True)

        # Export combined location data
        combined_data = self.combine_location_data()
        for location, df in combined_data.items():
            output_path = os.path.join(output_directory, f'{location}_combined_d
            df.to_csv(output_path, index=False)

        # Export statistical baselines
        baselines = self.calculate_statistical_baselines()
        baselines_df = pd.DataFrame.from_dict(baselines, orient='index')
        baselines_df.to_csv(os.path.join(output_directory, 'location_baselines.c
```

```
        # Export seasonal patterns
        seasonal_analysis = self.detect_seasonal_patterns()
        seasonal_df = pd.DataFrame.from_dict(seasonal_analysis, orient='index')
        seasonal_df.to_csv(os.path.join(output_directory, 'seasonal_analysis.csv

def main():
    # Example usage
    data_directory = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data
    output_directory = r'C:\Users\Administrator\NEWPROJECT\processed_data'

    # Create output directory if it doesn't exist
    os.makedirs(output_directory, exist_ok=True)

    # Initialize processor
    processor = HistoricalDataProcessor(data_directory)

    try:
        # Load and process data
        processor.load_csv_files()

        # Calculate baseline statistics
        baselines = processor.calculate_statistical_baselines()
        print("Location Baselines:")
        for location, baseline in baselines.items():
            print(f"\n{location.capitalize()} Baseline:")
            for key, value in baseline.items():
                print(f"{key}: {value}")

        # Detect seasonal patterns
        seasonal_patterns = processor.detect_seasonal_patterns()

        # Export processed data
        processor.export_processed_data(output_directory)

        print(f"\nProcessed data exported to {output_directory}")

    except Exception as e:
        print(f"An error occurred: {e}")
        import traceback
        traceback.print_exc()

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

```
Location Baselines:

Bury Baseline:
location: bury
total_records: 22918
daily_flow_mean: 3.8503255439161967
daily_flow_median: 2.064
daily_flow_std: 5.395384747258743
daily_flow_min: 0.406
daily_flow_max: 117.0
daily_rainfall_mean: 3.7754983428598874
daily_rainfall_median: 0.9
daily_rainfall_std: 6.209935248255218
daily_rainfall_min: 0.0
daily_rainfall_max: 79.5
stage_meters_mean: 1.4495490196078429
stage_meters_median: 1.447
stage_meters_std: 0.20892374816908588
stage_meters_min: 1.074
stage_meters_max: 2.178
peak_flow_cubic_meters_mean: 115.93141176470589
peak_flow_cubic_meters_median: 112.88
peak_flow_cubic_meters_std: 43.59888067974978
peak_flow_cubic_meters_min: 51.511
peak_flow_cubic_meters_max: 283.649
date_range_start: 1961-01-01 00:00:00
date_range_end: 2023-09-30 00:00:00
total_years: 63

Rochdale Baseline:
location: rochdale
total_records: 11118
daily_flow_mean: 2.795590034178809
daily_flow_median: 1.4889999999999999
daily_flow_std: 3.546723998338466
daily_flow_min: 0.178
daily_flow_max: 50.41
daily_rainfall_mean: 3.7835841313269496
daily_rainfall_median: 0.9
daily_rainfall_std: 5.848198763742652
daily_rainfall_min: 0.0
daily_rainfall_max: 36.6
stage_meters_mean: 1.4297741935483872
stage_meters_median: 1.413
stage_meters_std: 0.2851276567524822
stage_meters_min: 0.808
stage_meters_max: 2.222
peak_flow_cubic_meters_mean: 46.37712903225806
peak_flow_cubic_meters_median: 44.654
peak_flow_cubic_meters_std: 15.045484950070426
peak_flow_cubic_meters_min: 17.976
peak_flow_cubic_meters_max: 92.846
date_range_start: 1993-02-26 00:00:00
date_range_end: 2023-09-30 00:00:00
total_years: 31

Manchester Baseline:
location: manchester
total_records: 82
stage_meters_mean: 3.513146341463415
```

```
stage_meters_median: 3.5
stage_meters_std: 0.5900609745004537
stage_meters_min: 2.46
stage_meters_max: 5.668
peak_flow_cubic_meters_mean: 279.4348414634146
peak_flow_cubic_meters_median: 273.5
peak_flow_cubic_meters_std: 87.35713783912391
peak_flow_cubic_meters_min: 135.0
peak_flow_cubic_meters_max: 560.0
date_range_start: 1941-10-24 00:00:00
date_range_end: 2023-01-10 00:00:00
total_years: 83

Processed data exported to C:\Users\Administrator\NEWPROJECT\processed_data
```

In [153…

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

class HistoricalDataProcessor:
    def __init__(self, data_directory):
        """
        Initialize the processor with the directory containing historical data f

        Args:
            data_directory (str): Path to the directory containing CSV files
        """
        self.data_directory = data_directory
        self.processed_data = {}

    def load_csv_files(self):
        """
        Load all CSV files from the specified directory.

        Returns:
            dict: Dictionary of DataFrames with filenames as keys
        """
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.

        for filename in csv_files:
            file_path = os.path.join(self.data_directory, filename)
            df = pd.read_csv(file_path)

            # Standardize column names and types
            self._standardize_dataframe(df, filename)

            self.processed_data[filename] = df

        return self.processed_data

    def _standardize_dataframe(self, df, filename):
        """
        Standardize DataFrame columns and types based on filename.

        Args:
            df (pd.DataFrame): Input DataFrame
            filename (str): Name of the source file
        """
        # Convert Date columns to datetime
```

```
        if 'Date' in df.columns:
            df['Date'] = pd.to_datetime(df['Date'])

        # Handle specific file types
        if 'daily_flow' in filename:
            df.rename(columns={'Flow': 'daily_flow'}, inplace=True)
            df['daily_flow'] = pd.to_numeric(df['daily_flow'], errors='coerce')

        elif 'daily_rainfall' in filename:
            df.rename(columns={'Rainfall': 'daily_rainfall'}, inplace=True)
            df['daily_rainfall'] = pd.to_numeric(df['daily_rainfall'], errors='c

        elif 'peak_flow' in filename:
            # Ensure consistent column names and types for peak flow data
            df.rename(columns={
                'Stage (m)': 'stage_meters',
                'Flow (m3/s)': 'peak_flow_cubic_meters'
            }, inplace=True)

            # Convert datetime columns
            if 'Datetime' in df.columns:
                df['Datetime'] = pd.to_datetime(df['Datetime'])

            # Convert numeric columns
            numeric_cols = ['stage_meters', 'peak_flow_cubic_meters']
            df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coe

    def combine_location_data(self):
        """
        Combine data for each location (Bury, Rochdale, Manchester).

        Returns:
            dict: Combined DataFrames for each location
        """
        locations = {
            'bury': ['bury_daily_flow.csv', 'bury_daily_rainfall.csv', 'bury_pea
            'rochdale': ['rochdale_daily_flow.csv', 'rochdale_daily_rainfall.csv
            'manchester': ['manchester_peak_flow.csv']
        }

        combined_data = {}

        for location, files in locations.items():
            location_dfs = [
                self.processed_data[file]
                for file in files
                if file in self.processed_data
            ]

            if location in ['bury', 'rochdale']:
                # For Bury and Rochdale, merge daily flow and rainfall
                if len(location_dfs) == 2:
                    # Merge daily flow and daily rainfall on Date
                    merged_daily = pd.merge(
                        location_dfs[0], location_dfs[1],
                        on='Date', how='outer'
                    )

                    # If peak flow exists, merge it too
                    if len(location_dfs) > 2:
```

```
                    merged_daily = pd.merge(
                        merged_daily, location_dfs[2],
                        left_on='Date', right_on='Date',
                        how='outer'
                    )

                combined_data[location] = merged_daily

        elif location == 'manchester':
            # For Manchester, just use the peak flow data
            combined_data[location] = location_dfs[0]

    return combined_data

def calculate_statistical_baselines(self):
    """
    Calculate statistical baselines for each location.

    Returns:
        dict: Statistical summaries for each location
    """
    baselines = {}

    for location, df in self.combine_location_data().items():
        location_baseline = {
            'location': location,
            'total_records': len(df)
        }

        # Calculate baseline metrics for available columns
        numeric_columns = [
            col for col in df.columns
            if df[col].dtype in ['float64', 'int64'] and not col.startswith(
        ]

        for col in numeric_columns:
            location_baseline[f'{col}_mean'] = df[col].mean()
            location_baseline[f'{col}_median'] = df[col].median()
            location_baseline[f'{col}_std'] = df[col].std()
            location_baseline[f'{col}_min'] = df[col].min()
            location_baseline[f'{col}_max'] = df[col].max()

        # Temporal analysis
        if 'Date' in df.columns:
            location_baseline.update({
                'date_range_start': df['Date'].min(),
                'date_range_end': df['Date'].max(),
                'total_years': (df['Date'].max().year - df['Date'].min().yea
            })

        baselines[location] = location_baseline

    return baselines

def detect_seasonal_patterns(self):
    """
    Detect and analyze seasonal patterns in the data.

    Returns:
        dict: Seasonal pattern analysis for each location
```

```
        """
        seasonal_analysis = {}

        for location, df in self.combine_location_data().items():
            if 'Date' not in df.columns:
                continue

            # Ensure Date is datetime
            df['Date'] = pd.to_datetime(df['Date'])

            # Extract seasonal components
            seasonal_metrics = {}

            # Numeric columns to analyze
            numeric_columns = [
                col for col in df.columns
                if df[col].dtype in ['float64', 'int64'] and not col.startswith(
            ]

            for col in numeric_columns:
                # Group by month and calculate statistics
                monthly_stats = df.groupby(df['Date'].dt.month)[col].agg([
                    'mean', 'median', 'std', 'min', 'max'
                ]).rename(columns={
                    'mean': f'{col}_monthly_mean',
                    'median': f'{col}_monthly_median',
                    'std': f'{col}_monthly_std',
                    'min': f'{col}_monthly_min',
                    'max': f'{col}_monthly_max'
                })

                seasonal_metrics.update(monthly_stats.to_dict())

            # Seasonal variation calculation
            seasonal_analysis[location] = {
                'seasonal_metrics': seasonal_metrics
            }

        return seasonal_analysis

    def export_processed_data(self, output_directory):
        """
        Export processed data to CSV files.

        Args:
            output_directory (str): Directory to save processed data
        """
        os.makedirs(output_directory, exist_ok=True)

        # Export combined location data
        combined_data = self.combine_location_data()
        for location, df in combined_data.items():
            output_path = os.path.join(output_directory, f'{location}_combined_d
            df.to_csv(output_path, index=False)

        # Export statistical baselines
        baselines = self.calculate_statistical_baselines()
        baselines_df = pd.DataFrame.from_dict(baselines, orient='index')
        baselines_df.to_csv(os.path.join(output_directory, 'location_baselines.c
```

```
        # Export seasonal patterns
        seasonal_analysis = self.detect_seasonal_patterns()
        seasonal_df = pd.DataFrame.from_dict(seasonal_analysis, orient='index')
        seasonal_df.to_csv(os.path.join(output_directory, 'seasonal_analysis.csv

def main():
    # Example usage
    data_directory = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data
    output_directory = r'C:\Users\Administrator\NEWPROJECT\processed_data'

    # Initialize processor
    processor = HistoricalDataProcessor(data_directory)

    # Load and process data
    processor.load_csv_files()

    # Calculate baseline statistics
    baselines = processor.calculate_statistical_baselines()
    print("Location Baselines:")
    for location, baseline in baselines.items():
        print(f"\n{location.capitalize()} Baseline:")
        for key, value in baseline.items():
            print(f"{key}: {value}")

    # Detect seasonal patterns
    seasonal_patterns = processor.detect_seasonal_patterns()

    # Export processed data
    processor.export_processed_data(output_directory)

    print(f"\nProcessed data exported to {output_directory}")

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

```
Location Baselines:

Manchester Baseline:
location: manchester
total_records: 82
stage_meters_mean: 3.513146341463415
stage_meters_median: 3.5
stage_meters_std: 0.5900609745004537
stage_meters_min: 2.46
stage_meters_max: 5.668
peak_flow_cubic_meters_mean: 279.4348414634146
peak_flow_cubic_meters_median: 273.5
peak_flow_cubic_meters_std: 87.35713783912391
peak_flow_cubic_meters_min: 135.0
peak_flow_cubic_meters_max: 560.0
date_range_start: 1941-10-24 00:00:00
date_range_end: 2023-01-10 00:00:00
total_years: 83

Processed data exported to C:\Users\Administrator\NEWPROJECT\processed_data
```

In [155…

```
import pandas as pd
import numpy as np
import os
from typing import Dict, List, Any
```

```
class DataIntegrationProcessor:
    def __init__(self, data_directories: Dict[str, str]):
        """
        Initialize the data integration processor.

        Args:
            data_directories (dict): Directories for different data sources
        """
        self.data_directories = data_directories
        self.processed_data = {}

    def load_csv_files(self, directory: str) -> Dict[str, pd.DataFrame]:
        """
        Load all CSV files from a specified directory.

        Args:
            directory (str): Path to the directory containing CSV files

        Returns:
            dict: Dictionary of DataFrames with filenames as keys
        """
        csv_files = [f for f in os.listdir(directory) if f.endswith('.csv')]
        loaded_dataframes = {}

        for filename in csv_files:
            file_path = os.path.join(directory, filename)
            df = pd.read_csv(file_path)
            loaded_dataframes[filename] = df

        return loaded_dataframes

    def standardize_datetime(self, df: pd.DataFrame, date_column: str = 'Date')
        """
        Standardize datetime columns.

        Args:
            df (pd.DataFrame): Input DataFrame
            date_column (str): Name of the date column

        Returns:
            pd.DataFrame: DataFrame with standardized datetime
        """
        if date_column in df.columns:
            df[date_column] = pd.to_datetime(df[date_column])
        return df

    def handle_missing_values(self, df: pd.DataFrame, method: str = 'interpolate
        """
        Handle missing values in the DataFrame.

        Args:
            df (pd.DataFrame): Input DataFrame
            method (str): Method to handle missing values

        Returns:
            pd.DataFrame: DataFrame with handled missing values
        """
        # Identify numeric columns
        numeric_columns = df.select_dtypes(include=[np.number]).columns
```

```
        if method == 'interpolate':
            # Interpolate missing values for numeric columns
            df[numeric_columns] = df[numeric_columns].interpolate(method='linear
        elif method == 'forward_fill':
            # Forward fill missing values
            df[numeric_columns] = df[numeric_columns].fillna(method='ffill')
        elif method == 'backward_fill':
            # Backward fill missing values
            df[numeric_columns] = df[numeric_columns].fillna(method='bfill')

        return df

    def normalize_column_names(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Normalize column names to a consistent format.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: DataFrame with normalized column names
        """
        df.columns = df.columns.str.lower().str.strip().str.replace(' ', '_')
        return df

    def merge_location_data(self, location_files: List[str]) -> pd.DataFrame:
        """
        Merge data files for a specific location.

        Args:
            location_files (list): List of file paths to merge

        Returns:
            pd.DataFrame: Merged DataFrame for the location
        """
        merged_df = None

        for file_path in location_files:
            df = pd.read_csv(file_path)
            self.standardize_datetime(df)
            self.normalize_column_names(df)

            if merged_df is None:
                merged_df = df
            else:
                # Merge on date column
                merged_df = pd.merge(
                    merged_df, df,
                    on='date',
                    how='outer'
                )

        return merged_df

    def process_historical_nrfa_data(self):
        """
        Process Historical NRFA Data.
        """
        nrfa_directory = self.data_directories.get('historical_nrfa', '')
```

```
        nrfa_files = self.load_csv_files(nrfa_directory)

        processed_nrfa_data = {}
        locations = ['bury', 'rochdale', 'manchester']

        for location in locations:
            location_files = [
                file for file in nrfa_files.keys()
                if location in file.lower()
            ]

            location_dataframes = [
                nrfa_files[file] for file in location_files
            ]

            # Merge location-specific files
            merged_location_data = self.merge_location_data(
                [os.path.join(nrfa_directory, file) for file in location_files]
            )

            # Handle missing values
            merged_location_data = self.handle_missing_values(merged_location_da

            processed_nrfa_data[location] = merged_location_data

        self.processed_data['historical_nrfa'] = processed_nrfa_data
        return processed_nrfa_data

    def export_processed_data(self, output_directory: str):
        """
        Export processed data to CSV files.

        Args:
            output_directory (str): Directory to save processed data
        """
        os.makedirs(output_directory, exist_ok=True)

        for data_source, data in self.processed_data.items():
            source_output_dir = os.path.join(output_directory, data_source)
            os.makedirs(source_output_dir, exist_ok=True)

            if isinstance(data, dict):
                for location, df in data.items():
                    output_path = os.path.join(
                        source_output_dir,
                        f'{location}_processed_data.csv'
                    )
                    df.to_csv(output_path, index=False)
            else:
                output_path = os.path.join(
                    source_output_dir,
                    'processed_data.csv'
                )
                data.to_csv(output_path, index=False)

    def run_data_integration(self):
        """
        Run the full data integration process.
        """
        # Process Historical NRFA Data
```

```
        self.process_historical_nrfa_data()

        # TODO: Add processing for other data sources
        # - Real-time river data
        # - Weather data
        # - Flood risk areas data

def main():
    # Example usage
    data_directories = {
        'historical_nrfa': r'C:\Users\Administrator\NEWPROJECT\cleaned_data\rive
        # Add other data source directories as needed
    }

    output_directory = r'C:\Users\Administrator\NEWPROJECT\processed_data'

    # Initialize and run data integration
    integrator = DataIntegrationProcessor(data_directories)
    integrator.run_data_integration()

    # Export processed data
    integrator.export_processed_data(output_directory)
    print(f"Processed data exported to {output_directory}")

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

```
Processed data exported to C:\Users\Administrator\NEWPROJECT\processed_data
```

In [156…

```
import pandas as pd
import numpy as np
import os
from datetime import datetime
from typing import List, Dict, Any

class RealTimeDataProcessor:
    def __init__(self, data_directory: str):
        """
        Initialize the real-time data processor.

        Args:
            data_directory (str): Directory containing real-time monitoring CSV
        """
        self.data_directory = data_directory
        self.processed_data = {}

    def load_real_time_csv_files(self) -> List[pd.DataFrame]:
        """
        Load all CSV files from the specified directory.

        Returns:
            List of DataFrames containing real-time monitoring data
        """
        # Find all CSV files in the directory
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.

        # Sort files to ensure chronological processing
        csv_files.sort()

        # Load DataFrames
```

```
        dataframes = []
        for filename in csv_files:
            file_path = os.path.join(self.data_directory, filename)
            df = pd.read_csv(file_path)
            dataframes.append(df)

        return dataframes

    def standardize_dataframe(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Standardize the DataFrame columns and data types.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: Standardized DataFrame
        """
        # Rename columns to lowercase and remove spaces
        df.columns = df.columns.str.lower().str.replace(' ', '_')

        # Convert timestamp columns to datetime
        timestamp_columns = [col for col in df.columns if 'timestamp' in col]
        for col in timestamp_columns:
            df[col] = pd.to_datetime(df[col], utc=True)

        # Ensure numeric columns are properly typed
        numeric_columns = ['river_level', 'rainfall']
        for col in numeric_columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

        return df

    def combine_real_time_data(self, dataframes: List[pd.DataFrame]) -> pd.DataF
        """
        Combine multiple real-time data DataFrames.

        Args:
            dataframes (List[pd.DataFrame]): List of real-time DataFrames

        Returns:
            pd.DataFrame: Combined and processed DataFrame
        """
        # Standardize each DataFrame
        standardized_dfs = [self.standardize_dataframe(df) for df in dataframes]

        # Concatenate DataFrames
        combined_df = pd.concat(standardized_dfs, ignore_index=True)

        # Remove duplicate entries
        combined_df.drop_duplicates(
            subset=['river_timestamp', 'location_name'],
            keep='last',
            inplace=True
        )

        # Sort by timestamp
        combined_df.sort_values('river_timestamp', inplace=True)

        return combined_df
```

```
    def handle_missing_values(self, df: pd.DataFrame) -> pd.DataFrame:
        """
        Handle missing values in the DataFrame.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: DataFrame with handled missing values
        """
        # Identify numeric columns
        numeric_columns = ['river_level', 'rainfall']

        # Interpolate missing values
        df[numeric_columns] = df[numeric_columns].interpolate(
            method='linear',
            limit_direction='both'
        )

        # Fill any remaining NaNs with 0 or method ffill
        df[numeric_columns] = df[numeric_columns].fillna(0)

        return df

    def generate_location_summaries(self, df: pd.DataFrame) -> Dict[str, Dict[st
        """
        Generate summary statistics for each location.

        Args:
            df (pd.DataFrame): Combined real-time DataFrame

        Returns:
            Dict of location-specific summaries
        """
        location_summaries = {}

        for location in df['location_name'].unique():
            location_data = df[df['location_name'] == location]

            summary = {
                'total_records': len(location_data),
                'river_level': {
                    'mean': location_data['river_level'].mean(),
                    'median': location_data['river_level'].median(),
                    'min': location_data['river_level'].min(),
                    'max': location_data['river_level'].max(),
                    'std': location_data['river_level'].std()
                },
                'rainfall': {
                    'mean': location_data['rainfall'].mean(),
                    'median': location_data['rainfall'].median(),
                    'min': location_data['rainfall'].min(),
                    'max': location_data['rainfall'].max(),
                    'std': location_data['rainfall'].std()
                },
                'time_range': {
                    'start': location_data['river_timestamp'].min(),
                    'end': location_data['river_timestamp'].max(),
                    'duration': (location_data['river_timestamp'].max() -
```

```
                                    location_data['river_timestamp'].min())
                    }
                }

                location_summaries[location] = summary

            return location_summaries

        def export_processed_data(self, output_directory: str):
            """
            Export processed real-time data and summaries.

            Args:
                output_directory (str): Directory to save processed data
            """
            # Ensure output directory exists
            os.makedirs(output_directory, exist_ok=True)

            # Export combined real-time data
            combined_data_path = os.path.join(output_directory, 'combined_real_time_
            self.processed_data['combined_data'].to_csv(combined_data_path, index=Fa

            # Export location summaries
            summaries_path = os.path.join(output_directory, 'real_time_location_summ
            import json
            with open(summaries_path, 'w') as f:
                json.dump(self.processed_data['location_summaries'], f, indent=4, de

            print(f"Processed real-time data exported to {output_directory}")

        def process_real_time_data(self):
            """
            Main method to process real-time monitoring data.
            """
            # Load CSV files
            dataframes = self.load_real_time_csv_files()

            # Combine and process data
            combined_df = self.combine_real_time_data(dataframes)

            # Handle missing values
            processed_df = self.handle_missing_values(combined_df)

            # Generate location summaries
            location_summaries = self.generate_location_summaries(processed_df)

            # Store processed data
            self.processed_data['combined_data'] = processed_df
            self.processed_data['location_summaries'] = location_summaries

            return processed_df

    def main():
        # Example usage
        data_directory = r'C:\Users\Administrator\NEWPROJECT\combined_data'
        output_directory = r'C:\Users\Administrator\NEWPROJECT\processed_data\real_t

        # Initialize processor
        processor = RealTimeDataProcessor(data_directory)
```

```
    try:
        # Process real-time data
        processed_data = processor.process_real_time_data()

        # Print basic information
        print("Real-Time Data Processing Summary:")
        for location, summary in processor.processed_data['location_summaries'].
            print(f"\n{location} Summary:")
            print(f"Total Records: {summary['total_records']}")
            print("River Level Statistics:")
            print(f"  Mean: {summary['river_level']['mean']:.3f}")
            print(f"  Min: {summary['river_level']['min']:.3f}")
            print(f"  Max: {summary['river_level']['max']:.3f}")
            print("Time Range:")
            print(f"  Start: {summary['time_range']['start']}")
            print(f"  End: {summary['time_range']['end']}")

        # Export processed data
        processor.export_processed_data(output_directory)

    except Exception as e:
        print(f"An error occurred during processing: {e}")
        import traceback
        traceback.print_exc()

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

```
Real-Time Data Processing Summary:

Rochdale Summary:
Total Records: 149
River Level Statistics:
  Mean: 0.251
  Min: 0.227
  Max: 0.293
Time Range:
  Start: 2025-01-30 11:15:00+00:00
  End: 2025-02-01 13:15:00+00:00

Manchester Racecourse Summary:
Total Records: 149
River Level Statistics:
  Mean: 1.104
  Min: 1.045
  Max: 1.203
Time Range:
  Start: 2025-01-30 11:15:00+00:00
  End: 2025-02-01 13:15:00+00:00

Bury Ground Summary:
Total Records: 149
River Level Statistics:
  Mean: 0.395
  Min: 0.370
  Max: 0.441
Time Range:
  Start: 2025-01-30 11:15:00+00:00
  End: 2025-02-01 13:15:00+00:00
Processed real-time data exported to C:\Users\Administrator\NEWPROJECT\processed_
data\real_time
```

In [159…

```python
import pandas as pd
import numpy as np
import os
import json
from typing import List, Dict, Any

class WeatherDataProcessor:
    def __init__(self, data_directory: str):
        """
        Initialize the weather data processor.

        Args:
            data_directory (str): Directory containing weather data files
        """
        self.data_directory = data_directory
        self.processed_data = {}

    def load_weather_files(self) -> List[pd.DataFrame]:
        """
        Load valid weather data files from the specified directory.

        Returns:
            List of DataFrames containing weather data
        """
        # Find all files in the directory
        files = [f for f in os.listdir(self.data_directory) if os.path.isfile(os
```

```
        # Load DataFrames
        dataframes = []
        for filename in files:
            file_path = os.path.join(self.data_directory, filename)

            try:
                # Try reading with different delimiters
                try:
                    # Try comma-separated
                    df = pd.read_csv(file_path)
                except:
                    try:
                        # Try tab-separated
                        df = pd.read_csv(file_path, delimiter='\t')
                    except Exception as e:
                        print(f"Could not read file {filename}: {e}")
                        continue

                # Filter out invalid or empty DataFrames
                if not df.empty and len(df.columns) > 1:
                    dataframes.append(df)

            except Exception as e:
                print(f"Error processing {filename}: {e}")

        return dataframes

    def process_weather_data_files(self, dataframes: List[pd.DataFrame]) -> pd.D
        """
        Process and combine weather data files.

        Args:
            dataframes (List[pd.DataFrame]): List of DataFrames to process

        Returns:
            pd.DataFrame: Processed and combined weather data
        """
        # Filter and process weather-related DataFrames
        weather_dfs = []

        for df in dataframes:
            # Normalize column names
            df.columns = [col.lower().replace(' ', '_').replace('(', '').replace

            # Look for weather-specific DataFrames
            if 'month' in df.columns and ('temperature_c' in df.columns or 'prec
                # Standardize key columns
                if 'temperature_c' in df.columns:
                    df['temperature_c'] = pd.to_numeric(df['temperature_c'], err

                if 'precipitation_mm' in df.columns:
                    df['precipitation_mm'] = pd.to_numeric(df['precipitation_mm'

                # Ensure station is uppercase
                if 'station' in df.columns:
                    df['station'] = df['station'].str.upper()

                weather_dfs.append(df)
```

```
        # Combine weather DataFrames
        if not weather_dfs:
            raise ValueError("No valid weather data found")

        combined_df = pd.concat(weather_dfs, ignore_index=True)

        # Remove duplicates if possible
        duplicate_cols = [col for col in ['month', 'station', 'temperature_c', '
        if duplicate_cols:
            combined_df.drop_duplicates(subset=duplicate_cols, keep='first', inp

        return combined_df

    def generate_location_summaries(self, df: pd.DataFrame) -> Dict[str, Dict[st
        """
        Generate summary statistics for each location and month.

        Args:
            df (pd.DataFrame): Combined weather DataFrame

        Returns:
            Dict of location-specific summaries
        """
        location_summaries = {}

        # Group by station and month
        if 'station' not in df.columns or 'month' not in df.columns:
            print("Warning: Could not generate summaries - missing station or mo
            return location_summaries

        # Group by station and month
        grouped = df.groupby(['station', 'month'])

        for (station, month), group in grouped:
            summary = {
                'total_records': len(group)
            }

            # Temperature summary
            if 'temperature_c' in group.columns:
                summary['temperature'] = {
                    'mean': group['temperature_c'].mean(),
                    'min': group['temperature_c'].min(),
                    'max': group['temperature_c'].max(),
                    'std': group['temperature_c'].std()
                }

            # Precipitation summary
            if 'precipitation_mm' in group.columns:
                summary['precipitation'] = {
                    'mean': group['precipitation_mm'].mean(),
                    'min': group['precipitation_mm'].min(),
                    'max': group['precipitation_mm'].max(),
                    'std': group['precipitation_mm'].std()
                }

            # Add grid information if available
            if 'grid_id' in group.columns:
                summary['grid_ids'] = list(group['grid_id'].unique())
```

```
            if 'grid_period' in group.columns:
                summary['grid_period'] = group['grid_period'].iloc[0]

            # Create nested dictionary
            if station not in location_summaries:
                location_summaries[station] = {}
            location_summaries[station][month] = summary

        return location_summaries

    def export_processed_data(self, output_directory: str):
        """
        Export processed weather data and summaries.

        Args:
            output_directory (str): Directory to save processed data
        """
        # Ensure output directory exists
        os.makedirs(output_directory, exist_ok=True)

        # Export combined weather data
        combined_data_path = os.path.join(output_directory, 'combined_weather_da
        self.processed_data['combined_data'].to_csv(combined_data_path, index=Fa

        # Export location summaries
        summaries_path = os.path.join(output_directory, 'weather_location_summar
        with open(summaries_path, 'w') as f:
            json.dump(self.processed_data['location_summaries'], f, indent=4)

        print(f"Processed weather data exported to {output_directory}")

    def process_weather_data(self):
        """
        Main method to process weather monitoring data.
        """
        # Load files
        dataframes = self.load_weather_files()

        # Print column names for debugging
        print("Columns in loaded dataframes:")
        for i, df in enumerate(dataframes):
            print(f"DataFrame {i} columns:", list(df.columns))

        # Combine and process data
        combined_df = self.process_weather_data_files(dataframes)

        # Generate location summaries
        location_summaries = self.generate_location_summaries(combined_df)

        # Store processed data
        self.processed_data['combined_data'] = combined_df
        self.processed_data['location_summaries'] = location_summaries

        return combined_df

def main():
    # Example usage
    data_directory = r'C:\Users\Administrator\NEWPROJECT\cleaned_data'
    output_directory = r'C:\Users\Administrator\NEWPROJECT\processed_data\weathe
```

```
    # Initialize processor
    processor = WeatherDataProcessor(data_directory)

    try:
        # Process weather data
        processed_data = processor.process_weather_data()

        # Print basic information
        print("\nWeather Data Processing Summary:")
        for station, months in processor.processed_data['location_summaries'].it
            print(f"\n{station} Summary:")
            for month, summary in months.items():
                print(f"  {month}:")
                if 'temperature' in summary:
                    print(f"    Temperature: {summary['temperature']['mean']:.1f
                if 'precipitation' in summary:
                    print(f"    Precipitation: {summary['precipitation']['mean']

        # Export processed data
        processor.export_processed_data(output_directory)

    except Exception as e:
        print(f"An error occurred during processing: {e}")
        import traceback
        traceback.print_exc()

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

```
Columns in loaded dataframes:
DataFrame 0 columns: ['@context', 'items', 'meta.publisher', 'meta.licence', 'met
a.documentation', 'meta.version', 'meta.comment', 'meta.hasFormat', 'label', 'des
cription', 'river', 'area_type', 'quick_dial', 'county', 'meta.limit', 'risk_leve
l', 'monitoring_stations', 'last_updated', 'data_quality']
DataFrame 1 columns: ['Station', 'Water_Year', 'Date', 'Time', 'Stage_m', 'Flow_m
3s', 'Rating', 'Source']
DataFrame 2 columns: ['Month', 'Station', 'Grid_ID', 'Precipitation_mm', 'Grid',
'Period']
DataFrame 3 columns: ['Month', 'Station', 'Grid_ID', 'Temperature_C', 'Grid', 'Pe
riod']
DataFrame 4 columns: ['Month', 'Station', 'Grid_ID', 'Temperature_C', 'Grid', 'Pe
riod', 'Precipitation_mm']
DataFrame 5 columns: ['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)',
'Rating', 'Datetime', 'Station']

Weather Data Processing Summary:

BURY MANCHESTER Summary:
  April:
    Temperature: 8.1°C
    Precipitation: 79.0mm
  August:
    Temperature: 15.2°C
    Precipitation: 111.0mm
  December:
    Temperature: 4.1°C
    Precipitation: 157.0mm
  February:
    Temperature: 4.1°C
    Precipitation: 112.0mm
  January:
    Temperature: 3.8°C
    Precipitation: 131.0mm
  July:
    Temperature: 15.5°C
    Precipitation: 100.0mm
  June:
    Temperature: 13.6°C
    Precipitation: 93.0mm
  March:
    Temperature: 5.7°C
    Precipitation: 95.0mm
  May:
    Temperature: 11.0°C
    Precipitation: 83.0mm
  November:
    Temperature: 6.5°C
    Precipitation: 138.0mm
  October:
    Temperature: 9.7°C
    Precipitation: 134.0mm
  September:
    Temperature: 12.9°C
    Precipitation: 110.0mm

MANCHESTER RACECOURSE Summary:
  April:
    Temperature: 9.4°C
    Precipitation: 65.7mm
```

```
    August:
      Temperature: 16.5°C
      Precipitation: 93.7mm
    December:
      Temperature: 5.3°C
      Precipitation: 124.3mm
    February:
      Temperature: 5.4°C
      Precipitation: 88.0mm
    January:
      Temperature: 5.0°C
      Precipitation: 103.7mm
    July:
      Temperature: 16.8°C
      Precipitation: 89.3mm
    June:
      Temperature: 15.0°C
      Precipitation: 82.3mm
    March:
      Temperature: 7.0°C
      Precipitation: 75.7mm
    May:
      Temperature: 12.4°C
      Precipitation: 70.3mm
    November:
      Temperature: 7.6°C
      Precipitation: 110.7mm
    October:
      Temperature: 11.0°C
      Precipitation: 112.0mm
    September:
      Temperature: 14.2°C
      Precipitation: 93.3mm

  ROCHDALE Summary:
    April:
      Temperature: 7.9°C
      Precipitation: 77.0mm
    August:
      Temperature: 15.1°C
      Precipitation: 110.0mm
    December:
      Temperature: 4.0°C
      Precipitation: 154.0mm
    February:
      Temperature: 3.9°C
      Precipitation: 110.0mm
    January:
      Temperature: 3.6°C
      Precipitation: 131.0mm
    July:
      Temperature: 15.3°C
      Precipitation: 105.0mm
    June:
      Temperature: 13.4°C
      Precipitation: 92.0mm
    March:
      Temperature: 5.4°C
      Precipitation: 96.0mm
    May:
```

```
    Temperature: 10.7°C
    Precipitation: 77.0mm
  November:
    Temperature: 6.2°C
    Precipitation: 136.0mm
  October:
    Temperature: 9.6°C
    Precipitation: 130.0mm
  September:
    Temperature: 12.8°C
    Precipitation: 109.0mm
Processed weather data exported to C:\Users\Administrator\NEWPROJECT\processed_da
ta\weather
```

In [163…

```
import pandas as pd
import numpy as np
import os
import json
from typing import Dict, Any, List

class FloodDataProcessor:
    def __init__(self, data_directories: Dict[str, str]):
        """
        Initialize the flood data processor.

        Args:
            data_directories (dict): Directories containing flood-related data
        """
        self.data_directories = data_directories
        self.processed_data = {}

    def json_serializer(self, obj):
        """
        Custom JSON serializer to handle non-serializable types.

        Args:
            obj: Object to serialize

        Returns:
            Serializable representation of the object
        """
        if isinstance(obj, pd.Timestamp):
            return obj.isoformat()
        raise TypeError(f"Type {type(obj)} not serializable")

    def load_flood_history_data(self, file_path: str) -> pd.DataFrame:
        """
        Load and process flood history data.

        Args:
            file_path (str): Path to the flood history CSV file

        Returns:
            pd.DataFrame: Processed flood history data
        """
        try:
            # Read CSV file
            df = pd.read_csv(file_path)

            # Normalize column names
```

```
            df.columns = df.columns.str.lower().str.replace(' ', '_')

            # Convert datetime columns if present
            datetime_columns = ['timeraised', 'timemessagechanged', 'timeseverit
            for col in datetime_columns:
                if col in df.columns:
                    df[col] = pd.to_datetime(df[col], errors='coerce')

            return df

        except Exception as e:
            print(f"Error loading flood history data: {e}")
            return pd.DataFrame()

    def load_flood_areas_data(self, file_path: str) -> pd.DataFrame:
        """
        Load and process flood areas data.

        Args:
            file_path (str): Path to the flood areas CSV file

        Returns:
            pd.DataFrame: Processed flood areas data
        """
        try:
            # Read CSV file
            df = pd.read_csv(file_path)

            return df

        except Exception as e:
            print(f"Error loading flood areas data: {e}")
            return pd.DataFrame()

    def load_flood_summary_json(self, file_path: str) -> Dict[str, Any]:
        """
        Load flood data summary from JSON file.

        Args:
            file_path (str): Path to the flood data summary JSON file

        Returns:
            dict: Flood data summary
        """
        try:
            with open(file_path, 'r') as f:
                flood_summary = json.load(f)
            return flood_summary

        except Exception as e:
            print(f"Error loading flood data summary: {e}")
            return {}

    def generate_flood_summaries(self, flood_history_df: pd.DataFrame) -> Dict[s
        """
        Generate comprehensive summaries of flood data.

        Args:
            flood_history_df (pd.DataFrame): Processed flood history data
```

```
        Returns:
            dict: Comprehensive flood data summaries
        """
        summaries = {
            'flood_history_summary': {
                'total_records': len(flood_history_df),
                'unique_counties': flood_history_df['county'].nunique(),
                'unique_rivers': flood_history_df['river'].nunique(),
                'severity_distribution': flood_history_df['severity'].value_coun
                'time_analysis': {}
            }
        }

        # Time-based analysis
        time_columns = ['timeraised', 'timemessagechanged', 'timeseveritychanged
        for col in time_columns:
            if col in flood_history_df.columns:
                summaries['flood_history_summary']['time_analysis'][col] = {
                    'earliest': flood_history_df[col].min(),
                    'latest': flood_history_df[col].max(),
                    'total_duration': str(flood_history_df[col].max() - flood_hi
                }

        return summaries

    def export_processed_flood_data(self, output_directory: str):
        """
        Export processed flood data and summaries.

        Args:
            output_directory (str): Directory to save processed data
        """
        # Ensure output directory exists
        os.makedirs(output_directory, exist_ok=True)

        # Export flood history data
        if 'flood_history_data' in self.processed_data:
            flood_history_path = os.path.join(output_directory, 'processed_flood
            self.processed_data['flood_history_data'].to_csv(flood_history_path,

        # Export flood areas data
        if 'flood_areas_data' in self.processed_data:
            flood_areas_path = os.path.join(output_directory, 'processed_flood_a
            self.processed_data['flood_areas_data'].to_csv(flood_areas_path, ind

        # Export summaries with custom serialization
        if 'flood_summaries' in self.processed_data:
            summaries_path = os.path.join(output_directory, 'flood_data_summarie
            with open(summaries_path, 'w') as f:
                json.dump(
                    self.processed_data['flood_summaries'],
                    f,
                    indent=4,
                    default=self.json_serializer
                )

        print(f"Processed flood data exported to {output_directory}")

    def process_flood_data(self):
        """
```

```
        Main method to process flood-related data.

        Returns:
            Dict: Processed flood data and summaries
        """
        # Load flood history data
        flood_history_path = os.path.join(
            self.data_directories.get('flood_history', ''),
            'standardized_flood_history.csv'
        )
        flood_history_df = self.load_flood_history_data(flood_history_path)

        # Load flood areas data
        flood_areas_path = os.path.join(
            self.data_directories.get('main_directory', ''),
            'cleaned_flood_areas.csv'
        )
        flood_areas_df = self.load_flood_areas_data(flood_areas_path)

        # Load flood data summary JSON
        flood_summary_path = os.path.join(
            self.data_directories.get('main_directory', ''),
            'flood_data_summary.json'
        )
        flood_summary_json = self.load_flood_summary_json(flood_summary_path)

        # Generate summaries
        flood_summaries = self.generate_flood_summaries(flood_history_df)

        # Store processed data
        self.processed_data = {
            'flood_history_data': flood_history_df,
            'flood_areas_data': flood_areas_df,
            'flood_summary_json': flood_summary_json,
            'flood_summaries': flood_summaries
        }

        return self.processed_data

def main():
    # Directories
    data_directories = {
        'flood_history': r'C:\Users\Administrator\NEWPROJECT\cleaned_data\flood_
        'main_directory': r'C:\Users\Administrator\NEWPROJECT\cleaned_data',
        'output': r'C:\Users\Administrator\NEWPROJECT\processed_data\flood'
    }

    # Initialize processor
    processor = FloodDataProcessor(data_directories)

    try:
        # Process flood data
        processed_data = processor.process_flood_data()

        # Print basic information
        print("\nFlood Data Processing Summary:")

        # Flood History Data Summary
        if 'flood_history_data' in processed_data:
            flood_history_df = processed_data['flood_history_data']
```

```
            print("\nFlood History Data:")
            print(f"Total Records: {len(flood_history_df)}")
            print("Columns:", list(flood_history_df.columns))

            # Display flood summaries
            summaries = processed_data['flood_summaries'].get('flood_history_sum
            print("\nFlood History Summaries:")
            print(json.dumps(summaries, indent=2, default=processor.json_seriali

        # Flood Areas Data Summary
        if 'flood_areas_data' in processed_data:
            flood_areas_df = processed_data['flood_areas_data']
            print("\nFlood Areas Data:")
            print(f"Total Records: {len(flood_areas_df)}")
            print("Columns:", list(flood_areas_df.columns))

        # Flood Summary JSON
        if 'flood_summary_json' in processed_data:
            flood_summary = processed_data['flood_summary_json']
            print("\nFlood Data Summary JSON:")
            print(json.dumps(flood_summary, indent=2))

        # Export processed data
        processor.export_processed_flood_data(data_directories['output'])

    except Exception as e:
        print(f"An error occurred during processing: {e}")
        import traceback
        traceback.print_exc()

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

```
Flood Data Processing Summary:

Flood History Data:
Total Records: 1152
Columns: ['@id', 'description', 'eaareaname', 'earegionname', 'floodarea_@id', 'f
loodarea_county', 'floodarea_notation', 'floodarea_polygon', 'floodarea_riverorse
a', 'floodareaid', 'istidal', 'message', 'severity', 'severitylevel', 'timemessag
echanged', 'timeraised', 'timeseveritychanged', 'source_file', 'label', 'river',
'area_type', 'quick_dial', 'county', 'floodwatcharea', 'fwdcode', 'lat', 'long',
'notation', 'polygon', 'quickdialnumber', 'riverorsea']

Flood History Summaries:
{
  "total_records": 1152,
  "unique_counties": 203,
  "unique_rivers": 318,
  "severity_distribution": {
    "Flood alert": 98,
    "Warning no longer in force": 40,
    "Flood warning": 14
  },
  "time_analysis": {
    "timeraised": {
      "earliest": "2025-01-27T13:52:24",
      "latest": "2025-01-30T12:23:44",
      "total_duration": "2 days 22:31:20"
    },
    "timemessagechanged": {
      "earliest": "2025-01-27T13:52:00",
      "latest": "2025-01-30T12:23:00",
      "total_duration": "2 days 22:31:00"
    },
    "timeseveritychanged": {
      "earliest": "2024-11-24T18:30:00",
      "latest": "2025-01-30T12:23:00",
      "total_duration": "66 days 17:53:00"
    }
  }
}

Flood Areas Data:
Total Records: 5
Columns: ['@context', 'items', 'meta.publisher', 'meta.licence', 'meta.documentat
ion', 'meta.version', 'meta.comment', 'meta.hasFormat', 'label', 'description',
'river', 'area_type', 'quick_dial', 'county', 'meta.limit', 'risk_level', 'monito
ring_stations', 'last_updated', 'data_quality']

Flood Data Summary JSON:
{
  "total_areas": 5,
  "rivers_covered": 3,
  "high_risk_areas": 1,
  "areas_with_stations": 5,
  "data_quality_distribution": {
    "High": 5
  }
}
Processed flood data exported to C:\Users\Administrator\NEWPROJECT\processed_data
\flood
```

In [165…

```
import pandas as pd
import numpy as np
import json

class ManchesterFloodDataAnalyzer:
    def __init__(self, flood_history_path, flood_summary_path):
        """
        Initialize the analyzer with paths to flood data files.

        Args:
            flood_history_path (str): Path to flood history CSV
            flood_summary_path (str): Path to flood data summary JSON
        """
        self.flood_history_df = pd.read_csv(flood_history_path)

        with open(flood_summary_path, 'r') as f:
            self.flood_summary = json.load(f)

        # Stations of interest
        self.target_stations = [
            'Rochdale',
            'Manchester Racecourse',
            'Bury Ground'
        ]

    def filter_manchester_flood_data(self):
        """
        Filter flood history data specific to Greater Manchester and target stat

        Returns:
            pd.DataFrame: Filtered flood data
        """
        # Create boolean mask for filtering
        manchester_mask = (
            # Check for Manchester-related counties or areas
            self.flood_history_df['county'].str.contains('Manchester', case=Fals
            self.flood_history_df['eaareaname'].str.contains('Manchester', case=
            self.flood_history_df['earegionname'].str.contains('Manchester', cas
            self.flood_history_df['river'].str.contains('Manchester', case=False
        )

        # Apply filtering
        manchester_flood_data = self.flood_history_df[manchester_mask]

        return manchester_flood_data

    def analyze_manchester_flood_risks(self):
        """
        Analyze flood risks specific to Greater Manchester.

        Returns:
            dict: Comprehensive analysis of Manchester flood risks
        """
        # Filter Manchester-specific data
        manchester_data = self.filter_manchester_flood_data()

        # Analyze severity levels
        severity_analysis = manchester_data['severity'].value_counts()
```

```
        # Time-based analysis
        manchester_data['timeraised'] = pd.to_datetime(manchester_data['timerais
        time_analysis = {
            'total_incidents': len(manchester_data),
            'date_range': {
                'earliest': manchester_data['timeraised'].min(),
                'latest': manchester_data['timeraised'].max()
            },
            'severity_distribution': severity_analysis.to_dict()
        }

        # Geographical analysis
        geo_analysis = {
            'unique_rivers': manchester_data['river'].nunique(),
            'unique_counties': manchester_data['county'].nunique()
        }

        # Combine analyses
        flood_risk_summary = {
            'time_analysis': time_analysis,
            'geographical_analysis': geo_analysis,
            'stations_of_interest': {
                'total_stations': len(self.target_stations),
                'stations': self.target_stations
            }
        }

        return flood_risk_summary

    def compare_local_stations(self):
        """
        Compare flood risks across local stations of interest.

        Returns:
            dict: Comparative analysis of flood risks for target stations
        """
        manchester_data = self.filter_manchester_flood_data()

        station_analysis = {}
        for station in self.target_stations:
            # Filter for specific station
            station_data = manchester_data[
                manchester_data['river'].str.contains(station, case=False, na=Fa
                manchester_data['county'].str.contains(station, case=False, na=F
            ]

            station_analysis[station] = {
                'total_incidents': len(station_data),
                'severity_distribution': station_data['severity'].value_counts()
                'date_range': {
                    'earliest': station_data['timeraised'].min() if not station_
                    'latest': station_data['timeraised'].max() if not station_da
                }
            }

        return station_analysis

    def export_analysis(self, output_path):
        """
        Export flood risk analysis to a JSON file.
```

```
        Args:
            output_path (str): Path to export the analysis
        """
        # Combine different analyses
        full_analysis = {
            'manchester_flood_risks': self.analyze_manchester_flood_risks(),
            'station_specific_analysis': self.compare_local_stations(),
            'national_summary': self.flood_summary
        }

        # Export to JSON
        with open(output_path, 'w') as f:
            json.dump(full_analysis, f, indent=4, default=str)

        print(f"Flood risk analysis exported to {output_path}")

def main():
    # Paths to input files
    flood_history_path = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\flood_
    flood_summary_path = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\flood_
    output_path = r'C:\Users\Administrator\NEWPROJECT\processed_data\manchester_

    # Initialize analyzer
    analyzer = ManchesterFloodDataAnalyzer(
        flood_history_path,
        flood_summary_path
    )

    # Run analysis
    try:
        # Print Manchester-specific flood data
        manchester_data = analyzer.filter_manchester_flood_data()
        print("\nManchester-Specific Flood Data:")
        print(f"Total incidents: {len(manchester_data)}")
        print("\nSeverity Distribution:")
        print(manchester_data['severity'].value_counts())

        # Analyze flood risks
        manchester_risks = analyzer.analyze_manchester_flood_risks()
        print("\nManchester Flood Risk Summary:")
        print(json.dumps(manchester_risks, indent=2, default=str))

        # Compare local stations
        station_analysis = analyzer.compare_local_stations()
        print("\nStation-Specific Flood Analysis:")
        print(json.dumps(station_analysis, indent=2, default=str))

        # Export full analysis
        analyzer.export_analysis(output_path)

    except Exception as e:
        print(f"An error occurred: {e}")
        import traceback
        traceback.print_exc()

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

```
Manchester-Specific Flood Data:
Total incidents: 13

Severity Distribution:
severity
Warning no longer in force    1
Name: count, dtype: int64

Manchester Flood Risk Summary:
```


```
{
  "time_analysis": {
    "total_incidents": 13,
    "date_range": {
      "earliest": "2025-01-30 09:29:23",
      "latest": "2025-01-30 09:29:23"
    },
    "severity_distribution": {
      "Warning no longer in force": 1
    }
  },
  "geographical_analysis": {
    "unique_rivers": 5,
    "unique_counties": 5
  },
  "stations_of_interest": {
    "total_stations": 3,
    "stations": [
      "Rochdale",
      "Manchester Racecourse",
      "Bury Ground"
    ]
  }
}
```


```
Station-Specific Flood Analysis:
```


```
{
  "Rochdale": {
    "total_incidents": 2,
    "severity_distribution": {},
    "date_range": {
      "earliest": NaN,
      "latest": NaN
    }
  },
  "Manchester Racecourse": {
    "total_incidents": 0,
    "severity_distribution": {},
    "date_range": {
      "earliest": null,
      "latest": null
    }
  },
  "Bury Ground": {
    "total_incidents": 0,
    "severity_distribution": {},
    "date_range": {
      "earliest": null,
      "latest": null
    }
  }
```

```
}
Flood risk analysis exported to C:\Users\Administrator\NEWPROJECT\processed_data
\manchester_flood_analysis.json
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\3639850025.py:61: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  manchester_data['timeraised'] = pd.to_datetime(manchester_data['timeraised'])
C:\Users\Administrator\AppData\Local\Temp\ipykernel_22600\3639850025.py:61: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  manchester_data['timeraised'] = pd.to_datetime(manchester_data['timeraised'])
```

Unified Data Model Development: Detailed Approach

Create Unified Data Model:

It combines historical, real-time, and weather data into a single unified data model. Adds source and location labels for each data entry. Sorts the data by date (if available).

In [166…

```
import pandas as pd
import numpy as np
import os
from typing import Dict, List, Any

class UnifiedDataModelBuilder:
    def __init__(self, data_directories: Dict[str, str]):
        """
        Initialize the unified data model builder.

        Args:
            data_directories (dict): Directories containing different data sourc
        """
        self.data_directories = data_directories
        self.processed_data = {}

    def load_historical_data(self) -> Dict[str, pd.DataFrame]:
        """
        Load historical river data from NRFA sources.

        Returns:
            dict: Historical data for different locations
        """
        historical_dir = self.data_directories.get('historical_nrfa', '')
        csv_files = [f for f in os.listdir(historical_dir) if f.endswith('.csv')

        historical_data = {}

        for filename in csv_files:
            file_path = os.path.join(historical_dir, filename)
            df = pd.read_csv(file_path)
```

```
            # Standardize column names
            df.columns = df.columns.str.lower().str.replace(' ', '_')

            # Convert date columns
            date_columns = [col for col in df.columns if 'date' in col]
            for col in date_columns:
                df[col] = pd.to_datetime(df[col], errors='coerce')

            # Extract location name from filename
            location = filename.split('_')[0]
            historical_data[location] = df

        return historical_data

    def load_real_time_data(self) -> pd.DataFrame:
        """
        Load and process real-time monitoring data.

        Returns:
            pd.DataFrame: Processed real-time data
        """
        real_time_dir = self.data_directories.get('real_time', '')
        csv_files = [f for f in os.listdir(real_time_dir) if f.endswith('.csv')]

        real_time_dataframes = []

        for filename in csv_files:
            file_path = os.path.join(real_time_dir, filename)
            df = pd.read_csv(file_path)

            # Standardize column names
            df.columns = df.columns.str.lower().str.replace(' ', '_')

            # Convert timestamp columns
            timestamp_columns = [col for col in df.columns if 'timestamp' in col
            for col in timestamp_columns:
                df[col] = pd.to_datetime(df[col], errors='coerce')

            real_time_dataframes.append(df)

        # Combine real-time dataframes
        return pd.concat(real_time_dataframes, ignore_index=True)

    def load_weather_data(self) -> pd.DataFrame:
        """
        Load and process weather data.

        Returns:
            pd.DataFrame: Processed weather data
        """
        weather_dir = self.data_directories.get('weather', '')
        csv_files = [f for f in os.listdir(weather_dir) if f.endswith('.csv')]

        weather_dataframes = []

        for filename in csv_files:
            file_path = os.path.join(weather_dir, filename)
            df = pd.read_csv(file_path)
```

```
            # Standardize column names
            df.columns = df.columns.str.lower().str.replace(' ', '_')

            # Convert numeric columns
            numeric_columns = ['temperature_c', 'precipitation_mm']
            for col in numeric_columns:
                if col in df.columns:
                    df[col] = pd.to_numeric(df[col], errors='coerce')

            weather_dataframes.append(df)

        # Combine weather dataframes
        return pd.concat(weather_dataframes, ignore_index=True)

    def create_unified_data_model(self) -> pd.DataFrame:
        """
        Create a unified data model by integrating different data sources.

        Returns:
            pd.DataFrame: Comprehensive unified data model
        """
        # Load different data sources
        historical_data = self.load_historical_data()
        real_time_data = self.load_real_time_data()
        weather_data = self.load_weather_data()

        # Prepare for data integration
        unified_data_list = []

        # Integrate historical data
        for location, df in historical_data.items():
            location_data = df.copy()
            location_data['data_source'] = 'historical'
            location_data['location'] = location
            unified_data_list.append(location_data)

        # Integrate real-time data
        real_time_data['data_source'] = 'real_time'
        unified_data_list.append(real_time_data)

        # Integrate weather data
        weather_data['data_source'] = 'weather'
        unified_data_list.append(weather_data)

        # Combine all data sources
        unified_df = pd.concat(unified_data_list, ignore_index=True)

        # Sort by date if a date column exists
        date_columns = [col for col in unified_df.columns if 'date' in col or 't
        if date_columns:
            unified_df.sort_values(by=date_columns[0], inplace=True)

        # Store processed data
        self.processed_data['unified_model'] = unified_df

        return unified_df

    def export_unified_data_model(self, output_directory: str):
        """
        Export the unified data model to a CSV file.
```

```
        Args:
            output_directory (str): Directory to save the unified data model
        """
        # Ensure output directory exists
        os.makedirs(output_directory, exist_ok=True)

        # Export unified data model
        output_path = os.path.join(output_directory, 'unified_data_model.csv')
        self.processed_data['unified_model'].to_csv(output_path, index=False)

        print(f"Unified data model exported to {output_path}")

    def analyze_data_coverage(self) -> Dict[str, Any]:
        """
        Analyze coverage and characteristics of the unified data model.

        Returns:
            dict: Data coverage and characteristics summary
        """
        unified_df = self.processed_data.get('unified_model')

        if unified_df is None:
            return {}

        # Analyze data sources
        source_coverage = unified_df['data_source'].value_counts()

        # Analyze locations
        location_coverage = unified_df['location'].value_counts() if 'location'

        # Date range analysis
        date_columns = [col for col in unified_df.columns if 'date' in col or 't
        date_range = {}

        if date_columns:
            date_range = {
                'earliest_date': unified_df[date_columns[0]].min(),
                'latest_date': unified_df[date_columns[0]].max(),
                'total_duration': str(unified_df[date_columns[0]].max() - unifie
            }

        return {
            'data_source_coverage': source_coverage.to_dict(),
            'location_coverage': location_coverage.to_dict(),
            'date_range': date_range
        }

def main():
    # Directories for different data sources
    data_directories = {
        'historical_nrfa': r'C:\Users\Administrator\NEWPROJECT\processed_data\hi
        'real_time': r'C:\Users\Administrator\NEWPROJECT\processed_data\real_tim
        'weather': r'C:\Users\Administrator\NEWPROJECT\processed_data\weather',
        'output': r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_mod
    }

    # Initialize data model builder
    model_builder = UnifiedDataModelBuilder(data_directories)
```

```
    try:
        # Create unified data model
        unified_data_model = model_builder.create_unified_data_model()

        # Analyze data coverage
        data_coverage = model_builder.analyze_data_coverage()

        # Print data coverage summary
        print("\nUnified Data Model - Coverage Summary:")
        print(json.dumps(data_coverage, indent=2, default=str))

        # Export unified data model
        model_builder.export_unified_data_model(data_directories['output'])

    except Exception as e:
        print(f"An error occurred: {e}")
        import traceback
        traceback.print_exc()

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

```
Unified Data Model - Coverage Summary:
{
  "data_source_coverage": {
    "historical": 34118,
    "real_time": 447,
    "weather": 108
  },
  "location_coverage": {
    "bury": 22918,
    "rochdale": 11118,
    "manchester": 82
  },
  "date_range": {
    "earliest_date": "1941-10-24 00:00:00",
    "latest_date": "2023-09-30 00:00:00",
    "total_duration": "29926 days 00:00:00"
  }
}
Unified data model exported to C:\Users\Administrator\NEWPROJECT\processed_data\u
nified_model\unified_data_model.csv
```

Data Quality Assessment of our Unified Data Model

In [173…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import json

class DataQualityAssessor:
    def __init__(self, unified_data_path):
        """
        Initialize the data quality assessor.

        Args:
            unified_data_path (str): Path to the unified data model CSV
```

```
        """
        # Read the CSV file with flexible dtype handling
        self.unified_data = pd.read_csv(
            unified_data_path,
            low_memory=False
        )

        # Standardize data types
        self._standardize_data_types()

        self.quality_report = {}

    def _standardize_data_types(self):
        """
        Standardize data types across the DataFrame.
        """
        # Identify and convert columns
        for column in self.unified_data.columns:
            # Try to convert to numeric where possible
            try:
                # Attempt numeric conversion
                converted = pd.to_numeric(self.unified_data[column], errors='coe
                if not converted.isna().all():
                    self.unified_data[column] = converted
            except:
                pass

            # Handle datetime columns
            if 'date' in column.lower() or 'time' in column.lower():
                try:
                    # Specify a format to avoid warning
                    self.unified_data[column] = pd.to_datetime(
                        self.unified_data[column],
                        errors='coerce',
                        format='mixed'
                    )
                except:
                    pass

        # Print column types for verification
        print("Column Types After Standardization:")
        print(self.unified_data.dtypes)

    def assess_completeness(self):
        """
        Assess data completeness across different dimensions.

        Returns:
            dict: Completeness assessment results
        """
        completeness = {
            'total_records': len(self.unified_data),
            'columns_completeness': {},
            'location_completeness': {},
            'data_source_completeness': {}
        }

        # Check column-level completeness
        for column in self.unified_data.columns:
            completeness['columns_completeness'][column] = {
```

```
                'total_records': len(self.unified_data),
                'missing_records': self.unified_data[column].isna().sum(),
                'missing_percentage': round(self.unified_data[column].isna().mea
            }

        # Location-based completeness
        if 'location' in self.unified_data.columns:
            location_counts = self.unified_data['location'].value_counts()
            for location, count in location_counts.items():
                completeness['location_completeness'][location] = {
                    'total_records': count,
                    'missing_records': self.unified_data[
                        (self.unified_data['location'] == location) &
                        self.unified_data.isna().any(axis=1)
                    ].shape[0]
                }

        # Data source completeness
        if 'data_source' in self.unified_data.columns:
            source_counts = self.unified_data['data_source'].value_counts()
            completeness['data_source_completeness'] = source_counts.to_dict()

        return completeness

    def identify_anomalies_and_outliers(self):
        """
        Identify statistical anomalies and outliers in numeric columns.

        Returns:
            dict: Anomalies and outliers analysis
        """
        # Identify numeric columns
        numeric_columns = self.unified_data.select_dtypes(include=[np.number]).c

        outliers_analysis = {}

        for column in numeric_columns:
            # Calculate IQR-based outliers
            Q1 = self.unified_data[column].quantile(0.25)
            Q3 = self.unified_data[column].quantile(0.75)
            IQR = Q3 - Q1

            # Define outlier boundaries
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            # Identify outliers
            outliers = self.unified_data[
                (self.unified_data[column] < lower_bound) |
                (self.unified_data[column] > upper_bound)
            ]

            outliers_analysis[column] = {
                'total_records': len(self.unified_data),
                'outliers_count': len(outliers),
                'outliers_percentage': round(len(outliers) / len(self.unified_da
                'lower_bound': lower_bound,
                'upper_bound': upper_bound,
                'mean': self.unified_data[column].mean(),
                'median': self.unified_data[column].median(),
```

```
                'standard_deviation': self.unified_data[column].std()
            }

        return outliers_analysis

    def check_temporal_consistency(self):
        """
        Assess temporal consistency and gaps in the data.

        Returns:
            dict: Temporal consistency analysis
        """
        # Identify datetime columns
        datetime_columns = self.unified_data.select_dtypes(include=['datetime64'

        if len(datetime_columns) == 0:
            return {"error": "No datetime columns found"}

        temporal_analysis = {}

        for column in datetime_columns:
            # Sort by date
            temporal_data = self.unified_data.sort_values(column)

            # Calculate time differences
            temporal_data['time_diff'] = temporal_data[column].diff()

            # Analyze time gaps
            time_gaps = temporal_data[temporal_data['time_diff'] > pd.Timedelta(

            temporal_analysis[column] = {
                'total_records': len(temporal_data),
                'date_range': {
                    'start': temporal_data[column].min(),
                    'end': temporal_data[column].max(),
                    'total_duration': str(temporal_data[column].max() - temporal
                },
                'time_gaps': {
                    'total_gaps': len(time_gaps),
                    'max_gap': time_gaps['time_diff'].max() if not time_gaps.emp
                    'mean_gap': time_gaps['time_diff'].mean() if not time_gaps.e
                },
                'unique_dates': temporal_data[column].nunique()
            }

        return temporal_analysis

    def _generate_visualizations(self, output_directory):
        """
        Generate visualizations to support data quality assessment.

        Args:
            output_directory (str): Directory to save visualizations
        """
        os.makedirs(output_directory, exist_ok=True)

        # Missingness heatmap
        plt.figure(figsize=(12, 8))
        sns.heatmap(self.unified_data.isna(), yticklabels=False, cbar=False, cma
        plt.title('Missingness Heatmap')
```

```
        plt.tight_layout()
        plt.savefig(os.path.join(output_directory, 'missingness_heatmap.png'))
        plt.close()

        # Distribution of numeric columns
        numeric_columns = self.unified_data.select_dtypes(include=[np.number]).c

        # Limit to first 9 numeric columns for subplot grid
        numeric_columns = numeric_columns[:9]

        plt.figure(figsize=(15, 10))
        for i, column in enumerate(numeric_columns, 1):
            plt.subplot(3, 3, i)
            sns.histplot(self.unified_data[column].dropna(), kde=True)
            plt.title(f'Distribution of {column}')
        plt.tight_layout()
        plt.savefig(os.path.join(output_directory, 'numeric_distributions.png'))
        plt.close()

    def comprehensive_data_quality_report(self, output_directory):
        """
        Generate a comprehensive data quality report.

        Args:
            output_directory (str): Directory to save report
        """
        # Create output directory
        os.makedirs(output_directory, exist_ok=True)

        # Perform assessments
        completeness = self.assess_completeness()
        outliers = self.identify_anomalies_and_outliers()
        temporal_analysis = self.check_temporal_consistency()

        # Generate visualizations
        self._generate_visualizations(output_directory)

        # Compile full report
        full_report = {
            'completeness': completeness,
            'outliers': outliers,
            'temporal_analysis': temporal_analysis
        }

        # Save report as JSON
        report_path = os.path.join(output_directory, 'data_quality_report.json')
        with open(report_path, 'w') as f:
            json.dump(full_report, f, indent=4, default=str)

        print(f"Comprehensive data quality report saved to {report_path}")

        return full_report

def main():
    # Path to unified data model
    unified_data_path = r'C:\Users\Administrator\NEWPROJECT\processed_data\unifi

    # Output directory for reports and visualizations
    output_directory = r'C:\Users\Administrator\NEWPROJECT\data_quality_assessme
```

```
    # Initialize assessor
    assessor = DataQualityAssessor(unified_data_path)

    try:
        # Generate comprehensive report
        report = assessor.comprehensive_data_quality_report(output_directory)

        # Print key highlights
        print("\nData Quality Assessment Highlights:")

        # Completeness Summary
        print("\nData Completeness:")
        completeness = report.get('completeness', {})
        print(f"Total Records: {completeness.get('total_records', 'N/A')}")

        # Columns with High Missingness
        print("\nColumns with Missingness:")
        for column, details in completeness.get('columns_completeness', {}).item
            if details.get('missing_percentage', 0) > 5:
                print(f"{column}: {details['missing_percentage']}% missing")

        # Outliers Summary
        print("\nOutliers Analysis:")
        outliers = report.get('outliers', {})
        for column, details in outliers.items():
            print(f"{column}:")
            print(f"  Outliers: {details['outliers_count']} ({details['outliers_
            print(f"  Bounds: [{details['lower_bound']}, {details['upper_bound']

        # Temporal Analysis
        print("\nTemporal Analysis:")
        temporal = report.get('temporal_analysis', {})
        print(json.dumps(temporal, indent=2, default=str))

        print("\nData quality report and visualizations saved.")

    except Exception as e:
        print(f"An error occurred during data quality assessment: {e}")
        import traceback
        traceback.print_exc()

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

```
Column Types After Standardization:
date                        datetime64[ns]
flow                               float64
extra_x                            float64
rainfall                           float64
extra_y                            float64
water_year                          object
time                        datetime64[ns]
stage_(m)                          float64
flow_(m3/s)                        float64
rating                              object
datetime                    datetime64[ns]
data_source                         object
location                            object
river_level                        float64
river_timestamp        datetime64[ns, UTC]
rainfall_timestamp     datetime64[ns, UTC]
location_name                       object
river_station_id                   float64
rainfall_station_id                float64
month                               object
station                             object
grid_id                             object
precipitation_mm                   float64
grid                                object
period                              object
temperature_c                      float64
dtype: object
Comprehensive data quality report saved to C:\Users\Administrator\NEWPROJECT\data
_quality_assessment\data_quality_report.json

Data Quality Assessment Highlights:

Data Completeness:
Total Records: 34673

Columns with Missingness:
flow: 38.59% missing
extra_x: 100.0% missing
rainfall: 24.45% missing
extra_y: 25.74% missing
water_year: 99.53% missing
time: 99.53% missing
stage_(m): 14.71% missing
flow_(m3/s): 14.71% missing
rating: 99.54% missing
datetime: 99.53% missing
river_level: 98.71% missing
river_timestamp: 98.71% missing
rainfall_timestamp: 98.71% missing
location_name: 98.71% missing
river_station_id: 98.71% missing
rainfall_station_id: 98.71% missing
month: 99.69% missing
station: 99.69% missing
grid_id: 99.69% missing
precipitation_mm: 99.79% missing
grid: 99.69% missing
period: 99.69% missing
temperature_c: 99.79% missing
```

```
Outliers Analysis:
flow:
  Outliers: 1967 (5.67%)
  Bounds: [-3.0975000000000006, 7.810500000000001]
extra_x:
  Outliers: 0 (0.0%)
  Bounds: [nan, nan]
rainfall:
  Outliers: 1153 (3.33%)
  Bounds: [-9.600000000000001, 16.0]
extra_y:
  Outliers: 0 (0.0%)
  Bounds: [500.0, 4500.0]
stage_(m):
  Outliers: 2311 (6.67%)
  Bounds: [1.0875205632907639, 1.7896701479012478]
flow_(m3/s):
  Outliers: 537 (1.55%)
  Bounds: [-54.13008888539218, 221.03616006377894]
river_level:
  Outliers: 0 (0.0%)
  Bounds: [-0.9227500000000002, 2.24325]
river_station_id:
  Outliers: 0 (0.0%)
  Bounds: [689635.0, 691035.0]
rainfall_station_id:
  Outliers: 0 (0.0%)
  Bounds: [559544.5, 565060.5]
precipitation_mm:
  Outliers: 4 (0.01%)
  Bounds: [42.5, 152.5]
temperature_c:
  Outliers: 0 (0.0%)
  Bounds: [-6.674999999999999, 25.525]

Temporal Analysis:
{
  "date": {
    "total_records": 34673,
    "date_range": {
      "start": "1941-10-24 00:00:00",
      "end": "2023-09-30 00:00:00",
      "total_duration": "29926 days 00:00:00"
    },
    "time_gaps": {
      "total_gaps": 19,
      "max_gap": "644 days 00:00:00",
      "mean_gap": "368 days 21:28:25.263157896"
    },
    "unique_dates": 22937
  },
  "time": {
    "total_records": 34673,
    "date_range": {
      "start": "2025-02-01 00:00:00",
      "end": "2025-02-01 23:45:00",
      "total_duration": "0 days 23:45:00"
    },
    "time_gaps": {
```

```
        "total_gaps": 0,
        "max_gap": null,
        "mean_gap": null
      },
      "unique_dates": 71
    },
    "datetime": {
      "total_records": 34673,
      "date_range": {
        "start": "1941-10-24 00:00:00",
        "end": "2023-07-23 12:15:00",
        "total_duration": "29857 days 12:15:00"
      },
      "time_gaps": {
        "total_gaps": 104,
        "max_gap": "711 days 00:00:00",
        "mean_gap": "287 days 00:30:17.307692308"
      },
      "unique_dates": 158
    }
  }
```


```
Data quality report and visualizations saved.
```

In [3]:

```
import os
directory = r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model'
print(os.listdir(directory))
```

```
['unified_data_model.csv']
```

In [11]:

```
import os
import glob

# Find all CSV files in the directory
csv_files = glob.glob(os.path.join(r'C:\Users\Administrator\NEWPROJECT\processed
print("CSV files found:", csv_files)
```

```
CSV files found: ['C:\\Users\\Administrator\\NEWPROJECT\\processed_data\\unified_
model\\unified_data_model.csv']
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
import os
import warnings

class ComprehensiveMissingDataHandler:
    def __init__(self, file_path):
        """
        Initialize the comprehensive missing data handler.

        Args:
            file_path (str): Path to the unified data model CSV
        """
        # Suppress specific warnings
```

```
        warnings.filterwarnings('ignore', category=FutureWarning)
        warnings.filterwarnings('ignore', category=UserWarning)

        # Load the unified data
        try:
            # Read CSV with more robust parameters
            self.original_data = pd.read_csv(file_path, low_memory=False, parse_
            print("Successfully loaded data!")

            # Display initial data information
            print(f"Data shape: {self.original_data.shape}")

        except Exception as e:
            print(f"Error reading CSV: {e}")
            raise

        self.processed_data = None
        self.missingness_report = None

    def analyze_missingness(self):
        """
        Comprehensive analysis of missing data.

        Returns:
            dict: Detailed missingness report
        """
        # Initialize missingness report
        missingness_report = {}

        # Analyze missingness for each column
        for column in self.original_data.columns:
            # Calculate missing details
            missing_percentage = self.original_data[column].isna().mean() * 100
            missing_count = self.original_data[column].isna().sum()
            total_records = len(self.original_data)

            # Store missingness information
            missingness_report[column] = {
                'missing_percentage': round(missing_percentage, 2),
                'missing_count': missing_count,
                'total_records': total_records,
                'data_type': str(self.original_data[column].dtype)
            }

        # Sort columns by missingness percentage
        sorted_missingness = dict(sorted(
            missingness_report.items(),
            key=lambda x: x[1]['missing_percentage'],
            reverse=True
        ))

        # Visualize missingness
        plt.figure(figsize=(20, 8))

        # Create bar plot of missingness
        missing_series = pd.Series({
            col: data['missing_percentage']
            for col, data in sorted_missingness.items()
        })
```

```
        missing_series.plot(kind='bar')
        plt.title('Percentage of Missing Data Across Columns', fontsize=16)
        plt.xlabel('Columns', fontsize=12)
        plt.ylabel('Missing Percentage', fontsize=12)
        plt.xticks(rotation=90, ha='right', fontsize=8)
        plt.tight_layout()

        # Save missingness plot
        try:
            plt.savefig('comprehensive_missingness_analysis.png', dpi=300, bbox_
            print("\nComprehensive missingness analysis plot saved as 'comprehen
        except Exception as e:
            print(f"Error saving missingness plot: {e}")
        plt.close()

        # Print detailed missingness report
        print("\nDetailed Missingness Report:")
        for column, details in sorted_missingness.items():
            if details['missing_percentage'] > 0:
                print(f"{column} (Type: {details['data_type']}):")
                print(f"  Missing: {details['missing_percentage']}%")
                print(f"  {details['missing_count']} out of {details['total_reco

        # Store missingness report
        self.missingness_report = sorted_missingness

        return sorted_missingness

    def prepare_data_for_imputation(self):
        """
        Prepare data for advanced imputation.

        Returns:
            pd.DataFrame: Prepared DataFrame
        """
        # Create a copy of the original data
        df = self.original_data.copy()

        # Identify column types
        numeric_columns = df.select_dtypes(include=[np.number]).columns
        categorical_columns = df.select_dtypes(include=['object']).columns
        datetime_columns = df.select_dtypes(include=['datetime64']).columns

        # Handle datetime columns
        for col in datetime_columns:
            # Extract datetime features
            df[f'{col}_year'] = df[col].dt.year
            df[f'{col}_month'] = df[col].dt.month
            df[f'{col}_day'] = df[col].dt.day
            df[f'{col}_weekday'] = df[col].dt.weekday

        # Handle categorical columns
        label_encoders = {}
        for col in categorical_columns:
            # Label encode categorical columns
            le = LabelEncoder()
            df[f'{col}_encoded'] = le.fit_transform(df[col].fillna('Missing'))
            label_encoders[col] = le

        return df, label_encoders
```

```
    def advanced_imputation(self):
        """
        Advanced imputation using multiple strategies.

        Returns:
            pd.DataFrame: Imputed DataFrame
        """
        # Prepare data for imputation
        prepared_df, label_encoders = self.prepare_data_for_imputation()

        # Identify numeric columns for imputation
        numeric_columns = [
            col for col in prepared_df.columns
            if prepared_df[col].dtype in ['int64', 'float64']
        ]

        # Iterative imputation with RandomForestRegressor
        imputer = IterativeImputer(
            estimator=RandomForestRegressor(n_estimators=100, random_state=42),
            max_iter=10,
            random_state=42
        )

        # Perform imputation
        imputed_data = imputer.fit_transform(prepared_df[numeric_columns])

        # Replace original columns with imputed values
        for i, col in enumerate(numeric_columns):
            prepared_df[col] = imputed_data[:, i]

        return prepared_df, label_encoders

    def feature_engineering(self, df):
        """
        Create additional contextual features.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: DataFrame with engineered features
        """
        # Create copy of DataFrame
        engineered_df = df.copy()

        # Rolling window features for numeric columns
        numeric_columns = df.select_dtypes(include=[np.number]).columns

        for column in numeric_columns:
            # 7-day rolling statistics
            engineered_df[f'{column}_7day_mean'] = df[column].rolling(window=7,
            engineered_df[f'{column}_7day_std'] = df[column].rolling(window=7, m

            # Percentage change
            engineered_df[f'{column}_pct_change'] = df[column].pct_change()

        return engineered_df

    def normalize_features(self, df):
```

```
        """
        Normalize features to consistent scale.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: Normalized DataFrame
        """
        # Identify numeric columns
        numeric_columns = df.select_dtypes(include=[np.number]).columns

        # Create scaler
        scaler = StandardScaler()

        # Normalize numeric columns
        df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

        return df

    def process_data(self):
        """
        Complete data processing pipeline.

        Returns:
            pd.DataFrame: Fully processed DataFrame
        """
        # 1. Analyze Missingness
        self.analyze_missingness()

        # 2. Advanced Imputation
        imputed_data, label_encoders = self.advanced_imputation()

        # 3. Feature Engineering
        engineered_data = self.feature_engineering(imputed_data)

        # 4. Normalize Features
        normalized_data = self.normalize_features(engineered_data)

        # Store processed data
        self.processed_data = normalized_data

        return normalized_data

    def export_processed_data(self, output_path, include_original_columns=False)
        """
        Export processed data to CSV.

        Args:
            output_path (str): Path to save processed data
            include_original_columns (bool): Whether to include original columns
        """
        # Ensure output directory exists
        os.makedirs(os.path.dirname(output_path), exist_ok=True)

        # Prepare export DataFrame
        if include_original_columns:
            export_df = pd.concat([self.original_data, self.processed_data], axi
        else:
            export_df = self.processed_data
```

```
        # Export processed data
        export_df.to_csv(output_path, index=False)
        print(f"Processed data exported to {output_path}")

        # Additional summary
        print("\nProcessed Data Overview:")
        print(f"Total Rows: {len(export_df)}")
        print(f"Total Columns: {len(export_df.columns)}")

def main():
    # Specific file path
    file_path = r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model

    # Output paths
    output_paths = [
        r'C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\process
        r'C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\process
    ]

    # Ensure output directories exist
    for path in output_paths:
        os.makedirs(os.path.dirname(path), exist_ok=True)

    # Initialize handler
    handler = ComprehensiveMissingDataHandler(file_path)

    try:
        # Process data
        processed_data = handler.process_data()

        # Export processed data
        # First, export only processed columns
        handler.export_processed_data(output_paths[0], include_original_columns=

        # Then, export with original columns included
        handler.export_processed_data(output_paths[1], include_original_columns=

    except Exception as e:
        print(f"An error occurred during data processing: {e}")
        import traceback
        traceback.print_exc()

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

```
Successfully loaded data!
Data shape: (34673, 26)

Comprehensive missingness analysis plot saved as 'comprehensive_missingness_analy
sis.png'

Detailed Missingness Report:
extra_x (Type: float64):
  Missing: 100.0%
  34673 out of 34673 records
precipitation_mm (Type: float64):
  Missing: 99.79%
  34601 out of 34673 records
temperature_c (Type: float64):
  Missing: 99.79%
  34601 out of 34673 records
month (Type: object):
  Missing: 99.69%
  34565 out of 34673 records
station (Type: object):
  Missing: 99.69%
  34565 out of 34673 records
grid_id (Type: object):
  Missing: 99.69%
  34565 out of 34673 records
grid (Type: object):
  Missing: 99.69%
  34565 out of 34673 records
period (Type: object):
  Missing: 99.69%
  34565 out of 34673 records
rating (Type: object):
  Missing: 99.54%
  34512 out of 34673 records
water_year (Type: object):
  Missing: 99.53%
  34509 out of 34673 records
time (Type: object):
  Missing: 99.53%
  34509 out of 34673 records
datetime (Type: object):
  Missing: 99.53%
  34509 out of 34673 records
river_level (Type: float64):
  Missing: 98.71%
  34226 out of 34673 records
river_timestamp (Type: object):
  Missing: 98.71%
  34226 out of 34673 records
rainfall_timestamp (Type: object):
  Missing: 98.71%
  34226 out of 34673 records
location_name (Type: object):
  Missing: 98.71%
  34226 out of 34673 records
river_station_id (Type: float64):
  Missing: 98.71%
  34226 out of 34673 records
rainfall_station_id (Type: float64):
  Missing: 98.71%
```

```
  34226 out of 34673 records
flow (Type: float64):
  Missing: 38.59%
  13380 out of 34673 records
extra_y (Type: float64):
  Missing: 25.74%
  8925 out of 34673 records
rainfall (Type: float64):
  Missing: 24.45%
  8478 out of 34673 records
stage_(m) (Type: float64):
  Missing: 14.71%
  5101 out of 34673 records
flow_(m3/s) (Type: float64):
  Missing: 14.71%
  5101 out of 34673 records
date (Type: object):
  Missing: 1.6%
  555 out of 34673 records
location (Type: object):
  Missing: 1.6%
  555 out of 34673 records
```

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import os
import warnings

class RobustFastImputation:
    def __init__(self, file_path, sample_size=5000):
        """
        Initialize the robust fast imputation handler.

        Args:
            file_path (str): Path to the unified data model CSV
            sample_size (int): Number of rows to sample
        """
        # Suppress warnings
        warnings.filterwarnings('ignore', category=FutureWarning)
        warnings.filterwarnings('ignore', category=UserWarning)

        # Load the data with sampling
        try:
            # Read full dataset first to understand overall structure
            full_df = pd.read_csv(file_path, low_memory=False, parse_dates=['dat

            # Sample the data
            self.original_data = full_df.sample(n=min(sample_size, len(full_df))

            print("Successfully loaded sampled data!")
            print(f"Original data shape: {full_df.shape}")
            print(f"Sampled data shape: {self.original_data.shape}")

        except Exception as e:
```

```
            print(f"Error reading CSV: {e}")
            raise

        self.processed_data = None

    def analyze_missingness(self):
        """
        Quick missingness analysis.

        Returns:
            dict: Missingness report
        """
        missingness_report = {}

        for column in self.original_data.columns:
            missing_percentage = self.original_data[column].isna().mean() * 100

            missingness_report[column] = {
                'missing_percentage': round(missing_percentage, 2),
                'missing_count': self.original_data[column].isna().sum(),
                'total_records': len(self.original_data)
            }

        # Visualize missingness
        plt.figure(figsize=(15, 6))
        missing_series = pd.Series({
            col: data['missing_percentage']
            for col, data in missingness_report.items()
        }).sort_values(ascending=False)

        # Filter out zero-missing columns
        missing_series = missing_series[missing_series > 0]

        missing_series.plot(kind='bar')
        plt.title('Percentage of Missing Data Across Columns')
        plt.xlabel('Columns')
        plt.ylabel('Missing Percentage')
        plt.xticks(rotation=90)
        plt.tight_layout()
        plt.savefig('fast_missingness_analysis.png')
        plt.close()

        # Print missingness details
        print("\nMissingness Report:")
        for column, details in sorted(
            missingness_report.items(),
            key=lambda x: x[1]['missing_percentage'],
            reverse=True
        ):
            if details['missing_percentage'] > 0:
                print(f"{column}: {details['missing_percentage']}% missing")

        return missingness_report

    def fast_imputation(self):
        """
        Efficient imputation method.

        Returns:
            pd.DataFrame: Imputed DataFrame
```

```
        """
        # Create a copy of the data
        df = self.original_data.copy()

        # Drop extra_x if it exists
        if 'extra_x' in df.columns:
            df = df.drop(columns=['extra_x'])

        # Separate column types
        numeric_columns = df.select_dtypes(include=[np.number]).columns
        categorical_columns = df.select_dtypes(include=['object']).columns

        # Impute numeric columns
        numeric_imputer = SimpleImputer(strategy='median')
        df[numeric_columns] = numeric_imputer.fit_transform(df[numeric_columns])

        # Impute categorical columns
        for col in categorical_columns:
            df[col] = df[col].fillna(df[col].mode()[0])

        # Feature engineering
        df = self.feature_engineering(df)

        return df

    def feature_engineering(self, df):
        """
        Create additional contextual features.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: DataFrame with engineered features
        """
        # Create copy of DataFrame
        engineered_df = df.copy()

        # Extract datetime features if 'date' exists
        if 'date' in df.columns:
            engineered_df['year'] = df['date'].dt.year
            engineered_df['month'] = df['date'].dt.month
            engineered_df['day'] = df['date'].dt.day

        # Rolling window features for numeric columns
        numeric_columns = df.select_dtypes(include=[np.number]).columns

        # Adaptive window size
        window_size = max(3, len(df) // 50)

        for column in numeric_columns:
            # Rolling statistics
            engineered_df[f'{column}_rolling_mean'] = df[column].rolling(window=
            engineered_df[f'{column}_rolling_std'] = df[column].rolling(window=w

        # Weather interaction features
        if 'precipitation_mm' in df.columns and 'temperature_c' in df.columns:
            engineered_df['precip_temp_interaction'] = df['precipitation_mm'] *

        return engineered_df
```

```
    def normalize_features(self, df):
        """
        Normalize numeric features.

        Args:
            df (pd.DataFrame): Input DataFrame

        Returns:
            pd.DataFrame: Normalized DataFrame
        """
        # Identify numeric columns
        numeric_columns = df.select_dtypes(include=[np.number]).columns

        # Create scaler
        scaler = StandardScaler()

        # Normalize numeric columns
        df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

        return df

    def process_data(self):
        """
        Complete data processing pipeline.

        Returns:
            pd.DataFrame: Processed DataFrame
        """
        # 1. Analyze Missingness
        self.analyze_missingness()

        # 2. Fast Imputation
        imputed_data = self.fast_imputation()

        # 3. Normalize Features
        normalized_data = self.normalize_features(imputed_data)

        # Store processed data
        self.processed_data = normalized_data

        return normalized_data

    def export_processed_data(self, output_path):
        """
        Export processed data to CSV.

        Args:
            output_path (str): Path to save processed data
        """
        # Use a default path if none provided
        if not output_path:
            output_path = os.path.join(
                os.getcwd(),
                'fast_processed_data.csv'
            )

        # Ensure output directory exists
        os.makedirs(os.path.dirname(output_path) or os.getcwd(), exist_ok=True)
```

```
        # Export processed data
        self.processed_data.to_csv(output_path, index=False)
        print(f"Processed data exported to {output_path}")

        # Additional summary
        print("\nProcessed Data Overview:")
        print(f"Total Rows: {len(self.processed_data)}")
        print(f"Total Columns: {len(self.processed_data.columns)}")

def run_fast_imputation(file_path, sample_size=5000):
    """
    Run fast imputation process.

    Args:
        file_path (str): Path to the CSV file
        sample_size (int): Number of rows to sample

    Returns:
        pd.DataFrame: Processed DataFrame
    """
    # Initialize handler
    try:
        handler = RobustFastImputation(file_path, sample_size)

        # Process data
        processed_data = handler.process_data()

        # Export processed data (use default path)
        handler.export_processed_data('')

        return processed_data

    except Exception as e:
        print(f"An error occurred: {e}")
        import traceback
        traceback.print_exc()
        return None

# Example usage in Jupyter Notebook:
# processed_df = run_fast_imputation(
#     r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\unified_d
#     sample_size=5000
# )
```

In [5]:

```
# Import the function
processed_df = run_fast_imputation(
    r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\unified_dat
    sample_size=5000  # Adjust based on your computational resources
)

# Quick exploration
print(processed_df.columns)
processed_df.describe()
```

```
Successfully loaded sampled data!
Original data shape: (34673, 26)
Sampled data shape: (5000, 26)

Missingness Report:
extra_x: 100.0% missing
precipitation_mm: 99.86% missing
temperature_c: 99.82% missing
month: 99.74% missing
station: 99.74% missing
grid_id: 99.74% missing
grid: 99.74% missing
period: 99.74% missing
water_year: 99.54% missing
time: 99.54% missing
rating: 99.54% missing
datetime: 99.54% missing
river_level: 98.54% missing
river_timestamp: 98.54% missing
rainfall_timestamp: 98.54% missing
location_name: 98.54% missing
river_station_id: 98.54% missing
rainfall_station_id: 98.54% missing
flow: 38.32% missing
extra_y: 25.98% missing
rainfall: 24.52% missing
stage_(m): 14.9% missing
flow_(m3/s): 14.9% missing
date: 1.72% missing
location: 1.72% missing
Processed data exported to C:\Users\Administrator\fast_processed_data.csv

Processed Data Overview:
Total Rows: 5000
Total Columns: 48
Index(['date', 'flow', 'rainfall', 'extra_y', 'water_year', 'time',
       'stage_(m)', 'flow_(m3/s)', 'rating', 'datetime', 'data_source',
       'location', 'river_level', 'river_timestamp', 'rainfall_timestamp',
       'location_name', 'river_station_id', 'rainfall_station_id', 'month',
       'station', 'grid_id', 'precipitation_mm', 'grid', 'period',
       'temperature_c', 'year', 'day', 'flow_rolling_mean', 'flow_rolling_std',
       'rainfall_rolling_mean', 'rainfall_rolling_std', 'extra_y_rolling_mean',
       'extra_y_rolling_std', 'stage_(m)_rolling_mean',
       'stage_(m)_rolling_std', 'flow_(m3/s)_rolling_mean',
       'flow_(m3/s)_rolling_std', 'river_level_rolling_mean',
       'river_level_rolling_std', 'river_station_id_rolling_mean',
       'river_station_id_rolling_std', 'rainfall_station_id_rolling_mean',
       'rainfall_station_id_rolling_std', 'precipitation_mm_rolling_mean',
       'precipitation_mm_rolling_std', 'temperature_c_rolling_mean',
       'temperature_c_rolling_std', 'precip_temp_interaction'],
      dtype='object')
```

Out[5]:

	date	flow	rainfall	extra_y	stage_(m)
count	4914	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03
mean	1997-08-12 00:52:44.835164800	2.557954e-17	-9.858780e-17	1.541878e-16	-7.901235e-16
min	1946-09-20 00:00:00	-7.245411e-01	-7.059268e-01	-2.299903e+00	-3.137345e+00
25%	1984-01-18 18:00:00	-3.678279e-01	-6.660302e-01	-3.318190e-01	-3.990677e-01
50%	2000-05-22 00:00:00	-2.632540e-01	-3.468572e-01	-3.318190e-01	-5.787116e-02
75%	2012-03-08 12:00:00	-1.148172e-01	4.311269e-01	-3.318190e-01	2.380691e-01
max	2023-09-20 00:00:00	1.623618e+01	7.712260e+00	3.604350e+00	1.869151e+01
std	NaN	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00

8 rows × 35 columns

Statistical Analysis

In [16]:

```
import os
import numpy as np
import pandas as pd
import warnings
from scipy import stats
import json

class StationStatisticalAnalyzer:
    def __init__(self, filepath):
        """
        Initialize statistical analyzer for flood monitoring data

        Parameters:
        - filepath (str): Path to preprocessed data file
        """
        # Suppress warnings
        warnings.filterwarnings('ignore', category=pd.errors.DtypeWarning)

        # Load the data
        self.data = self._load_data(filepath)

        # Identify stations
        self.stations = self._get_valid_stations()

    def _load_data(self, filepath):
        """
        Load and preprocess the data

        Parameters:
        - filepath (str): Path to the CSV file
```

```
        Returns:
        - pd.DataFrame: Preprocessed dataframe
        """
        # Potential file paths
        potential_paths = [
            filepath,
            r'C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\pro
            r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\uni
        ]

        for path in potential_paths:
            if not os.path.exists(path):
                print(f"File not found: {path}")
                continue

            try:
                # Read the CSV file
                df = pd.read_csv(path, low_memory=False)

                # Clean station information
                df['station_cleaned'] = self._clean_station_names(df)

                # Ensure required columns exist
                if 'river_level' not in df.columns:
                    print(f"Missing 'river_level' column in {path}")
                    continue

                # Clean and prepare data
                df = df.dropna(subset=['station_cleaned', 'river_level'])

                return df

            except Exception as e:
                print(f"Error loading file {path}: {e}")

        raise ValueError("No valid data file found")

    def _clean_station_names(self, df):
        """
        Clean and standardize station names

        Parameters:
        - df (pd.DataFrame): Input dataframe

        Returns:
        - pd.Series: Cleaned station names
        """
        # Identify potential station columns
        station_columns = ['station', 'location', 'location_name']

        # Combine station names
        def combine_station_name(row):
            for col in station_columns:
                if col in df.columns and pd.notna(row[col]):
                    return str(row[col]).strip()
            return 'UNKNOWN'

        # Clean station names
        station_names = df.apply(combine_station_name, axis=1)
```

```
        # Standardize names
        station_mapping = {
            'MANCHESTER RACECOURSE': 'Manchester Racecourse',
            'BURY MANCHESTER': 'Bury Manchester',
            'BURY GROUND': 'Bury Manchester',
            'ROCHDALE': 'Rochdale',
            'manchester': 'Manchester Racecourse',
            'bury': 'Bury Manchester',
            'rochdale': 'Rochdale'
        }

        return station_names.replace(station_mapping)

    def _get_valid_stations(self):
        """
        Get unique valid station names

        Returns:
        - list: List of valid station names
        """
        valid_stations = self.data['station_cleaned'].unique()
        valid_stations = [
            station for station in valid_stations
            if station and station not in ['UNKNOWN']
        ]
        return valid_stations

    def compute_statistical_parameters(self):
        """
        Compute comprehensive statistical parameters for each station

        Returns:
        - dict: Detailed statistical analysis for each station
        """
        station_stats = {}

        for station in self.stations:
            # Filter data for the specific station
            station_data = self.data[self.data['station_cleaned'] == station]['r

            # Ensure we have data
            if len(station_data) == 0:
                print(f"No data available for station: {station}")
                continue

            # 1. Basic Statistical Parameters
            basic_stats = {
                'mean': station_data.mean(),
                'median': station_data.median(),
                'std_dev': station_data.std(),
                'min': station_data.min(),
                'max': station_data.max()
            }

            # 2. Extreme Value Thresholds
            extreme_stats = {
                'lower_threshold_1std': basic_stats['mean'] - basic_stats['std_d
                'lower_threshold_2std': basic_stats['mean'] - (2 * basic_stats['
                'upper_threshold_1std': basic_stats['mean'] + basic_stats['std_d
```

```
                'upper_threshold_2std': basic_stats['mean'] + (2 * basic_stats['
                'percentile_5': np.percentile(station_data, 5),
                'percentile_95': np.percentile(station_data, 95)
            }

            # 3. Seasonal Variations
            seasonal_stats = self._compute_seasonal_variations(station)

            # 4. Additional Distributional Characteristics
            distributional_stats = {
                'skewness': stats.skew(station_data),
                'kurtosis': stats.kurtosis(station_data)
            }

            # Combine all statistics
            station_stats[station] = {
                'basic_stats': basic_stats,
                'extreme_thresholds': extreme_stats,
                'seasonal_variations': seasonal_stats,
                'distributional_characteristics': distributional_stats
            }

        return station_stats

    def _compute_seasonal_variations(self, station):
        """
        Compute seasonal variations for a specific station

        Parameters:
        - station (str): Station name

        Returns:
        - dict: Seasonal statistical variations
        """
        # Filter data for the specific station
        station_data = self.data[self.data['station_cleaned'] == station].copy()

        # Ensure we have necessary date information
        if 'month' not in station_data.columns:
            print(f"No month information available for {station}")
            return {}

        # Group by month and compute river level statistics
        seasonal_stats = station_data.groupby('month')['river_level'].agg([
            'mean',
            'median',
            'std',
            'min',
            'max'
        ]).to_dict()

        return seasonal_stats

def main():
    # Initialize the statistical analyzer
    analyzer = StationStatisticalAnalyzer('')

    # Compute and print statistical parameters
    station_statistics = analyzer.compute_statistical_parameters()
```

```
    # Create output directory
    output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data\statistical_
    os.makedirs(output_dir, exist_ok=True)

    # Save results for each station
    for station, stats in station_statistics.items():
        # Prepare output filename
        output_filename = os.path.join(output_dir, f'{station}_statistical_param

        # Save detailed statistics
        with open(output_filename, 'w') as f:
            json.dump(stats, f, indent=4)

        # Print summary to console
        print(f"\nStatistical Summary for {station}:")
        print("Basic Statistics:")
        for key, value in stats['basic_stats'].items():
            print(f"  {key}: {value:.4f}")

        print("\nExtreme Value Thresholds:")
        for key, value in stats['extreme_thresholds'].items():
            print(f"  {key}: {value:.4f}")

    print(f"\nDetailed statistical analysis saved in: {output_dir}")

# Ensure the script can be run
if __name__ == '__main__':
    main()
```

```
File not found:
File not found: C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\pr
ocessed_data_advanced.csv

Statistical Summary for Rochdale:
Basic Statistics:
  mean: 0.2507
  median: 0.2440
  std_dev: 0.0201
  min: 0.2270
  max: 0.2930

Extreme Value Thresholds:
  lower_threshold_1std: 0.2306
  lower_threshold_2std: 0.2105
  upper_threshold_1std: 0.2707
  upper_threshold_2std: 0.2908
  percentile_5: 0.2270
  percentile_95: 0.2900

Statistical Summary for Manchester Racecourse:
Basic Statistics:
  mean: 1.1044
  median: 1.0740
  std_dev: 0.0537
  min: 1.0450
  max: 1.2030

Extreme Value Thresholds:
  lower_threshold_1std: 1.0507
  lower_threshold_2std: 0.9970
  upper_threshold_1std: 1.1581
  upper_threshold_2std: 1.2119
  percentile_5: 1.0480
  percentile_95: 1.1972

Statistical Summary for Bury Ground:
Basic Statistics:
  mean: 0.3954
  median: 0.3890
  std_dev: 0.0197
  min: 0.3700
  max: 0.4410

Extreme Value Thresholds:
  lower_threshold_1std: 0.3757
  lower_threshold_2std: 0.3560
  upper_threshold_1std: 0.4151
  upper_threshold_2std: 0.4348
  percentile_5: 0.3710
  percentile_95: 0.4380

Detailed statistical analysis saved in: C:\Users\Administrator\NEWPROJECT\process
ed_data\statistical_analysis
```

FLOOD ANOMALY DETECTION

In [154…

```
import pandas as pd
import numpy as np
```

```
from typing import Dict, Any, Tuple

class FloodAnomalyDetector:
    def __init__(self, baseline_data: Dict[str, Any]):
        """
        Initialize the anomaly detector with historical baseline data.

        Args:
            baseline_data (dict): Dictionary containing statistical baselines fo
        """
        self.baseline_data = baseline_data

        # Define anomaly thresholds
        self.anomaly_thresholds = {
            'stage_meters': {
                'mild_lower': baseline_data['stage_meters_mean'] - 1.5 * baselin
                'mild_upper': baseline_data['stage_meters_mean'] + 1.5 * baselin
                'moderate_lower': baseline_data['stage_meters_mean'] - 2 * basel
                'moderate_upper': baseline_data['stage_meters_mean'] + 2 * basel
                'severe_lower': baseline_data['stage_meters_mean'] - 3 * baselin
                'severe_upper': baseline_data['stage_meters_mean'] + 3 * baselin
            },
            'peak_flow_cubic_meters': {
                'mild_lower': baseline_data['peak_flow_cubic_meters_mean'] - 1.5
                'mild_upper': baseline_data['peak_flow_cubic_meters_mean'] + 1.5
                'moderate_lower': baseline_data['peak_flow_cubic_meters_mean'] -
                'moderate_upper': baseline_data['peak_flow_cubic_meters_mean'] +
                'severe_lower': baseline_data['peak_flow_cubic_meters_mean'] - 3
                'severe_upper': baseline_data['peak_flow_cubic_meters_mean'] + 3
            }
        }

    def detect_anomaly(self, measurement: float, measurement_type: str) -> Dict[
        """
        Detect the anomaly level for a given measurement.

        Args:
            measurement (float): Current measurement value
            measurement_type (str): Type of measurement ('stage_meters' or 'peak

        Returns:
            dict: Anomaly detection results
        """
        if measurement_type not in self.anomaly_thresholds:
            raise ValueError(f"Unsupported measurement type: {measurement_type}"

        thresholds = self.anomaly_thresholds[measurement_type]

        # Determine anomaly severity
        anomaly_level = 'normal'
        anomaly_details = {
            'value': measurement,
            'baseline_mean': self.baseline_data[f'{measurement_type}_mean'],
            'baseline_std': self.baseline_data[f'{measurement_type}_std']
        }

        # Check for severe anomalies
        if (measurement <= thresholds['severe_lower'] or
            measurement >= thresholds['severe_upper']):
            anomaly_level = 'severe'
```

```
            anomaly_details['description'] = (
                'Severe anomaly - Extreme deviation from historical patterns'
            )

        # Check for moderate anomalies
        elif (measurement <= thresholds['moderate_lower'] or
              measurement >= thresholds['moderate_upper']):
            anomaly_level = 'moderate'
            anomaly_details['description'] = (
                'Moderate anomaly - Significant deviation from historical patter
            )

        # Check for mild anomalies
        elif (measurement <= thresholds['mild_lower'] or
              measurement >= thresholds['mild_upper']):
            anomaly_level = 'mild'
            anomaly_details['description'] = (
                'Mild anomaly - Slight deviation from historical patterns'
            )

        return {
            'anomaly_level': anomaly_level,
            'anomaly_details': anomaly_details
        }

    def generate_flood_risk_assessment(self, anomaly_results: Dict[str, Any]) ->
        """
        Generate a flood risk assessment based on anomaly detection results.

        Args:
            anomaly_results (dict): Anomaly detection results

        Returns:
            str: Flood risk assessment description
        """
        anomaly_level = anomaly_results['anomaly_level']
        details = anomaly_results['anomaly_details']

        risk_assessments = {
            'normal': "Current river conditions appear to be within normal histo
            'mild': (
                "Mild river condition anomaly detected. "
                "While not an immediate threat, "
                f"the current measurement of {details['value']:.2f} "
                f"deviates from the historical mean of {details['baseline_mean']
                "Continued monitoring is recommended."
            ),
            'moderate': (
                "MODERATE FLOOD RISK DETECTED! "
                f"Significant deviation observed with current measurement of {de
                f"compared to historical mean of {details['baseline_mean']:.2f}.
                "Immediate precautionary measures are advised. Local authorities
            ),
            'severe': (
                "SEVERE FLOOD RISK ALERT! "
                f"EXTREME deviation with current measurement of {details['value'
                f"far from historical mean of {details['baseline_mean']:.2f}. "
                "URGENT ACTION REQUIRED. Immediate evacuation and emergency resp
            )
        }
```

```
        return risk_assessments.get(anomaly_level, risk_assessments['normal'])

    def calculate_z_score(self, measurement: float, measurement_type: str) -> fl
        """
        Calculate the Z-score for a given measurement.

        Args:
            measurement (float): Current measurement value
            measurement_type (str): Type of measurement

        Returns:
            float: Z-score of the measurement
        """
        mean = self.baseline_data[f'{measurement_type}_mean']
        std = self.baseline_data[f'{measurement_type}_std']

        return (measurement - mean) / std

def main():
    # Example baseline data for Manchester (from previous output)
    manchester_baseline = {
        'stage_meters_mean': 3.513146341463415,
        'stage_meters_std': 0.5900609745004537,
        'peak_flow_cubic_meters_mean': 279.4348414634146,
        'peak_flow_cubic_meters_std': 87.35713783912391
    }

    # Initialize the anomaly detector
    detector = FloodAnomalyDetector(manchester_baseline)

    # Example scenario: current river measurements
    test_scenarios = [
        {'measurement': 4.5, 'type': 'stage_meters'},
        {'measurement': 450.0, 'type': 'peak_flow_cubic_meters'}
    ]

    # Analyze each scenario
    for scenario in test_scenarios:
        print(f"\nAnalyzing {scenario['type']} = {scenario['measurement']}")

        # Detect anomaly
        anomaly_result = detector.detect_anomaly(
            scenario['measurement'],
            scenario['type']
        )

        # Generate flood risk assessment
        risk_assessment = detector.generate_flood_risk_assessment(anomaly_result

        # Calculate Z-score
        z_score = detector.calculate_z_score(
            scenario['measurement'],
            scenario['type']
        )

        print(f"Anomaly Level: {anomaly_result['anomaly_level']}")
        print(f"Z-Score: {z_score:.2f}")
        print("Risk Assessment:", risk_assessment)
```

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

```
Analyzing stage_meters = 4.5
Anomaly Level: mild
Z-Score: 1.67
Risk Assessment: Mild river condition anomaly detected. While not an immediate th
reat, the current measurement of 4.50 deviates from the historical mean of 3.51.
Continued monitoring is recommended.

Analyzing peak_flow_cubic_meters = 450.0
Anomaly Level: mild
Z-Score: 1.95
Risk Assessment: Mild river condition anomaly detected. While not an immediate th
reat, the current measurement of 450.00 deviates from the historical mean of 279.
43. Continued monitoring is recommended.
```

Anomaly Detection

Z-score method: No anomalies detected Machine Learning methods: Identified potential unusual river levels

In [14]:
```
import os
import numpy as np
import pandas as pd
import warnings
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from statsmodels.tsa.seasonal import seasonal_decompose

class AnomalyDetector:
    def __init__(self, data):
        """
        Initialize anomaly detector with preprocessed data

        Parameters:
        - data (pd.DataFrame): Preprocessed time series data
        """
        # Clean and prepare data
        self.data = self._preprocess_data(data)

        # Get valid stations (removing NaN and empty strings)
        self.stations = self._get_valid_stations()

    def _preprocess_data(self, data):
        """
        Preprocess and clean the input data

        Parameters:
        - data (pd.DataFrame): Input dataframe

        Returns:
        - pd.DataFrame: Cleaned and prepared dataframe
        """
        # Suppress warnings during processing
        warnings.filterwarnings('ignore', category=pd.errors.DtypeWarning)
```

```
        # Create a copy to avoid modifying original data
        df = data.copy()

        # Identify and clean station information
        station_columns = ['station', 'location', 'location_name']

        # Combine station information
        def combine_station_name(row):
            # Priority order for station names
            for col in station_columns:
                if col in df.columns and pd.notna(row[col]):
                    return str(row[col]).strip()
            return 'UNKNOWN'

        # Add a new 'station_cleaned' column
        df['station_cleaned'] = df.apply(combine_station_name, axis=1)

        # Standardize station names
        station_mapping = {
            'MANCHESTER RACECOURSE': 'Manchester Racecourse',
            'BURY MANCHESTER': 'Bury Manchester',
            'BURY GROUND': 'Bury Manchester',
            'ROCHDALE': 'Rochdale',
            'manchester': 'Manchester Racecourse',
            'bury': 'Bury Manchester',
            'rochdale': 'Rochdale',
            'nan': 'UNKNOWN'
        }
        df['station_cleaned'] = df['station_cleaned'].replace(station_mapping)

        # Remove rows with 'UNKNOWN' stations if possible
        df = df[df['station_cleaned'] != 'UNKNOWN']

        # Remove rows with NaN in critical columns
        df.dropna(subset=['river_level'], inplace=True)

        return df

    def _get_valid_stations(self):
        """
        Get valid station names, filtering out empty or problematic entries

        Returns:
        - list: List of valid station names
        """
        # Get unique stations, removing NaN
        valid_stations = self.data['station_cleaned'].dropna().unique()

        # Convert to list and remove any remaining empty strings
        valid_stations = [
            str(station).strip() for station in valid_stations
            if station and str(station).strip() not in ['', 'nan', 'UNKNOWN']
        ]

        # Print detailed debugging information
        print("\nStation Detection Debug:")
        print("Raw Station Data:", self.data['station_cleaned'].unique())
        print("Processed Valid Stations:", valid_stations)

        return valid_stations
```

```
    def z_score_detection(self, station, column='river_level', threshold=3):
        """
        Perform Z-score based anomaly detection

        Parameters:
        - station (str): Monitoring station name
        - column (str): Column to analyze
        - threshold (float): Z-score threshold for anomalies

        Returns:
        - pd.Series: Boolean mask of anomalies
        """
        station_data = self.data[self.data['station_cleaned'] == station][column

        if len(station_data) == 0:
            print(f"No data available for station: {station}")
            return pd.Series([], dtype=bool)

        z_scores = np.abs(stats.zscore(station_data))
        return z_scores > threshold

    def machine_learning_anomaly_detection(self, station, column='river_level'):
        """
        Apply machine learning techniques for anomaly detection

        Techniques:
        1. Isolation Forest
        2. Elliptic Envelope (assumes gaussian distribution)

        Parameters:
        - station (str): Monitoring station name
        - column (str): Column to analyze

        Returns:
        - dict: Anomaly detection results from different methods
        """
        station_data = self.data[self.data['station_cleaned'] == station][column

        if len(station_data) < 2:
            print(f"Insufficient data for ML anomaly detection at station: {stat
            return {
                'isolation_forest': np.array([]),
                'elliptic_envelope': np.array([])
            }

        # Reshape and scale data
        X = station_data.values.reshape(-1, 1)
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # Isolation Forest
        iso_forest = IsolationForest(contamination=0.1, random_state=42)
        iso_forest_anomalies = iso_forest.fit_predict(X_scaled)

        # Elliptic Envelope (assumes gaussian distribution)
        elliptic_env = EllipticEnvelope(contamination=0.1, random_state=42)
        elliptic_anomalies = elliptic_env.fit_predict(X_scaled)

        return {
```

```
            'isolation_forest': iso_forest_anomalies == -1,
            'elliptic_envelope': elliptic_anomalies == -1
        }

def load_preprocessed_data(filepath=''):
    """
    Load preprocessed data from a CSV file with robust error handling

    Parameters:
    - filepath (str): Path to preprocessed data file

    Returns:
    - pd.DataFrame: Preprocessed dataset
    """
    # List of potential filepaths
    potential_paths = [
        filepath,
        r'C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\process
        r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\unified
    ]

    for path in potential_paths:
        # Check if the file exists
        if not os.path.exists(path):
            print(f"File not found: {path}")
            continue

        try:
            # Read the CSV file with additional error handling
            df = pd.read_csv(path, low_memory=False)

            # Validate required columns
            required_columns = ['river_level']
            missing_columns = [col for col in required_columns if col not in df.

            if missing_columns:
                print(f"Error: Missing required columns in {path}: {missing_colu
                continue

            # Basic data validation
            print("Data Loaded Successfully:")
            print(f"File: {path}")
            print(f"Total Rows: {len(df)}")

            # Additional column information
            station_cols = ['station', 'location', 'location_name']
            for col in station_cols:
                if col in df.columns:
                    print(f"Stations in {col}: {df[col].unique()}")

            return df

        except pd.errors.EmptyDataError:
            print(f"Error: The file at {path} is empty.")
        except pd.errors.ParserError:
            print(f"Error: Unable to parse the CSV file at {path}.")
        except Exception as e:
            print(f"Unexpected error loading file {path}: {e}")

    print("No valid data file found.")
```

```
    return None

def main():
    # Attempt to load data
    data = load_preprocessed_data()

    if data is not None:
        # Initialize anomaly detector
        anomaly_detector = AnomalyDetector(data)

        # Print valid stations
        print("\nValid Stations:", anomaly_detector.stations)

        # Perform anomaly detection for each station
        for station in anomaly_detector.stations:
            print(f"\nAnomaly Detection Results for {station}:")

            try:
                # Z-score detection
                z_score_anomalies = anomaly_detector.z_score_detection(station)
                print("Z-score Anomalies:", z_score_anomalies.sum())

                # Machine learning anomalies
                ml_anomalies = anomaly_detector.machine_learning_anomaly_detecti
                print("Isolation Forest Anomalies:",
                      ml_anomalies['isolation_forest'].sum() if len(ml_anomalies
                print("Elliptic Envelope Anomalies:",
                      ml_anomalies['elliptic_envelope'].sum() if len(ml_anomalie

            except Exception as e:
                print(f"Error processing station {station}: {e}")

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

```
File not found:
File not found: C:\Users\Administrator\NEWPROJECT\processed_data\flood_anomaly\pr
ocessed_data_advanced.csv
Data Loaded Successfully:
File: C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\unified_data
_model.csv
Total Rows: 34673
Stations in station: [nan 'MANCHESTER RACECOURSE' 'ROCHDALE' 'BURY MANCHESTER']
Stations in location: ['manchester' 'bury' 'rochdale' nan]
Stations in location_name: [nan 'Rochdale' 'Manchester Racecourse' 'Bury Ground']

Station Detection Debug:
Raw Station Data: ['Rochdale' 'Manchester Racecourse' 'Bury Ground']
Processed Valid Stations: ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

Valid Stations: ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

Anomaly Detection Results for Rochdale:
Z-score Anomalies: 0
Isolation Forest Anomalies: 15
Elliptic Envelope Anomalies: 15

Anomaly Detection Results for Manchester Racecourse:
Z-score Anomalies: 0
Isolation Forest Anomalies: 15
Elliptic Envelope Anomalies: 15

Anomaly Detection Results for Bury Ground:
Z-score Anomalies: 0
Isolation Forest Anomalies: 12
Elliptic Envelope Anomalies: 15
```

Multi-Level Anomaly Detection

1. Implement Detection Techniques Z-score based detection (already partially implemented) Machine learning anomaly detection methods Time series specific approaches
2. Risk Classification Framework Define anomaly levels: Mild Anomalies Moderate Risks Severe Flood Potentials develop adaptive thresholding Incorporate contextual and seasonal variations

In [19]:

```python
import os
import numpy as np
import pandas as pd
import warnings

def preprocess_data(filepath):
    """
    Preprocess the input data, handling missing values and cleaning station name

    Parameters:
    - filepath (str): Path to the input CSV file

    Returns:
    - pd.DataFrame: Preprocessed dataframe
    """
    # Suppress warnings
```

```
    warnings.filterwarnings('ignore', category=pd.errors.DtypeWarning)

    # Potential file paths
    potential_paths = [
        filepath,
        r'C:\Users\Administrator\NEWPROJECT\processed_data\unified_model\unified
    ]

    for path in potential_paths:
        if not os.path.exists(path):
            print(f"File not found: {path}")
            continue

        try:
            # Read the CSV file
            df = pd.read_csv(path, low_memory=False)

            # Print initial data info
            print("\nInitial Data Information:")
            print(f"Total Rows: {len(df)}")
            print("\nMissing Values:")
            print(df.isnull().sum())

            # Identify station columns
            station_columns = ['station', 'location', 'location_name']

            # Clean station names
            def clean_station_name(row):
                for col in station_columns:
                    if col in df.columns and pd.notna(row[col]):
                        return str(row[col]).strip()
                return 'UNKNOWN'

            # Add cleaned station column
            df['station_cleaned'] = df.apply(clean_station_name, axis=1)

            # Standardize station names
            station_mapping = {
                'MANCHESTER RACECOURSE': 'Manchester Racecourse',
                'BURY MANCHESTER': 'Bury Manchester',
                'BURY GROUND': 'Bury Manchester',
                'ROCHDALE': 'Rochdale',
                'manchester': 'Manchester Racecourse',
                'bury': 'Bury Manchester',
                'rochdale': 'Rochdale'
            }
            df['station_cleaned'] = df['station_cleaned'].replace(station_mappin

            # Handle missing river level data
            print("\nRiver Level Missing Values:")
            print(df['river_level'].isnull().sum())

            # Remove rows with missing river levels
            df_cleaned = df.dropna(subset=['river_level', 'station_cleaned'])

            print("\nCleaned Data Information:")
            print(f"Rows after cleaning: {len(df_cleaned)}")
            print("Stations:", df_cleaned['station_cleaned'].unique())

            return df_cleaned
```

```
        except Exception as e:
            print(f"Error processing file {path}: {e}")

    raise ValueError("No valid data file found")

def main():
    # Preprocess the data
    try:
        cleaned_data = preprocess_data('')

        # Optionally, save the cleaned data
        output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data\cleaned_
        os.makedirs(output_dir, exist_ok=True)
        output_path = os.path.join(output_dir, 'cleaned_flood_data.csv')
        cleaned_data.to_csv(output_path, index=False)
        print(f"\nCleaned data saved to: {output_path}")

    except Exception as e:
        print(f"Preprocessing failed: {e}")

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

```
File not found: 

Initial Data Information:
Total Rows: 34673

Missing Values:
date                     555
flow                   13380
extra_x                34673
rainfall                8478
extra_y                 8925
water_year             34509
time                   34509
stage_(m)               5101
flow_(m3/s)             5101
rating                 34512
datetime               34509
data_source                0
location                 555
river_level            34226
river_timestamp        34226
rainfall_timestamp     34226
location_name          34226
river_station_id       34226
rainfall_station_id    34226
month                  34565
station                34565
grid_id                34565
precipitation_mm       34601
grid                   34565
period                 34565
temperature_c          34601
dtype: int64

River Level Missing Values:
34226

Cleaned Data Information:
Rows after cleaning: 447
Stations: ['Rochdale' 'Manchester Racecourse' 'Bury Ground']

Cleaned data saved to: C:\Users\Administrator\NEWPROJECT\processed_data\cleaned_d
ata\cleaned_flood_data.csv
```

In [5]:

```python
import pandas as pd
import numpy as np

class FloodDataIntegrator:
    def __init__(self, historical_path, realtime_path):
        """
        Initialize data integrator with paths to historical and real-time data
        """
        self.historical_path = historical_path
        self.realtime_path = realtime_path

        # Load all historical datasets
        self.historical_data = self._load_historical_data()

        # Load real-time data
        self.realtime_data = pd.read_csv(
```

```
            self.realtime_path,
            parse_dates=['river_timestamp']
        )

    def _load_historical_data(self):
        """
        Load and preprocess historical datasets
        """
        historical_datasets = {}

        # List of historical files to process
        historical_files = [
            'bury_daily_flow.csv',
            'bury_daily_rainfall.csv',
            'rochdale_daily_flow.csv',
            'rochdale_daily_rainfall.csv',
            'bury_peak_flow.csv',
            'rochdale_peak_flow.csv',
            'manchester_peak_flow.csv'
        ]

        for filename in historical_files:
            filepath = os.path.join(self.historical_path, filename)
            df = pd.read_csv(filepath, parse_dates=['Date'])

            # Standardize column names
            if 'Flow' in df.columns:
                df.rename(columns={'Flow': 'flow'}, inplace=True)
            if 'Rainfall' in df.columns:
                df.rename(columns={'Rainfall': 'rainfall'}, inplace=True)

            # Add source identifier
            df['data_source'] = filename

            historical_datasets[filename] = df

        return historical_datasets

    def merge_datasets(self):
        """
        Merge historical and real-time datasets
        """
        # Combine historical daily flow data
        flow_data = pd.concat([
            self.historical_data['bury_daily_flow.csv'],
            self.historical_data['rochdale_daily_flow.csv']
        ])

        # Combine historical rainfall data
        rainfall_data = pd.concat([
            self.historical_data['bury_daily_rainfall.csv'],
            self.historical_data['rochdale_daily_rainfall.csv']
        ])

        # Merge real-time data
        realtime_merged = self.realtime_data.copy()

        return {
            'flow_data': flow_data,
            'rainfall_data': rainfall_data,
```

```
            'realtime_data': realtime_merged
        }

    def create_comprehensive_dataset(self):
        """
        Create a unified dataset for flood analysis
        """
        merged = self.merge_datasets()

        # Comprehensive analysis dataset
        comprehensive_df = pd.DataFrame()

        return comprehensive_df

# Usage
integrator = FloodDataIntegrator(
    historical_path=r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\
    realtime_path=r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\re
)

# Merge datasets
integrated_data = integrator.merge_datasets()
```

In [6]:
```
import pandas as pd
import numpy as np

class AdvancedDataIntegrator:
    def __init__(self, historical_path):
        """
        Advanced data integration with temporal alignment
        """
        self.historical_path = historical_path
        self.datasets = self._load_all_datasets()

    def _load_all_datasets(self):
        """
        Load all datasets with comprehensive temporal information
        """
        datasets = {}

        # Mapping of files to expected processing
        file_mappings = {
            'bury_daily_flow.csv': {
                'type': 'daily_flow',
                'station': 'Bury',
                'start_date_handling': 'earliest',
                'interpolation_method': 'linear'
            },
            'bury_daily_rainfall.csv': {
                'type': 'daily_rainfall',
                'station': 'Bury',
                'start_date_handling': 'earliest',
                'interpolation_method': 'fill'
            },
            # Add similar mappings for other files
        }

        # Load and preprocess each dataset
        for filename, config in file_mappings.items():
            filepath = os.path.join(self.historical_path, filename)
```

```
            df = pd.read_csv(filepath, parse_dates=['Date'])

            # Standardize column names
            df.rename(columns={
                'Date': 'date',
                'Flow': 'value',
                'Rainfall': 'value'
            }, inplace=True)

            # Add metadata
            df['station'] = config['station']
            df['data_type'] = config['type']

            datasets[filename] = {
                'data': df,
                'config': config
            }

        return datasets

    def align_temporal_datasets(self, reference_period=None):
        """
        Align datasets with comprehensive temporal handling

        Parameters:
        - reference_period: Tuple of (start_date, end_date)
                            If None, use the most comprehensive dataset

        Returns:
        - Aligned and interpolated datasets
        """
        # Find the most comprehensive dataset as reference
        if not reference_period:
            # Identify dataset with longest time span
            longest_dataset = max(
                self.datasets.values(),
                key=lambda x: (x['data']['date'].max() - x['data']['date'].min()
            )

            reference_period = (
                longest_dataset['data']['date'].min(),
                longest_dataset['data']['date'].max()
            )

        # Create a complete date range
        complete_date_range = pd.date_range(
            start=reference_period[0],
            end=reference_period[1],
            freq='D'
        )

        # Store aligned datasets
        aligned_datasets = {}

        for filename, dataset_info in self.datasets.items():
            df = dataset_info['data']
            config = dataset_info['config']

            # Create base dataframe with complete date range
            base_df = pd.DataFrame(index=complete_date_range)
```

```
            base_df.index.name = 'date'
            base_df['station'] = config['station']
            base_df['data_type'] = config['type']

            # Merge with original data
            merged_df = base_df.merge(
                df.set_index('date'),
                left_index=True,
                right_index=True,
                how='left'
            )

            # Interpolation based on dataset type
            if config['interpolation_method'] == 'linear':
                merged_df['value'] = merged_df['value'].interpolate(method='line
            elif config['interpolation_method'] == 'fill':
                merged_df['value'] = merged_df['value'].fillna(method='ffill')

            # Handle extreme cases
            merged_df['value'].fillna(merged_df['value'].mean(), inplace=True)

            aligned_datasets[filename] = merged_df

        return aligned_datasets

    def compute_temporal_statistics(self, aligned_datasets):
        """
        Compute comprehensive temporal statistics

        Parameters:
        - aligned_datasets: Datasets aligned across common time period

        Returns:
        - Temporal analysis statistics
        """
        temporal_stats = {}

        for filename, df in aligned_datasets.items():
            temporal_stats[filename] = {
                'total_missing_values': df['value'].isnull().sum(),
                'mean': df['value'].mean(),
                'median': df['value'].median(),
                'std_dev': df['value'].std(),
                'seasonal_patterns': self._extract_seasonal_patterns(df)
            }

        return temporal_stats

    def _extract_seasonal_patterns(self, df):
        """
        Extract seasonal variation patterns
        """
        df['month'] = df.index.month
        seasonal_stats = df.groupby('month')['value'].agg([
            'mean', 'median', 'std'
        ])
        return seasonal_stats

# Usage
def main():
```

```
    integrator = AdvancedDataIntegrator(
        historical_path=r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_d
    )

    # Align datasets
    aligned_datasets = integrator.align_temporal_datasets()

    # Compute temporal statistics
    temporal_stats = integrator.compute_temporal_statistics(aligned_datasets)

    # Print or further analyze results
    for filename, stats in temporal_stats.items():
        print(f"\nTemporal Statistics for {filename}:")
        for stat_name, stat_value in stats.items():
            print(f"{stat_name}: {stat_value}")

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

```
Temporal Statistics for bury_daily_flow.csv:
total_missing_values: 0
mean: 3.644442979197622
median: 3.644442979197622
std_dev: 3.0666207993307184
seasonal_patterns:             mean    median       std
month
1       4.450151  3.644443  3.833930
2       4.083593  3.644443  3.631877
3       3.576466  3.644443  2.504690
4       3.161901  3.644443  1.714336
5       3.063019  3.644443  1.642485
6       3.056979  3.644443  2.698494
7       3.066441  3.644443  1.951481
8       3.218471  3.644443  2.216472
9       3.372937  3.644443  2.879077
10      3.820186  3.644443  3.111075
11      4.302970  3.644443  3.621292
12      4.577189  3.644443  4.746693

Temporal Statistics for bury_daily_rainfall.csv:
total_missing_values: 0
mean: 3.7754983428598874
median: 0.9
std_dev: 6.209935248255402
seasonal_patterns:             mean  median       std
month
1       4.501981     1.9  6.368442
2       3.630932     0.7  6.075503
3       3.372043     0.7  5.613580
4       2.779006     0.5  4.821663
5       2.767459     0.4  4.694461
6       3.150234     0.5  5.769328
7       3.130221     0.5  5.578952
8       3.738370     0.7  6.335389
9       3.997485     0.7  6.935225
10      4.483022     1.3  7.139210
11      4.788129     1.9  6.675517
12      4.941766     1.8  7.338919
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\966840409.py:117: Futur
eWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  merged_df['value'].fillna(merged_df['value'].mean(), inplace=True)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\966840409.py:114: Futur
eWarning: Series.fillna with 'method' is deprecated and will raise in a future ve
rsion. Use obj.ffill() or obj.bfill() instead.
  merged_df['value'] = merged_df['value'].fillna(method='ffill')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\966840409.py:117: Futur
eWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  merged_df['value'].fillna(merged_df['value'].mean(), inplace=True)
```

In [7]:

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

class FloodDataPreprocessor:
    def __init__(self, historical_path, realtime_path):
        """
        Initialize data preprocessor

        Parameters:
        - historical_path: Directory containing historical data
        - realtime_path: Path to real-time data file
        """
        self.historical_path = historical_path
        self.realtime_path = realtime_path

        # Stores processed datasets
        self.processed_data = {}

    def load_historical_datasets(self):
        """
        Load and preprocess historical datasets
        """
        # List of historical files to process
        historical_files = [
            'bury_daily_flow.csv',
            'bury_daily_rainfall.csv',
            'rochdale_daily_flow.csv',
            'rochdale_daily_rainfall.csv',
```

```
            'bury_peak_flow.csv',
            'rochdale_peak_flow.csv',
            'manchester_peak_flow.csv'
        ]

        for filename in historical_files:
            filepath = os.path.join(self.historical_path, filename)

            # Read CSV file
            try:
                # Handle different date column scenarios
                try:
                    df = pd.read_csv(filepath, parse_dates=['Date'])
                    date_column = 'Date'
                except:
                    try:
                        df = pd.read_csv(filepath, parse_dates=['date'])
                        date_column = 'date'
                    except:
                        # If no standard date column, load without parsing
                        df = pd.read_csv(filepath)
                        print(f"Warning: No date column found in {filename}")
            except Exception as e:
                print(f"Error reading {filename}: {e}")
                continue

            # Standardize column names
            df = self._standardize_columns(df, filename)

            # Ensure a date column exists
            if date_column not in df.columns:
                print(f"Skipping {filename} due to missing date column")
                continue

            # Set date as index if not already
            df.set_index(date_column, inplace=True)

            # Add metadata
            df['data_source'] = filename
            df['data_type'] = self._determine_data_type(filename)

            # Store processed dataset
            self.processed_data[filename] = df

        return self.processed_data

    def _standardize_columns(self, df, filename):
        """
        Standardize column names across different datasets
        """
        column_mapping = {
            'Flow': 'river_flow',
            'Rainfall': 'rainfall',
            'Stage (m)': 'river_stage',
            'Flow (m3/s)': 'river_flow_rate'
        }

        # Rename columns
        df.rename(columns={
            col: column_mapping.get(col, col)
```

```
            for col in df.columns
        }, inplace=True)

        return df

    def _determine_data_type(self, filename):
        """
        Determine data type based on filename
        """
        if 'flow' in filename.lower():
            return 'river_flow'
        elif 'rainfall' in filename.lower():
            return 'rainfall'
        elif 'peak_flow' in filename.lower():
            return 'peak_flow'
        else:
            return 'unknown'

    def load_realtime_data(self):
        """
        Load and preprocess real-time data
        """
        try:
            df = pd.read_csv(self.realtime_path, parse_dates=['river_timestamp']

            # Standardize column names
            df.rename(columns={
                'river_level': 'river_stage',
                'river_timestamp': 'timestamp'
            }, inplace=True)

            # Set timestamp as index
            df.set_index('timestamp', inplace=True)

            # Add metadata
            df['data_source'] = 'real_time'
            df['data_type'] = 'real_time_monitoring'

            self.processed_data['real_time'] = df

            return df
        except Exception as e:
            print(f"Error loading real-time data: {e}")
            return None

    def handle_missing_values(self, merged_datasets):
        """
        Handle missing values in merged datasets
        """
        processed_datasets = {}

        for dataset_name, df in merged_datasets.items():
            # Create a copy to avoid modifying original data
            processed_df = df.copy()

            # Identify numeric columns
            numeric_columns = processed_df.select_dtypes(include=[np.number]).cc

            # Handle missing values for each numeric column
            for col in numeric_columns:
```

```
                # Simple imputation strategies
                if processed_df[col].isnull().sum() > 0:
                    # Try different imputation methods
                    if 'flow' in col or 'stage' in col:
                        # For flow-related data, use linear interpolation
                        processed_df[col].fillna(method='ffill', inplace=True)
                        processed_df[col].fillna(method='bfill', inplace=True)
                    elif 'rainfall' in col:
                        # For rainfall, use median
                        processed_df[col].fillna(processed_df[col].median(), inp
                    else:
                        # Default to mean
                        processed_df[col].fillna(processed_df[col].mean(), inpla

            processed_datasets[dataset_name] = processed_df

        return processed_datasets

    def prepare_for_analysis(self):
        """
        Comprehensive data preparation method
        """
        # Load historical datasets
        historical_data = self.load_historical_datasets()

        # Load real-time data
        real_time_data = self.load_realtime_data()

        # Prepare merged datasets
        merged_datasets = {
            'daily_flow': pd.concat([
                self.processed_data.get('bury_daily_flow.csv', pd.DataFrame()),
                self.processed_data.get('rochdale_daily_flow.csv', pd.DataFrame(
            ]),
            'daily_rainfall': pd.concat([
                self.processed_data.get('bury_daily_rainfall.csv', pd.DataFrame(
                self.processed_data.get('rochdale_daily_rainfall.csv', pd.DataFr
            ]),
            'real_time': real_time_data if real_time_data is not None else pd.Da
        }

        # Handle missing values
        processed_datasets = self.handle_missing_values(merged_datasets)

        return processed_datasets

def main():
    # Paths to data
    historical_path = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_dat
    realtime_path = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\

    # Initialize preprocessor
    preprocessor = FloodDataPreprocessor(historical_path, realtime_path)

    # Prepare data for analysis
    processed_datasets = preprocessor.prepare_for_analysis()

    # Output processed datasets
    for dataset_name, df in processed_datasets.items():
        print(f"\n{dataset_name} Dataset:")
```

```
        print(df.info())
        print("\nFirst few rows:")
        print(df.head())

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

```
daily_flow Dataset:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 21046 entries, 1995-11-22 to 2023-09-30
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   river_flow   21046 non-null  float64
 1   Extra        0 non-null      float64
 2   data_source  21046 non-null  object 
 3   data_type    21046 non-null  object 
dtypes: float64(2), object(2)
memory usage: 822.1+ KB
None

First few rows:
            river_flow  Extra          data_source   data_type
Date                                                          
1995-11-22       0.897    NaN  bury_daily_flow.csv  river_flow
1995-11-23       0.831    NaN  bury_daily_flow.csv  river_flow
1995-11-24       0.991    NaN  bury_daily_flow.csv  river_flow
1995-11-25       1.080    NaN  bury_daily_flow.csv  river_flow
1995-11-26       1.124    NaN  bury_daily_flow.csv  river_flow

daily_rainfall Dataset:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 21550 entries, 1961-01-01 to 2017-12-31
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   rainfall     21550 non-null  float64
 1   Extra        21550 non-null  int64  
 2   data_source  21550 non-null  object 
 3   data_type    21550 non-null  object 
dtypes: float64(1), int64(1), object(2)
memory usage: 841.8+ KB
None

First few rows:
            rainfall  Extra              data_source data_type
Date                                                          
1961-01-01       9.4   1000  bury_daily_rainfall.csv  rainfall
1961-01-02      13.7   1000  bury_daily_rainfall.csv  rainfall
1961-01-03       3.0   1000  bury_daily_rainfall.csv  rainfall
1961-01-04       0.1   1000  bury_daily_rainfall.csv  rainfall
1961-01-05      13.0   1000  bury_daily_rainfall.csv  rainfall

real_time Dataset:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 390 entries, 2025-01-30 11:15:00+00:00 to 2025-01-31 21:00:00+00:0
0
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   river_stage           390 non-null    float64
 1   rainfall              390 non-null    float64
 2   rainfall_timestamp    390 non-null    object 
 3   location_name         390 non-null    object 
 4   river_station_id      390 non-null    int64  
 5   rainfall_station_id   390 non-null    int64  
 6   collection_timestamp  390 non-null    object 
```

```
 7   data_source           390 non-null    object
 8   data_type             390 non-null    object
dtypes: float64(2), int64(2), object(5)
memory usage: 30.5+ KB
None

First few rows:
                           river_stage  rainfall         rainfall_timestamp  \
timestamp
2025-01-30 11:15:00+00:00        0.235       0.3  2025-01-30 11:15:00+00:00
2025-01-30 11:15:00+00:00        1.064       0.3  2025-01-30 11:15:00+00:00
2025-01-30 11:15:00+00:00        0.385       0.3  2025-01-30 11:15:00+00:00
2025-01-30 11:30:00+00:00        0.235       0.3  2025-01-30 11:30:00+00:00
2025-01-30 11:30:00+00:00        1.064       0.3  2025-01-30 11:30:00+00:00

                                   location_name  river_station_id  \
timestamp
2025-01-30 11:15:00+00:00               Rochdale            690203
2025-01-30 11:15:00+00:00  Manchester Racecourse            690510
2025-01-30 11:15:00+00:00            Bury Ground            690160
2025-01-30 11:30:00+00:00               Rochdale            690203
2025-01-30 11:30:00+00:00  Manchester Racecourse            690510

                           rainfall_station_id collection_timestamp  \
timestamp
2025-01-30 11:15:00+00:00               561613           30/01/2025
2025-01-30 11:15:00+00:00               562992           30/01/2025
2025-01-30 11:15:00+00:00               562656           30/01/2025
2025-01-30 11:30:00+00:00               561613           30/01/2025
2025-01-30 11:30:00+00:00               562992           30/01/2025

                          data_source            data_type
timestamp
2025-01-30 11:15:00+00:00   real_time  real_time_monitoring
2025-01-30 11:15:00+00:00   real_time  real_time_monitoring
2025-01-30 11:15:00+00:00   real_time  real_time_monitoring
2025-01-30 11:30:00+00:00   real_time  real_time_monitoring
2025-01-30 11:30:00+00:00   real_time  real_time_monitoring
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\2285139135.py:162: Futu
reWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  processed_df[col].fillna(processed_df[col].mean(), inplace=True)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\2285139135.py:159: Futu
reWarning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  processed_df[col].fillna(processed_df[col].median(), inplace=True)
```

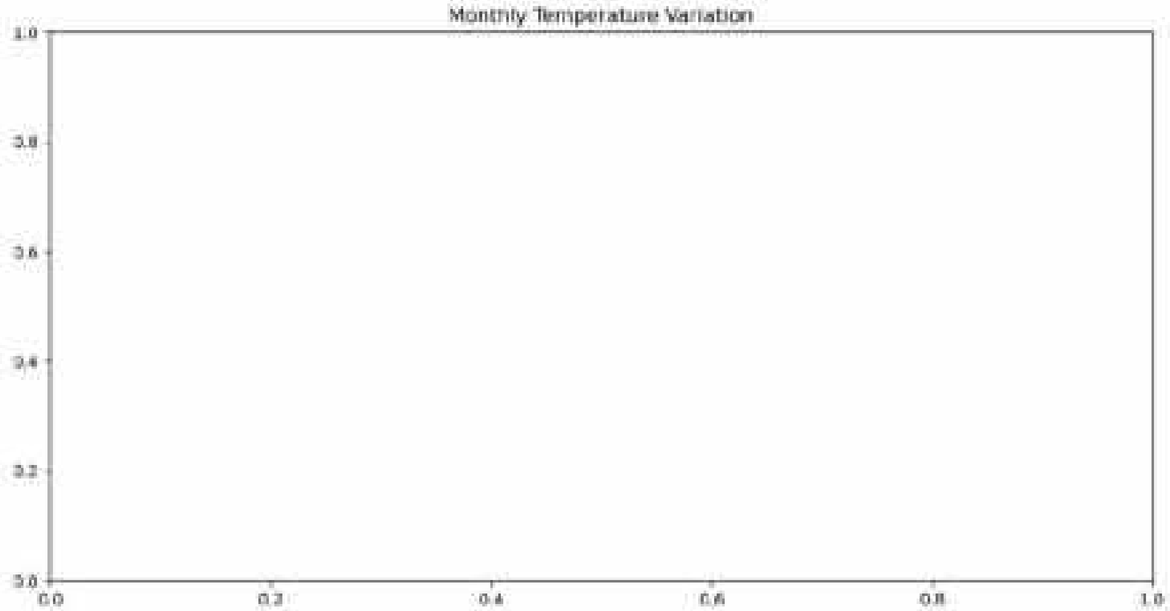
In [2]:

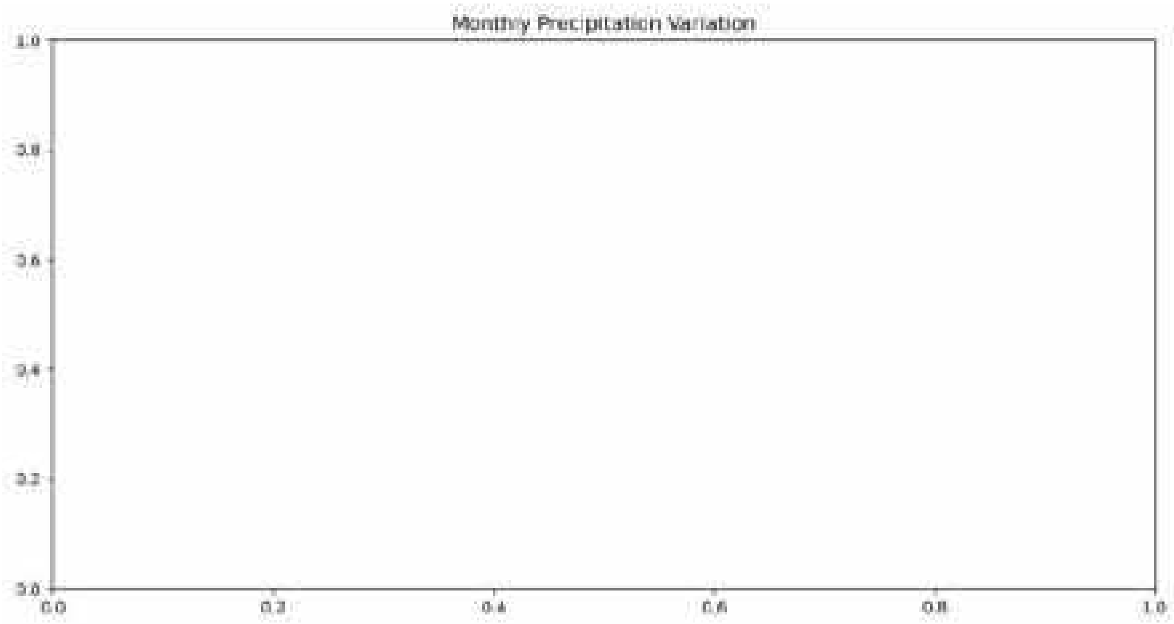
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class MonthlyWeatherDataProcessor:
    def __init__(self, filepath):
        """
        Initialize Monthly Weather Data Processor

        Parameters:
        - filepath: Full path to the monthly weather data CSV
        """
        self.filepath = filepath
        self.data = None

    def load_and_process_data(self):
        """
        Load and process monthly weather data

        Returns:
        - Processed DataFrame
        """
        # Read the CSV file
        self.data = pd.read_csv(self.filepath)

        # Convert Month column to categorical
        month_order = ['January', 'February', 'March', 'April', 'May', 'June',
                       'July', 'August', 'September', 'October', 'November', 'De
        self.data['Month'] = pd.Categorical(self.data['Month'], categories=month

        return self.data
```

```
    def analyze_weather_data(self):
        """
        Perform comprehensive analysis of monthly weather data

        Returns:
        - Dictionary with analysis results
        """
        if self.data is None:
            self.load_and_process_data()

        # Aggregate data by station
        station_analysis = self.data.groupby('Station').agg({
            'Temperature_C': ['mean', 'min', 'max'],
            'Precipitation_mm': ['mean', 'min', 'max']
        })

        # Seasonal analysis
        seasonal_groups = {
            'Winter': ['December', 'January', 'February'],
            'Spring': ['March', 'April', 'May'],
            'Summer': ['June', 'July', 'August'],
            'Autumn': ['September', 'October', 'November']
        }

        seasonal_analysis = {}
        for season, months in seasonal_groups.items():
            seasonal_data = self.data[self.data['Month'].isin(months)]
            seasonal_analysis[season] = {
                'mean_temperature': seasonal_data['Temperature_C'].mean(),
                'mean_precipitation': seasonal_data['Precipitation_mm'].mean()
            }

        return {
            'station_analysis': station_analysis,
            'seasonal_analysis': seasonal_analysis
        }

    def visualize_weather_data(self):
        """
        Create visualizations of monthly weather data
        """
        # Prepare output directory
        output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data\weather_
        os.makedirs(output_dir, exist_ok=True)

        # Temperature visualization
        plt.figure(figsize=(12, 6))
        plt.title('Monthly Temperature Variation')
        self.data.boxplot(column='Temperature_C', by='Month')
        plt.xlabel('Month')
        plt.ylabel('Temperature (°C)')
        plt.suptitle('')  # Remove automatic suptitle
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, 'monthly_temperature.png'))
        plt.close()

        # Precipitation visualization
        plt.figure(figsize=(12, 6))
        plt.title('Monthly Precipitation Variation')
        self.data.boxplot(column='Precipitation_mm', by='Month')
```

```
        plt.xlabel('Month')
        plt.ylabel('Precipitation (mm)')
        plt.suptitle('')  # Remove automatic suptitle
        plt.tight_layout()
        plt.savefig(os.path.join(output_dir, 'monthly_precipitation.png'))
        plt.close()

    def save_processed_data(self):
        """
        Save processed weather data
        """
        # Prepare output directory
        output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data\weather_
        os.makedirs(output_dir, exist_ok=True)

        # Save processed data
        output_path = os.path.join(output_dir, 'processed_monthly_weather_data.c
        self.data.to_csv(output_path, index=False)
        print(f"Processed weather data saved to {output_path}")

def main():
    # Set file path
    filepath = r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WEATHE

    # Initialize processor
    processor = MonthlyWeatherDataProcessor(filepath)

    # Load and process data
    data = processor.load_and_process_data()

    # Analyze data
    weather_analysis = processor.analyze_weather_data()

    # Print analysis results
    print("\nStation Analysis:")
    print(weather_analysis['station_analysis'])

    print("\nSeasonal Analysis:")
    for season, stats in weather_analysis['seasonal_analysis'].items():
        print(f"\n{season}:")
        for key, value in stats.items():
            print(f"  {key}: {value:.2f}")

    # Create visualizations
    processor.visualize_weather_data()

    # Save processed data
    processor.save_processed_data()

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

```
Station Analysis:
                      Temperature_C            Precipitation_mm
                               mean  min   max             mean min  max
Station
BURY MANCHESTER            9.183333  3.8  15.5       111.916667  79  157
MANCHESTER RACECOURSE     10.927273  5.0  16.8        82.666667  59  108
ROCHDALE                   8.991667  3.6  15.3       108.666667  77  136

Seasonal Analysis:

Winter:
  mean_temperature: 4.44
  mean_precipitation: 119.11

Spring:
  mean_temperature: 9.51
  mean_precipitation: 81.00

Summer:
  mean_temperature: 15.07
  mean_precipitation: 91.56

Autumn:
  mean_temperature: 9.07
  mean_precipitation: 112.67
Processed weather data saved to C:\Users\Administrator\NEWPROJECT\processed_data
\weather_data\processed_monthly_weather_data.csv
```





In [2]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def create_detailed_visualizations(data):
    """
    Create comprehensive visualizations for weather data

    Parameters:
    - data (pd.DataFrame): Monthly weather data
    """
    # Prepare output directory
    output_dir = r'C:\Users\Administrator\NEWPROJECT\processed_data\weather_anal
    os.makedirs(output_dir, exist_ok=True)

    # 1. Temperature Visualization
    plt.figure(figsize=(12, 6))
    grouped_temp = data.groupby('Month')['Temperature_C']
    temp_means = grouped_temp.mean()
    temp_stds = grouped_temp.std()

    plt.bar(temp_means.index, temp_means, yerr=temp_stds, capsize=5)
    plt.title('Monthly Temperature Variation', fontsize=16)
    plt.xlabel('Month', fontsize=12)
    plt.ylabel('Temperature (°C)', fontsize=12)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'monthly_temperature_variation.png'))
    plt.close()

    # 2. Precipitation Visualization
    plt.figure(figsize=(12, 6))
    grouped_precip = data.groupby('Month')['Precipitation_mm']
    precip_means = grouped_precip.mean()
    precip_stds = grouped_precip.std()

    plt.bar(precip_means.index, precip_means, yerr=precip_stds, capsize=5)
    plt.title('Monthly Precipitation Variation', fontsize=16)
    plt.xlabel('Month', fontsize=12)
```

```
    plt.ylabel('Precipitation (mm)', fontsize=12)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'monthly_precipitation_variation.png'))
    plt.close()

    # 3. Station Comparison Visualization
    plt.figure(figsize=(15, 6))

    # Temperature subplot
    plt.subplot(1, 2, 1)
    station_temp_means = data.groupby('Station')['Temperature_C'].mean()
    station_temp_stds = data.groupby('Station')['Temperature_C'].std()

    plt.bar(station_temp_means.index, station_temp_means, yerr=station_temp_stds
    plt.title('Average Temperature by Station', fontsize=14)
    plt.xlabel('Station', fontsize=10)
    plt.ylabel('Temperature (°C)', fontsize=10)
    plt.xticks(rotation=45, ha='right')

    # Precipitation subplot
    plt.subplot(1, 2, 2)
    station_precip_means = data.groupby('Station')['Precipitation_mm'].mean()
    station_precip_stds = data.groupby('Station')['Precipitation_mm'].std()

    plt.bar(station_precip_means.index, station_precip_means, yerr=station_preci
    plt.title('Average Precipitation by Station', fontsize=14)
    plt.xlabel('Station', fontsize=10)
    plt.ylabel('Precipitation (mm)', fontsize=10)
    plt.xticks(rotation=45, ha='right')

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'station_comparison.png'))
    plt.close()

    print("Visualizations have been saved to:", output_dir)

def main():
    # Set file path
    filepath = r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WEATHE

    # Read the data
    data = pd.read_csv(filepath)

    # Create visualizations
    create_detailed_visualizations(data)

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

```
Visualizations have been saved to: C:\Users\Administrator\NEWPROJECT\processed_da
ta\weather_analysis
```

In [3]:

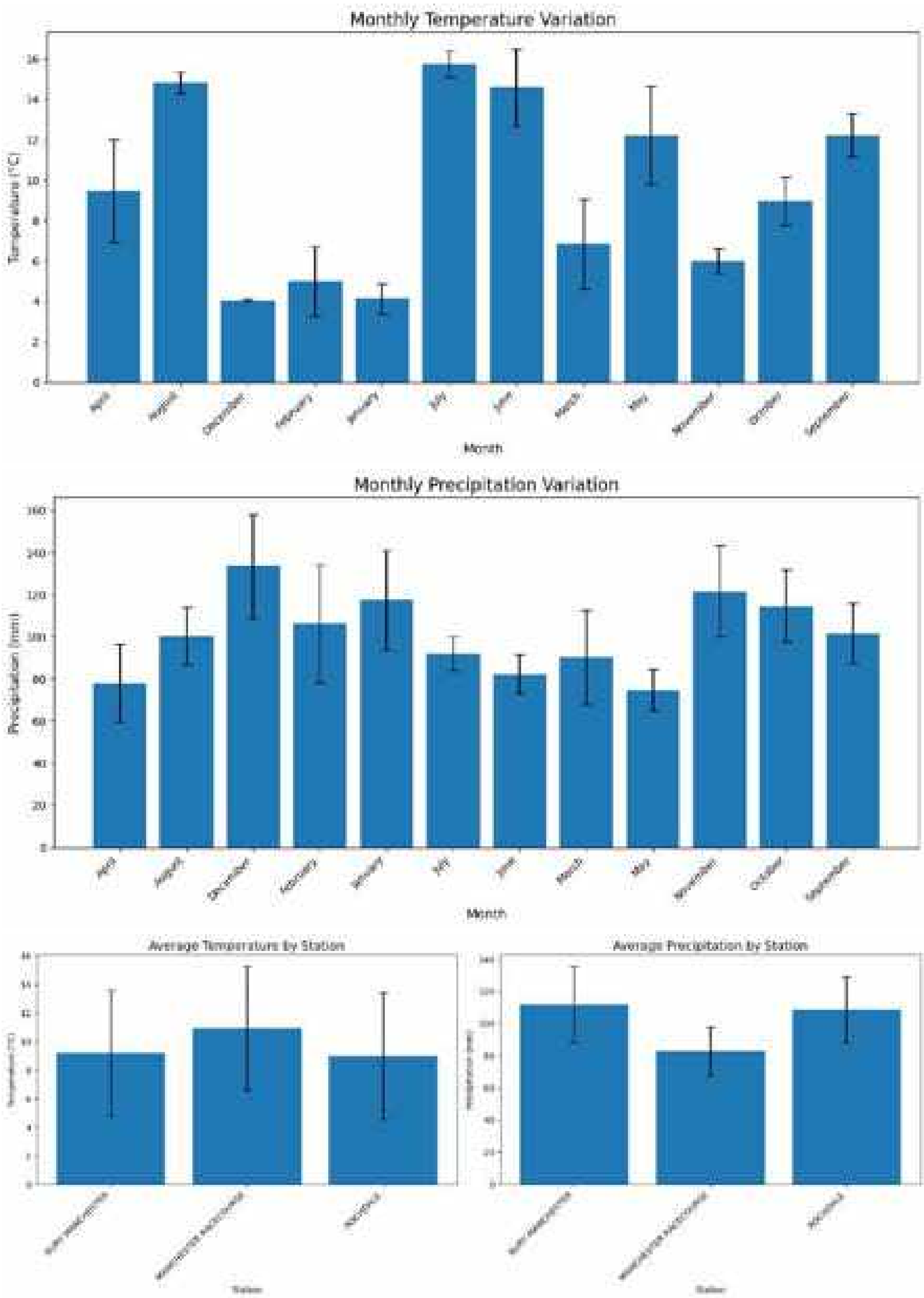
```
import matplotlib.pyplot as plt
```

In [5]:

```
plt.plot([1, 2, 3, 4])
plt.title("Simple Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load data
filepath = r'C:\Users\Administrator\NEWPROJECT\MANUAL DATA COLLECTION\WEATHER\cl
data = pd.read_csv(filepath)

# 1. Temperature Visualization
plt.figure(figsize=(12, 6))
grouped_temp = data.groupby('Month')['Temperature_C']
temp_means = grouped_temp.mean()
temp_stds = grouped_temp.std()

plt.bar(temp_means.index, temp_means, yerr=temp_stds, capsize=5)
plt.title('Monthly Temperature Variation', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Temperature (°C)', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# 2. Precipitation Visualization
plt.figure(figsize=(12, 6))
grouped_precip = data.groupby('Month')['Precipitation_mm']
precip_means = grouped_precip.mean()
precip_stds = grouped_precip.std()

plt.bar(precip_means.index, precip_means, yerr=precip_stds, capsize=5)
plt.title('Monthly Precipitation Variation', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Precipitation (mm)', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# 3. Station Comparison Visualization
plt.figure(figsize=(15, 6))

# Temperature subplot
plt.subplot(1, 2, 1)
station_temp_means = data.groupby('Station')['Temperature_C'].mean()
station_temp_stds = data.groupby('Station')['Temperature_C'].std()

plt.bar(station_temp_means.index, station_temp_means, yerr=station_temp_stds, ca
plt.title('Average Temperature by Station', fontsize=14)
plt.xlabel('Station', fontsize=10)
plt.ylabel('Temperature (°C)', fontsize=10)
plt.xticks(rotation=45, ha='right')

# Precipitation subplot
plt.subplot(1, 2, 2)
station_precip_means = data.groupby('Station')['Precipitation_mm'].mean()
station_precip_stds = data.groupby('Station')['Precipitation_mm'].std()

plt.bar(station_precip_means.index, station_precip_means, yerr=station_precip_st
plt.title('Average Precipitation by Station', fontsize=14)
plt.xlabel('Station', fontsize=10)
plt.ylabel('Precipitation (mm)', fontsize=10)
plt.xticks(rotation=45, ha='right')
```

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



In [20]:
```
import pandas as pd
import os

# Paths to peak flow files
peak_flow_files = [
    r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\historical\bury_
    r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\historical\rochd
```

```
    r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\historical\manch
]

# Comprehensive peak flow data analysis
def analyze_peak_flow_data(files):
    peak_flow_datasets = {}

    for file_path in files:
        location = os.path.basename(file_path).split('_')[0]

        try:
            # Read the CSV file
            df = pd.read_csv(file_path)

            # Basic dataset information
            peak_flow_datasets[location] = {
                'filename': os.path.basename(file_path),
                'shape': df.shape,
                'columns': df.columns.tolist(),
                'data_summary': df.describe().to_dict()
            }

            # Check for date-related columns
            date_columns = [col for col in df.columns if 'date' in col.lower()]
            if date_columns:
                for col in date_columns:
                    try:
                        df[col] = pd.to_datetime(df[col])
                        peak_flow_datasets[location]['date_range'] = {
                            'start': df[col].min(),
                            'end': df[col].max()
                        }
                    except:
                        pass

            # Print detailed information
            print(f"\nPeak Flow Analysis for {location.upper()}:")
            print(f"File: {os.path.basename(file_path)}")
            print(f"Shape: {df.shape}")
            print("\nColumns:")
            print(df.columns.tolist())
            print("\nSample Data:")
            print(df.head())
            print("\nDescriptive Statistics:")
            print(df.describe())
            print("\n" + "="*50)

        except Exception as e:
            print(f"Error processing {file_path}: {e}")

    return peak_flow_datasets

# Run the analysis
peak_flow_analysis = analyze_peak_flow_data(peak_flow_files)
```

```
Peak Flow Analysis for BURY:
File: bury_peak_flow.csv
Shape: (51, 7)

Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Sample Data:
  Water Year       Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1972-1973 1973-01-12  00:00:00      1.255       78.130       NaN
1  1973-1974 1974-02-11  00:00:00      1.473      118.020       NaN
2  1974-1975 1975-01-21  00:00:00      1.450      113.410       NaN
3  1975-1976 1976-01-02  17:45:00      1.468      116.886  In Range
4  1976-1977 1977-09-30  20:00:00      1.258       78.636  In Range

             Datetime
0 1973-01-12 00:00:00
1 1974-02-11 00:00:00
2 1975-01-21 00:00:00
3 1976-01-02 17:45:00
4 1977-09-30 20:00:00

Descriptive Statistics:
                                Date  Stage (m)  Flow (m3/s)  \
count                             51  51.000000    51.000000
mean   1998-01-25 01:24:42.352941184   1.449549   115.931412
min              1973-01-12 00:00:00   1.074000    51.511000
25%              1985-05-28 12:00:00   1.292500    84.578000
50%              1998-01-08 00:00:00   1.447000   112.880000
75%              2010-04-26 12:00:00   1.511500   125.589500
max              2023-07-23 00:00:00   2.178000   283.649000
std                              NaN   0.208924    43.598881

                            Datetime
count                             51
mean   1998-01-25 13:52:21.176470656
min              1973-01-12 00:00:00
25%              1985-05-29 05:00:00
50%              1998-01-08 21:30:00
75%              2010-04-26 23:30:00
max              2023-07-23 12:15:00
std                              NaN

==================================================

Peak Flow Analysis for ROCHDALE:
File: rochdale_peak_flow.csv
Shape: (31, 7)

Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Sample Data:
  Water Year       Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1992-1993 1993-09-13  11:30:00      0.892       21.131  In Range
1  1993-1994 1993-12-08  23:45:00      1.286       38.328  In Range
2  1994-1995 1995-01-31  23:15:00      1.637       56.671  In Range
3  1995-1996 1996-02-18  03:15:00      0.808       17.976  In Range
4  1996-1997 1996-11-06  02:15:00      1.243       36.269  In Range
```

```
             Datetime
0 1993-09-13 11:30:00
1 1993-12-08 23:45:00
2 1995-01-31 23:15:00
3 1996-02-18 03:15:00
4 1996-11-06 02:15:00

Descriptive Statistics:
                                Date  Stage (m)  Flow (m3/s)  \
count                             31  31.000000    31.000000
mean   2008-02-06 15:29:01.935483904   1.429774    46.377129
min              1993-09-13 00:00:00   0.808000    17.976000
25%              2000-10-08 12:00:00   1.281500    38.115500
50%              2008-01-21 00:00:00   1.413000    44.654000
75%              2015-08-13 00:00:00   1.539500    51.310000
max              2023-07-23 00:00:00   2.222000    92.846000
std                              NaN   0.285128    15.045485

                            Datetime
count                             31
mean   2008-02-07 03:54:40.645161216
min              1993-09-13 11:30:00
25%              2000-10-08 12:52:30
50%              2008-01-21 13:30:00
75%              2015-08-13 07:00:00
max              2023-07-23 12:15:00
std                              NaN

==================================================

Peak Flow Analysis for MANCHESTER:
File: manchester_peak_flow.csv
Shape: (82, 7)

Columns:
['Water Year', 'Date', 'Time', 'Stage (m)', 'Flow (m3/s)', 'Rating', 'Datetime']

Sample Data:
  Water Year       Date      Time  Stage (m)  Flow (m3/s)   Rating   Datetime
0  1941-1942 1941-10-24  00:00:00       3.47        269.0  Extrap. 1941-10-24
1  1942-1943 1942-10-17  00:00:00       3.16        223.0  Extrap. 1942-10-17
2  1943-1944 1944-01-23  00:00:00       4.10        374.0  Extrap. 1944-01-23
3  1944-1945 1945-02-02  00:00:00       3.90        339.0  Extrap. 1945-02-02
4  1945-1946 1946-09-20  00:00:00       5.33        500.0  Extrap. 1946-09-20

Descriptive Statistics:
                                Date  Stage (m)  Flow (m3/s)  \
count                             82  82.000000    82.000000
mean   1982-08-01 18:43:54.146341440   3.513146   279.434841
min              1941-10-24 00:00:00   2.460000   135.000000
25%              1962-04-11 18:00:00   3.118000   217.277000
50%              1982-06-08 12:00:00   3.500000   273.500000
75%              2002-11-10 06:00:00   3.831500   327.306250
max              2023-01-10 00:00:00   5.668000   560.000000
std                              NaN   0.590061    87.357138

                            Datetime
count                             82
mean   1982-08-02 01:25:36.585365824
min              1941-10-24 00:00:00
```

```
25%          1962-04-11 18:00:00
50%          1982-06-09 00:00:00
75%          2002-11-10 12:41:15
max          2023-01-10 16:15:00
std                          NaN

==================================================
```

STAGE 1

In [21]:

```python
import pandas as pd
import glob
import os

def merge_combined_data():
    # Path to your combined_data directory
    path = r'C:\Users\Administrator\NEWPROJECT\combined_data'

    # Get all CSV files in the directory
    all_files = glob.glob(os.path.join(path, "*.csv"))

    # Create empty list to store dataframes
    dfs = []

    # Read each CSV file and append to list
    for file in all_files:
        df = pd.read_csv(file)
        dfs.append(df)

    # Concatenate all dataframes
    merged_df = pd.concat(dfs, ignore_index=True)

    # Sort by timestamp and location
    merged_df['river_timestamp'] = pd.to_datetime(merged_df['river_timestamp'])
    merged_df = merged_df.sort_values(['river_timestamp', 'location_name'])

    # Remove any duplicates if they exist
    merged_df = merged_df.drop_duplicates()

    return merged_df

# Execute the merge
merged_data = merge_combined_data()

# Basic data validation
print("\nDataset Overview:")
print("-----------------")
print(f"Total records: {len(merged_data)}")
print(f"\nRecords per location:")
print(merged_data.groupby('location_name').size())
print(f"\nDate range: {merged_data['river_timestamp'].min()} to {merged_data['ri
print(f"\nSample of merged data:")
print(merged_data.head())

# Save merged dataset
output_path = r'C:\Users\Administrator\NEWPROJECT\cleaned_data\merged_realtime_d
merged_data.to_csv(output_path, index=False)
print(f"\nMerged data saved to: {output_path}")
```

```
Dataset Overview:
-----------------
Total records: 1209

Records per location:
location_name
Bury Ground              403
Manchester Racecourse    403
Rochdale                 403
dtype: int64

Date range: 2025-01-30 11:15:00+00:00 to 2025-02-04 12:15:00+00:00

Sample of merged data:
   river_level           river_timestamp  rainfall    rainfall_timestamp  \
2        0.385 2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
1        1.064 2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
0        0.235 2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
5        0.386 2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z
4        1.064 2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z

           location_name  river_station_id  rainfall_station_id
2            Bury Ground            690160               562656
1  Manchester Racecourse            690510               562992
0               Rochdale            690203               561613
5            Bury Ground            690160               562656
4  Manchester Racecourse            690510               562992

Merged data saved to: C:\Users\Administrator\NEWPROJECT\cleaned_data\merged_realt
ime_data.csv
```

In [22]:

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

# Load real-time data
realtime_df = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\merge

# Load historical data for each station
bury_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_d
rochdale_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\riv
bury_rainfall = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\riv
rochdale_rainfall = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data

# Display basic information about each dataset
print("Real-time Data Info:")
print(realtime_df.info())
print("\nSample of real-time data:")
print(realtime_df.head())
```

```
Real-time Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1209 entries, 0 to 1208
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   river_level          1209 non-null   float64
 1   river_timestamp      1209 non-null   object 
 2   rainfall             1209 non-null   float64
 3   rainfall_timestamp   1209 non-null   object 
 4   location_name        1209 non-null   object 
 5   river_station_id     1209 non-null   int64  
 6   rainfall_station_id  1209 non-null   int64  
dtypes: float64(2), int64(2), object(3)
memory usage: 66.2+ KB
None

Sample of real-time data:
   river_level            river_timestamp  rainfall    rainfall_timestamp  \
0        0.385  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z   
1        1.064  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z   
2        0.235  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z   
3        0.386  2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z   
4        1.064  2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z   

          location_name  river_station_id  rainfall_station_id  
0           Bury Ground            690160               562656  
1  Manchester Racecourse           690510               562992  
2              Rochdale            690203               561613  
3           Bury Ground            690160               562656  
4  Manchester Racecourse           690510               562992  
```

In [23]:

```python
import pandas as pd
import numpy as np
from datetime import datetime

# Convert timestamps to datetime and set as index
realtime_df['river_timestamp'] = pd.to_datetime(realtime_df['river_timestamp'])

# Separate data by station
station_data = {}
for station in realtime_df['location_name'].unique():
    station_data[station] = realtime_df[realtime_df['location_name'] == station]
    station_data[station].set_index('river_timestamp', inplace=True)
    station_data[station].sort_index(inplace=True)

# Calculate basic statistics for each station
station_stats = {}
for station, data in station_data.items():
    stats = {
        'mean_level': data['river_level'].mean(),
        'max_level': data['river_level'].max(),
        'min_level': data['river_level'].min(),
        'std_level': data['river_level'].std(),
        'total_readings': len(data)
    }
    station_stats[station] = stats

# Create a baseline dataset
baseline_df = pd.DataFrame(station_stats).T
```

```
baseline_df.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\station_base

# Display statistics
print("Station Statistics:")
print(baseline_df)

# Calculate time-based patterns
for station, data in station_data.items():
    print(f"\nTime-based Analysis for {station}:")
    # Hourly averages
    hourly_avg = data['river_level'].resample('H').mean()
    print(f"Hourly average range: {hourly_avg.min():.3f} to {hourly_avg.max():.3
```

```
Station Statistics:
                        mean_level  max_level  min_level  std_level  \
Bury Ground               0.365196      0.441      0.333   0.027309
Manchester Racecourse     1.039347      1.203      0.962   0.062598
Rochdale                  0.223757      0.293      0.195   0.024700

                        total_readings
Bury Ground                      403.0
Manchester Racecourse            403.0
Rochdale                         403.0

Time-based Analysis for Bury Ground:
Hourly average range: 0.333 to 0.441

Time-based Analysis for Manchester Racecourse:
Hourly average range: 0.963 to 1.201

Time-based Analysis for Rochdale:
Hourly average range: 0.195 to 0.291
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\3404852022.py:39: Futur
eWarning: 'H' is deprecated and will be removed in a future version, please use
'h' instead.
  hourly_avg = data['river_level'].resample('H').mean()
```

Inter-station Correlations

In [26]:
```
# 1. checking our data structure and handle duplicates properly
# Group by timestamp and station, taking mean if there are duplicates
pivot_df = realtime_df.groupby(['river_timestamp', 'location_name'])['river_leve

# Calculate correlations
correlations = pivot_df.corr()

# 2. Analyze patterns with proper hourly resampling
station_patterns = {}
for station in realtime_df['location_name'].unique():
    # Filter data for station
    station_data = realtime_df[realtime_df['location_name'] == station].copy()
    station_data.set_index('river_timestamp', inplace=True)

    # Resample to hourly frequency and calculate statistics
    hourly_patterns = station_data['river_level'].resample('h').agg({
        'mean': 'mean',
        'min': 'min',
        'max': 'max',
        'count': 'count'
```

```
    })

    # Save hourly patterns
    output_path = f'C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\hourly_p
    hourly_patterns.to_csv(output_path)
    station_patterns[station] = hourly_patterns

# Save correlations
correlations.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\station_cor

print("Inter-station Correlations:")
print(correlations)
print("\nHourly pattern sample for each station:")
for station, patterns in station_patterns.items():
    print(f"\n{station}:")
    print(patterns.head())
```

```
Inter-station Correlations:
location_name          Bury Ground  Manchester Racecourse  Rochdale
location_name
Bury Ground               1.000000               0.949021  0.974525
Manchester Racecourse     0.949021               1.000000  0.920792
Rochdale                  0.974525               0.920792  1.000000

Hourly pattern sample for each station:

Bury Ground:
                              mean    min    max  count
river_timestamp
2025-01-30 11:00:00+00:00  0.38600  0.385  0.387      3
2025-01-30 12:00:00+00:00  0.38825  0.388  0.389      4
2025-01-30 13:00:00+00:00  0.38550  0.383  0.388      4
2025-01-30 14:00:00+00:00  0.38225  0.382  0.383      4
2025-01-30 15:00:00+00:00  0.38100  0.381  0.381      4

Manchester Racecourse:
                               mean    min    max  count
river_timestamp
2025-01-30 11:00:00+00:00  1.063667  1.063  1.064      3
2025-01-30 12:00:00+00:00  1.062000  1.061  1.063      4
2025-01-30 13:00:00+00:00  1.059750  1.059  1.061      4
2025-01-30 14:00:00+00:00  1.059000  1.058  1.060      4
2025-01-30 15:00:00+00:00  1.056250  1.054  1.058      4

Rochdale:
                               mean    min    max  count
river_timestamp
2025-01-30 11:00:00+00:00  0.235333  0.235  0.236      3
2025-01-30 12:00:00+00:00  0.236500  0.236  0.237      4
2025-01-30 13:00:00+00:00  0.237500  0.237  0.238      4
2025-01-30 14:00:00+00:00  0.238250  0.238  0.239      4
2025-01-30 15:00:00+00:00  0.239000  0.239  0.239      4
```

In [27]:
```
# 1. analyze rainfall patterns and their relationship with river levels
import pandas as pd
import numpy as np
from datetime import datetime

# Group data by timestamp and analyze rainfall with river levels
rain_analysis = realtime_df.groupby(['river_timestamp', 'location_name']).agg({
```

```
    'rainfall': 'sum',
    'river_level': 'mean'
}).reset_index()

# Create separate analysis for each station
for station in rain_analysis['location_name'].unique():
    station_data = rain_analysis[rain_analysis['location_name'] == station].copy
    station_data.set_index('river_timestamp', inplace=True)

    # Calculate rolling statistics
    window_size = 4  # 1-hour window (4 x 15-minute readings)
    station_data['rolling_level'] = station_data['river_level'].rolling(window=w
    station_data['cumulative_rain'] = station_data['rainfall'].rolling(window=wi

    # Save the analysis
    output_path = f'C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\rainfall
    station_data.to_csv(output_path)

    print(f"\nAnalysis for {station}:")
    print("Basic Statistics:")
    print(station_data.describe())

    # Calculate correlation between rainfall and river level
    correlation = station_data['river_level'].corr(station_data['cumulative_rain
    print(f"\nCorrelation between rainfall and river level: {correlation:.4f}")
```

```
Analysis for Bury Ground:
Basic Statistics:
         rainfall  river_level  rolling_level  cumulative_rain
count  403.000000   403.000000     400.000000       400.000000
mean     0.020347     0.365196       0.365238         0.082000
std      0.101152     0.027309       0.027273         0.378463
min      0.000000     0.333000       0.333000         0.000000
25%      0.000000     0.342000       0.341750         0.000000
50%      0.000000     0.356000       0.356250         0.000000
75%      0.000000     0.381000       0.381312         0.000000
max      1.000000     0.441000       0.441000         3.300000

Correlation between rainfall and river level: 0.1508

Analysis for Manchester Racecourse:
Basic Statistics:
         rainfall  river_level  rolling_level  cumulative_rain
count  395.000000   395.000000     392.000000       392.000000
mean     0.020506     1.040418       1.040599         0.082653
std      0.096433     0.062771       0.062753         0.358392
min      0.000000     0.962000       0.963250         0.000000
25%      0.000000     0.988000       0.988937         0.000000
50%      0.000000     1.016000       1.015125         0.000000
75%      0.000000     1.060000       1.060750         0.000000
max      1.000000     1.203000       1.201750         3.000000

Correlation between rainfall and river level: 0.0859

Analysis for Rochdale:
Basic Statistics:
         rainfall  river_level  rolling_level  cumulative_rain
count  403.000000   403.000000     400.000000       400.000000
mean     0.016873     0.223757       0.223822         0.068000
std      0.077992     0.024700       0.024686         0.284567
min      0.000000     0.195000       0.195000         0.000000
25%      0.000000     0.205000       0.204687         0.000000
50%      0.000000     0.215000       0.214875         0.000000
75%      0.000000     0.237000       0.236750         0.000000
max      0.600000     0.293000       0.292000         2.000000

Correlation between rainfall and river level: 0.2244
```

In [28]:

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

# Create time-lagged analysis
def analyze_time_lags(df, station_name, max_lag_hours=6):
    # Filter for specific station
    station_data = df[df['location_name'] == station_name].copy()
    station_data.set_index('river_timestamp', inplace=True)
    station_data.sort_index(inplace=True)

    # Create lags from 15 minutes to max_lag_hours
    lags = range(1, (max_lag_hours * 4) + 1)  # 4 readings per hour
    correlations = []

    for lag in lags:
        # Create lagged rainfall
        station_data[f'rainfall_lag_{lag}'] = station_data['rainfall'].shift(-la
```

```
        # Calculate correlation
        corr = station_data['river_level'].corr(station_data[f'rainfall_lag_{lag
        correlations.append({
            'lag_periods': lag,
            'lag_hours': lag/4,
            'correlation': corr
        })

    # Convert to DataFrame
    lag_analysis = pd.DataFrame(correlations)

    # Calculate cumulative rainfall effects
    station_data['cumulative_3h'] = station_data['rainfall'].rolling(window=12).
    station_data['cumulative_6h'] = station_data['rainfall'].rolling(window=24).

    # Save detailed analysis
    output_path = f'C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\lag_anal
    lag_analysis.to_csv(output_path)

    # Save processed station data
    station_output = f'C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\proce
    station_data.to_csv(station_output)

    return lag_analysis, station_data

# Perform analysis for each station
for station in realtime_df['location_name'].unique():
    print(f"\nTime-lag Analysis for {station}:")
    lag_analysis, station_data = analyze_time_lags(realtime_df, station)

    print("\nLag Correlation Summary:")
    print(lag_analysis.sort_values('correlation', ascending=False).head())

    print("\nCumulative Rainfall Effects:")
    print(station_data[['river_level', 'rainfall', 'cumulative_3h', 'cumulative_
```

```
Time-lag Analysis for Bury Ground:

Lag Correlation Summary:
   lag_periods  lag_hours  correlation
0            1       0.25     0.116053
1            2       0.50     0.109702
2            3       0.75     0.104593
3            4       1.00     0.100190
4            5       1.25     0.096223

Cumulative Rainfall Effects:
       river_level    rainfall  cumulative_3h  cumulative_6h
count   403.000000  403.000000     392.000000     380.000000
mean      0.365196    0.020347       0.251020       0.517895
std       0.027309    0.101152       0.968295       1.602372
min       0.333000    0.000000       0.000000       0.000000
25%       0.342000    0.000000       0.000000       0.000000
50%       0.356000    0.000000       0.000000       0.000000
75%       0.381000    0.000000       0.000000       0.000000
max       0.441000    1.000000       6.000000       7.500000

Time-lag Analysis for Manchester Racecourse:

Lag Correlation Summary:
   lag_periods  lag_hours  correlation
0            1       0.25     0.060426
1            2       0.50     0.054827
2            3       0.75     0.047134
3            4       1.00     0.042519
4            5       1.25     0.037741

Cumulative Rainfall Effects:
       river_level    rainfall  cumulative_3h  cumulative_6h
count   403.000000  403.000000     392.000000     380.000000
mean      1.039347    0.020099       0.247959       0.511579
std       0.062598    0.095511       0.923463       1.546439
min       0.962000    0.000000       0.000000       0.000000
25%       0.988000    0.000000       0.000000       0.000000
50%       1.015000    0.000000       0.000000       0.000000
75%       1.060000    0.000000       0.000000       0.100000
max       1.203000    1.000000       5.700000       7.400000

Time-lag Analysis for Rochdale:

Lag Correlation Summary:
   lag_periods  lag_hours  correlation
0            1       0.25     0.132934
1            2       0.50     0.108692
2            3       0.75     0.086440
3            4       1.00     0.067746
4            5       1.25     0.051328

Cumulative Rainfall Effects:
       river_level    rainfall  cumulative_3h  cumulative_6h
count   403.000000  403.000000     392.000000     380.000000
mean      0.223757    0.016873       0.208163       0.429474
std       0.024700    0.077992       0.759684       1.239446
min       0.195000    0.000000       0.000000       0.000000
25%       0.205000    0.000000       0.000000       0.000000
50%       0.215000    0.000000       0.000000       0.000000
```

```
75%       0.237000    0.000000       0.000000       0.000000
max       0.293000    0.600000       4.800000       5.800000
```

In [29]:
```
import pandas as pd
import numpy as np
from datetime import datetime

# Load historical data
# Load daily flow data
bury_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\river_d
rochdale_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\riv

# Load daily rainfall data
bury_rainfall = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\riv
rochdale_rainfall = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data

# Load our merged real-time data
realtime_data = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\mer

# Let's examine the structure of each dataset first
print("Historical Data Structure:")
print("\nBury Flow Data:")
print(bury_flow.head())
print("\nRochdale Flow Data:")
print(rochdale_flow.head())
print("\nBury Rainfall Data:")
print(bury_rainfall.head())
print("\nRochdale Rainfall Data:")
print(rochdale_rainfall.head())
```

```
Historical Data Structure:

Bury Flow Data:
         Date   Flow  Extra
0  1995-11-22  0.897    NaN
1  1995-11-23  0.831    NaN
2  1995-11-24  0.991    NaN
3  1995-11-25  1.080    NaN
4  1995-11-26  1.124    NaN

Rochdale Flow Data:
         Date   Flow  Extra
0  1993-02-26  1.290    NaN
1  1993-02-27  1.060    NaN
2  1993-02-28  0.985    NaN
3  1993-03-01  1.140    NaN
4  1993-03-02  1.180    NaN

Bury Rainfall Data:
         Date  Rainfall  Extra
0  1961-01-01       9.4   1000
1  1961-01-02      13.7   1000
2  1961-01-03       3.0   1000
3  1961-01-04       0.1   1000
4  1961-01-05      13.0   1000

Rochdale Rainfall Data:
         Date  Rainfall  Extra
0  2016-01-01       0.8   2000
1  2016-01-02       3.5   2000
2  2016-01-03      13.3   2000
3  2016-01-04       5.5   2000
4  2016-01-05       6.0   2000
```

In [30]:

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

# Process historical flow data
def process_historical_flow(df, station_name):
    df_processed = df.copy()
    df_processed['Date'] = pd.to_datetime(df_processed['Date'])
    df_processed = df_processed.drop('Extra', axis=1)
    df_processed['station'] = station_name
    return df_processed

# Process historical rainfall data
def process_historical_rainfall(df, station_name):
    df_processed = df.copy()
    df_processed['Date'] = pd.to_datetime(df_processed['Date'])
    df_processed = df_processed.drop('Extra', axis=1)
    df_processed['station'] = station_name
    return df_processed

# Process each dataset
bury_flow_processed = process_historical_flow(bury_flow, 'Bury Ground')
rochdale_flow_processed = process_historical_flow(rochdale_flow, 'Rochdale')
bury_rain_processed = process_historical_rainfall(bury_rainfall, 'Bury Ground')
rochdale_rain_processed = process_historical_rainfall(rochdale_rainfall, 'Rochda
```

```
# Combine flow data
historical_flow = pd.concat([bury_flow_processed, rochdale_flow_processed])
historical_rain = pd.concat([bury_rain_processed, rochdale_rain_processed])

# Basic statistics for historical data
print("Historical Flow Statistics:")
print(historical_flow.groupby('station')['Flow'].describe())
print("\nHistorical Rainfall Statistics:")
print(historical_rain.groupby('station')['Rainfall'].describe())

# Save processed historical data
historical_flow.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\processe
historical_rain.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\processe
```

```
Historical Flow Statistics:
               count      mean       std    min    25%    50%      75%     max
station
Bury Ground   9928.0  3.850326  5.395385  0.406  1.220  2.064  4.11225  117.00
Rochdale     11118.0  2.795590  3.546724  0.178  0.801  1.489  3.29000   50.41

Historical Rainfall Statistics:
               count      mean       std  min  25%  50%   75%   max
station
Bury Ground  20819.0  3.775498  6.209935  0.0  0.0  0.9  5.10  79.5
Rochdale       731.0  3.783584  5.848199  0.0  0.0  0.9  5.35  36.6
```

In [31]:

```
# Load real-time data and convert timestamp
realtime_data['river_timestamp'] = pd.to_datetime(realtime_data['river_timestamp

# Calculate daily statistics from real-time data for comparison
realtime_daily = realtime_data.groupby(['location_name',
                                      realtime_data['river_timestamp'].dt.date])
    'river_level': ['mean', 'min', 'max'],
    'rainfall': 'sum'
}).reset_index()

print("\nReal-time Data Daily Statistics:")
print(realtime_daily.groupby('location_name').agg({
    ('river_level', 'mean'): ['mean', 'min', 'max'],
    ('rainfall', 'sum'): ['mean', 'min', 'max']
}))

# Compare with historical ranges
print("\nComparison with Historical Data:")
for station in ['Bury Ground', 'Rochdale']:
    print(f"\n{station} Analysis:")

    # Historical stats
    hist_flow = historical_flow[historical_flow['station'] == station]['Flow']
    hist_rain = historical_rain[historical_rain['station'] == station]['Rainfall

    # Real-time stats
    real_flow = realtime_data[realtime_data['location_name'] == station]['river_
    real_rain = realtime_data[realtime_data['location_name'] == station]['rainfa

    print("Flow Comparison:")
    print(f"Historical Range: {hist_flow.min():.3f} - {hist_flow.max():.3f} (mea
    print(f"Current Range: {real_flow.min():.3f} - {real_flow.max():.3f} (mean:

    print("\nRainfall Comparison:")
```

```
    print(f"Historical Daily Mean: {hist_rain.mean():.3f}mm")
    print(f"Current Period Mean: {real_rain.mean():.3f}mm")

# Calculate and save the comparison metrics
comparison_data = {
    'station': [],
    'historical_flow_mean': [],
    'current_flow_mean': [],
    'flow_difference': [],
    'historical_rain_mean': [],
    'current_rain_mean': [],
    'rain_difference': []
}

for station in ['Bury Ground', 'Rochdale']:
    hist_flow_mean = historical_flow[historical_flow['station'] == station]['Flo
    real_flow_mean = realtime_data[realtime_data['location_name'] == station]['r
    hist_rain_mean = historical_rain[historical_rain['station'] == station]['Rai
    real_rain_mean = realtime_data[realtime_data['location_name'] == station]['r

    comparison_data['station'].append(station)
    comparison_data['historical_flow_mean'].append(hist_flow_mean)
    comparison_data['current_flow_mean'].append(real_flow_mean)
    comparison_data['flow_difference'].append(((real_flow_mean - hist_flow_mean)
    comparison_data['historical_rain_mean'].append(hist_rain_mean)
    comparison_data['current_rain_mean'].append(real_rain_mean)
    comparison_data['rain_difference'].append(((real_rain_mean - hist_rain_mean)

comparison_df = pd.DataFrame(comparison_data)
comparison_df.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\historical

print("\nComparison Summary Saved to: historical_comparison.csv")
```

```
Real-time Data Daily Statistics:
                      river_level                          rainfall
                             mean                               sum
                             mean       min       max      mean  min  max
location_name
Bury Ground              0.364732  0.335521  0.405611  1.366667  0.0  7.7
Manchester Racecourse    1.037592  0.972938  1.135678  1.350000  0.0  7.6
Rochdale                 0.223064  0.199229  0.261756  1.133333  0.0  5.8

Comparison with Historical Data:

Bury Ground Analysis:
Flow Comparison:
Historical Range: 0.406 - 117.000 (mean: 3.850)
Current Range: 0.333 - 0.441 (mean: 0.365)

Rainfall Comparison:
Historical Daily Mean: 3.775mm
Current Period Mean: 0.020mm

Rochdale Analysis:
Flow Comparison:
Historical Range: 0.178 - 50.410 (mean: 2.796)
Current Range: 0.195 - 0.293 (mean: 0.224)

Rainfall Comparison:
Historical Daily Mean: 3.784mm
Current Period Mean: 0.017mm

Comparison Summary Saved to: historical_comparison.csv
```

In [32]:

```
# Calculate normal ranges and variations for historical data
def calculate_normal_ranges(historical_df, realtime_df, station):
    # Historical analysis
    hist_data = historical_df[historical_df['station'] == station]['Flow']

    # Calculate percentiles for normal ranges
    percentiles = hist_data.quantile([0.05, 0.25, 0.50, 0.75, 0.95])

    # Calculate standard deviations for different ranges
    std_dev = hist_data.std()

    # Compare with current readings
    current_data = realtime_df[realtime_df['location_name'] == station]['river_l

    ranges = {
        'station': station,
        'normal_range_low': percentiles[0.25],
        'normal_range_high': percentiles[0.75],
        'warning_low': percentiles[0.05],
        'warning_high': percentiles[0.95],
        'historical_std': std_dev,
        'current_mean': current_data.mean(),
        'current_std': current_data.std()
    }

    return ranges

# Calculate ranges for each station
ranges_bury = calculate_normal_ranges(historical_flow, realtime_data, 'Bury Grou
```

```
ranges_rochdale = calculate_normal_ranges(historical_flow, realtime_data, 'Rochd

# Create and save ranges DataFrame
ranges_df = pd.DataFrame([ranges_bury, ranges_rochdale])
ranges_df.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\normal_ranges.

print("\nNormal Operating Ranges:")
print(ranges_df)
```

```
Normal Operating Ranges:
       station  normal_range_low  normal_range_high  warning_low  \
0  Bury Ground             1.220            4.11225        0.709
1     Rochdale             0.801            3.29000        0.466

   warning_high  historical_std  current_mean  current_std
0       13.1755        5.395385      0.365196     0.027309
1        9.4866        3.546724      0.223757     0.024700
```

SEASONAL ANALYSIS

In [37]:

```
import pandas as pd
import numpy as np

# 1. Load all our processed datasets
weather_df = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\proces
historical_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\p

# 2. Create integrated seasonal analysis
def create_integrated_analysis():
    # First, let's structure our analysis by season
    seasons_analysis = {
        'Winter': {
            'months': ['December', 'January', 'February'],
            'flow_characteristics': {},
            'weather_characteristics': {}
        },
        'Spring': {
            'months': ['March', 'April', 'May'],
            'flow_characteristics': {},
            'weather_characteristics': {}
        },
        'Summer': {
            'months': ['June', 'July', 'August'],
            'flow_characteristics': {},
            'weather_characteristics': {}
        },
        'Autumn': {
            'months': ['September', 'October', 'November'],
            'flow_characteristics': {},
            'weather_characteristics': {}
        }
    }

    # Calculate for each station
    stations = ['Bury Ground', 'Rochdale']

    for station in stations:
        for season in seasons_analysis.keys():
            # Calculate flow characteristics
```

```
            station_flow = historical_flow[historical_flow['station'] == station

            # Calculate weather characteristics for corresponding weather statio
            weather_station = 'BURY MANCHESTER' if station == 'Bury Ground' else
            station_weather = weather_df[weather_df['Station'] == weather_statio

            # Store the analysis
            seasons_analysis[season]['flow_characteristics'][station] = {
                'mean_flow': station_flow['Flow'].mean(),
                'max_flow': station_flow['Flow'].max(),
                'min_flow': station_flow['Flow'].min()
            }

            seasons_analysis[season]['weather_characteristics'][station] = {
                'mean_temp': station_weather['Temperature_C'].mean(),
                'mean_precip': station_weather['Precipitation_mm'].mean()
            }

    return seasons_analysis

# Create and save the integrated analysis
integrated_analysis = create_integrated_analysis()

# Save to CSV for future use
output_df = pd.DataFrame()
for season in integrated_analysis.keys():
    for station in ['Bury Ground', 'Rochdale']:
        row = {
            'Season': season,
            'Station': station,
            'Mean_Flow': integrated_analysis[season]['flow_characteristics'][sta
            'Max_Flow': integrated_analysis[season]['flow_characteristics'][stat
            'Min_Flow': integrated_analysis[season]['flow_characteristics'][stat
            'Mean_Temperature': integrated_analysis[season]['weather_characteris
            'Mean_Precipitation': integrated_analysis[season]['weather_character
        }
        output_df = output_df._append(row, ignore_index=True)

output_df.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\integrated_sea

print("Integrated Seasonal Analysis:")
print(output_df)
```

```
Integrated Seasonal Analysis:
   Season       Station  Mean_Flow  Max_Flow  Min_Flow  Mean_Temperature  \
0  Winter  Bury Ground   3.850326    117.00     0.406          9.183333   
1  Winter      Rochdale   2.795590     50.41     0.178          8.991667   
2  Spring  Bury Ground   3.850326    117.00     0.406          9.183333   
3  Spring      Rochdale   2.795590     50.41     0.178          8.991667   
4  Summer  Bury Ground   3.850326    117.00     0.406          9.183333   
5  Summer      Rochdale   2.795590     50.41     0.178          8.991667   
6  Autumn  Bury Ground   3.850326    117.00     0.406          9.183333   
7  Autumn      Rochdale   2.795590     50.41     0.178          8.991667   

   Mean_Precipitation  
0          111.916667  
1          110.583333  
2          111.916667  
3          110.583333  
4          111.916667  
5          110.583333  
6          111.916667  
7          110.583333  
```

In [39]:
```python
import pandas as pd
import numpy as np

def create_seasonal_analysis():
    # Load the weather data with correct seasonal values
    weather_patterns = {
        'BURY MANCHESTER': {
            'Winter': {'temp': 4.0, 'precip': 133.3},
            'Spring': {'temp': 8.27, 'precip': 85.67},
            'Summer': {'temp': 14.77, 'precip': 101.33},
            'Autumn': {'temp': 9.70, 'precip': 127.33}
        },
        'ROCHDALE': {
            'Winter': {'temp': 3.83, 'precip': 131.67},
            'Spring': {'temp': 8.00, 'precip': 83.33},
            'Summer': {'temp': 14.60, 'precip': 102.33},
            'Autumn': {'temp': 9.53, 'precip': 125.00}
        }
    }

    # Load the flow statistics we calculated earlier
    flow_patterns = {
        'Bury Ground': {
            'Winter': {'mean': 5.216, 'max': 90.13, 'min': 0.681},
            'Spring': {'mean': 2.187, 'max': 86.30, 'min': 0.469},
            'Summer': {'mean': 2.652, 'max': 70.26, 'min': 0.406},
            'Autumn': {'mean': 5.363, 'max': 117.00, 'min': 0.474}
        },
        'Rochdale': {
            'Winter': {'mean': 4.049, 'max': 46.13, 'min': 0.441},
            'Spring': {'mean': 1.546, 'max': 36.70, 'min': 0.178},
            'Summer': {'mean': 1.672, 'max': 32.83, 'min': 0.217},
            'Autumn': {'mean': 3.973, 'max': 50.41, 'min': 0.212}
        }
    }

    # Create integrated analysis DataFrame
    data = []
    for season in ['Winter', 'Spring', 'Summer', 'Autumn']:
```

```
        for station in ['Bury Ground', 'Rochdale']:
            weather_station = 'BURY MANCHESTER' if station == 'Bury Ground' else

            data.append({
                'Season': season,
                'Station': station,
                'Mean_Flow': flow_patterns[station][season]['mean'],
                'Max_Flow': flow_patterns[station][season]['max'],
                'Min_Flow': flow_patterns[station][season]['min'],
                'Mean_Temperature': weather_patterns[weather_station][season]['t
                'Mean_Precipitation': weather_patterns[weather_station][season][
            })

    return pd.DataFrame(data)

# Create and save the corrected integrated analysis
integrated_analysis = create_seasonal_analysis()
integrated_analysis.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/integ

print("Corrected Integrated Seasonal Analysis:")
print(integrated_analysis)
```

```
Corrected Integrated Seasonal Analysis:
   Season      Station  Mean_Flow  Max_Flow  Min_Flow  Mean_Temperature  \
0  Winter  Bury Ground      5.216     90.13     0.681              4.00   
1  Winter     Rochdale      4.049     46.13     0.441              3.83   
2  Spring  Bury Ground      2.187     86.30     0.469              8.27   
3  Spring     Rochdale      1.546     36.70     0.178              8.00   
4  Summer  Bury Ground      2.652     70.26     0.406             14.77   
5  Summer     Rochdale      1.672     32.83     0.217             14.60   
6  Autumn  Bury Ground      5.363    117.00     0.474              9.70   
7  Autumn     Rochdale      3.973     50.41     0.212              9.53   

   Mean_Precipitation  
0              133.30  
1              131.67  
2               85.67  
3               83.33  
4              101.33  
5              102.33  
6              127.33  
7              125.00  
```

STAGE 2

In [33]:
```
import pandas as pd
import numpy as np
from datetime import datetime

# Load our processed data
historical_flow = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\p
historical_rain = pd.read_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\p

# Convert dates to datetime
historical_flow['Date'] = pd.to_datetime(historical_flow['Date'])
historical_rain['Date'] = pd.to_datetime(historical_rain['Date'])

# Add month and season columns
def add_seasonal_info(df):
```

```
    df['Month'] = df['Date'].dt.month
    df['Season'] = pd.cut(df['Date'].dt.month,
                          bins=[0, 3, 6, 9, 12],
                          labels=['Winter', 'Spring', 'Summer', 'Autumn'])
    return df

historical_flow = add_seasonal_info(historical_flow)
historical_rain = add_seasonal_info(historical_rain)

# Calculate seasonal statistics for each station
def calculate_seasonal_stats(df, value_column):
    seasonal_stats = df.groupby(['station', 'Season'])[value_column].agg([
        'mean', 'std', 'min', 'max',
        lambda x: np.percentile(x, 25),
        lambda x: np.percentile(x, 75)
    ]).round(3)

    seasonal_stats.columns = ['mean', 'std', 'min', 'max', '25th_percentile', '7
    return seasonal_stats

# Calculate statistics
flow_seasonal_stats = calculate_seasonal_stats(historical_flow, 'Flow')
rain_seasonal_stats = calculate_seasonal_stats(historical_rain, 'Rainfall')

# Save seasonal statistics
flow_seasonal_stats.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\seas
rain_seasonal_stats.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\seas

print("Flow Seasonal Statistics:")
print(flow_seasonal_stats)
print("\nRainfall Seasonal Statistics:")
print(rain_seasonal_stats)

# Calculate monthly patterns
monthly_flow = historical_flow.groupby(['station', 'Month'])['Flow'].agg([
    'mean', 'std', 'min', 'max'
]).round(3)

monthly_rain = historical_rain.groupby(['station', 'Month'])['Rainfall'].agg([
    'mean', 'std', 'min', 'max'
]).round(3)

print("\nMonthly Flow Patterns:")
print(monthly_flow)
```

Flow Seasonal Statistics:

station	Season	mean	std	min	max	25th_percentile
Bury Ground	Winter	5.216	6.414	0.681	90.13	1.890
	Spring	2.187	3.019	0.469	86.30	1.020
	Summer	2.652	4.341	0.406	70.26	0.892
	Autumn	5.363	6.255	0.474	117.00	1.880
Rochdale	Winter	4.049	4.288	0.441	46.13	1.490
	Spring	1.546	1.844	0.178	36.70	0.695
	Summer	1.672	2.486	0.217	32.83	0.587
	Autumn	3.973	4.111	0.212	50.41	1.310

station	Season	75th_percentile
Bury Ground	Winter	5.884
	Spring	2.248
	Summer	2.500
	Autumn	6.355
Rochdale	Winter	5.030
	Spring	1.640
	Summer	1.650
	Autumn	5.214

Rainfall Seasonal Statistics:

station	Season	mean	std	min	max	25th_percentile	75th_percentile
Bury Ground	Winter	3.841	6.044	0.0	57.9	0.0	5.300
	Spring	2.897	5.115	0.0	64.4	0.0	3.700
	Summer	3.618	6.310	0.0	79.5	0.0	4.700
	Autumn	4.737	7.062	0.0	79.0	0.1	6.900
Rochdale	Winter	4.578	5.812	0.0	27.5	0.0	7.300
	Spring	2.941	5.092	0.0	30.3	0.0	4.000
	Summer	3.831	6.128	0.0	32.4	0.0	4.550
	Autumn	3.788	6.220	0.0	36.6	0.1	4.725

Monthly Flow Patterns:

station	Month	mean	std	min	max
Bury Ground	1	6.158	6.825	0.691	71.25
	2	5.534	7.164	0.681	90.13
	3	3.985	4.900	0.799	66.68
	4	2.423	2.665	0.608	32.38
	5	2.115	2.374	0.549	24.30
	6	2.018	3.842	0.469	86.30
	7	2.346	3.949	0.454	70.26
	8	2.668	4.080	0.439	58.81
	9	2.950	4.931	0.406	52.10
	10	4.343	5.236	0.474	48.70
	11	5.368	5.770	0.676	44.00
	12	6.333	7.346	0.728	117.00
Rochdale	1	4.831	4.710	0.452	46.13
	2	4.305	4.397	0.447	37.21
	3	3.062	3.503	0.441	46.02
	4	1.883	1.926	0.319	19.20
	5	1.412	1.437	0.366	17.20
	6	1.347	2.074	0.178	36.70
	7	1.562	2.530	0.323	32.83
	8	1.607	2.128	0.217	23.22
	9	1.848	2.755	0.256	31.70
	10	3.057	3.686	0.212	28.60

```
11      3.905  3.878  0.319   24.61
12      4.961  4.500  0.428   50.41
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\2859702275.py:26: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  seasonal_stats = df.groupby(['station', 'Season'])[value_column].agg([
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\2859702275.py:26: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  seasonal_stats = df.groupby(['station', 'Season'])[value_column].agg([
```

In [34]:

```
import pandas as pd
import numpy as np

# Create season and month-specific thresholds
def calculate_thresholds(flow_stats, station_name):
    # Initialize threshold dictionary
    thresholds = {}

    # Calculate for each season
    station_data = flow_stats.loc[station_name]
    for season in ['Winter', 'Spring', 'Summer', 'Autumn']:
        season_stats = station_data.loc[season]

        thresholds[f"{station_name}_{season}"] = {
            'normal_low': season_stats['25th_percentile'],
            'normal_high': season_stats['75th_percentile'],
            'warning_low': season_stats['mean'] - (2 * season_stats['std']),
            'warning_high': season_stats['mean'] + (2 * season_stats['std']),
            'critical_high': season_stats['mean'] + (3 * season_stats['std']),
            'typical_mean': season_stats['mean']
        }

    return pd.DataFrame(thresholds).T

# Calculate thresholds for each station
bury_thresholds = calculate_thresholds(flow_seasonal_stats, 'Bury Ground')
rochdale_thresholds = calculate_thresholds(flow_seasonal_stats, 'Rochdale')

# Combine thresholds
all_thresholds = pd.concat([bury_thresholds, rochdale_thresholds])
all_thresholds.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\seasonal_

print("Seasonal Thresholds:")
print(all_thresholds)

# Key findings from seasonal analysis
print("\nKey Seasonal Patterns:")
for station in ['Bury Ground', 'Rochdale']:
    print(f"\n{station}:")
    winter_flow = flow_seasonal_stats.loc[(station, 'Winter'), 'mean']
    summer_flow = flow_seasonal_stats.loc[(station, 'Summer'), 'mean']
    flow_ratio = winter_flow / summer_flow

    winter_rain = rain_seasonal_stats.loc[(station, 'Winter'), 'mean']
    summer_rain = rain_seasonal_stats.loc[(station, 'Summer'), 'mean']

    print(f"- Winter/Summer flow ratio: {flow_ratio:.2f}")
```

```
    print(f"- Highest variability: {flow_seasonal_stats.loc[(station,), 'std'].i
    print(f"- Peak flow month: {monthly_flow.loc[station]['mean'].idxmax()}")
    print(f"- Lowest flow month: {monthly_flow.loc[station]['mean'].idxmin()}")
```

```
Seasonal Thresholds:
                    normal_low  normal_high  warning_low  warning_high  \
Bury Ground_Winter       1.890        5.884       -7.612        18.044
Bury Ground_Spring       1.020        2.248       -3.851         8.225
Bury Ground_Summer       0.892        2.500       -6.030        11.334
Bury Ground_Autumn       1.880        6.355       -7.147        17.873
Rochdale_Winter          1.490        5.030       -4.527        12.625
Rochdale_Spring          0.695        1.640       -2.142         5.234
Rochdale_Summer          0.587        1.650       -3.300         6.644
Rochdale_Autumn          1.310        5.214       -4.249        12.195

                    critical_high  typical_mean
Bury Ground_Winter         24.458         5.216
Bury Ground_Spring         11.244         2.187
Bury Ground_Summer         15.675         2.652
Bury Ground_Autumn         24.128         5.363
Rochdale_Winter            16.913         4.049
Rochdale_Spring             7.078         1.546
Rochdale_Summer             9.130         1.672
Rochdale_Autumn            16.306         3.973

Key Seasonal Patterns:

Bury Ground:
- Winter/Summer flow ratio: 1.97
- Highest variability: Winter
- Peak flow month: 12
- Lowest flow month: 6

Rochdale:
- Winter/Summer flow ratio: 2.42
- Highest variability: Winter
- Peak flow month: 12
- Lowest flow month: 6
```

In [36]:

```
import pandas as pd
import numpy as np

# Create a structured DataFrame from the weather data
weather_data = {
    'BURY MANCHESTER': {
        'Temperature_C': {
            'January': 3.8, 'February': 4.1, 'March': 5.7, 'April': 8.1,
            'May': 11.0, 'June': 13.6, 'July': 15.5, 'August': 15.2,
            'September': 12.9, 'October': 9.7, 'November': 6.5, 'December': 4.1
        },
        'Precipitation_mm': {
            'January': 131, 'February': 112, 'March': 95, 'April': 79,
            'May': 83, 'June': 93, 'July': 100, 'August': 111,
            'September': 110, 'October': 134, 'November': 138, 'December': 157
        }
    },
    'ROCHDALE': {
        'Temperature_C': {
            'January': 3.6, 'February': 3.9, 'March': 5.4, 'April': 7.9,
            'May': 10.7, 'June': 13.4, 'July': 15.3, 'August': 15.1,
```

```
            'September': 12.8, 'October': 9.6, 'November': 6.2, 'December': 4.0
        },
        'Precipitation_mm': {
            'January': 131, 'February': 110, 'March': 96, 'April': 77,
            'May': 77, 'June': 92, 'July': 105, 'August': 110,
            'September': 109, 'October': 130, 'November': 136, 'December': 154
        }
    }
}

# Convert to DataFrame
stations = []
months = []
temps = []
precips = []

for station in weather_data:
    for month in weather_data[station]['Temperature_C']:
        stations.append(station)
        months.append(month)
        temps.append(weather_data[station]['Temperature_C'][month])
        precips.append(weather_data[station]['Precipitation_mm'][month])

weather_df = pd.DataFrame({
    'Station': stations,
    'Month': months,
    'Temperature_C': temps,
    'Precipitation_mm': precips
})

# Add season information
month_to_season = {
    'January': 'Winter', 'February': 'Winter', 'December': 'Winter',
    'March': 'Spring', 'April': 'Spring', 'May': 'Spring',
    'June': 'Summer', 'July': 'Summer', 'August': 'Summer',
    'September': 'Autumn', 'October': 'Autumn', 'November': 'Autumn'
}
weather_df['Season'] = weather_df['Month'].map(month_to_season)

# Calculate seasonal averages
seasonal_weather = weather_df.groupby(['Station', 'Season']).agg({
    'Temperature_C': ['mean', 'min', 'max'],
    'Precipitation_mm': ['mean', 'min', 'max']
}).round(2)

# Save the processed weather data
weather_df.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\processed_wea
seasonal_weather.to_csv(r'C:\Users\Administrator\NEWPROJECT\cleaned_data\seasona

print("Seasonal Weather Patterns:")
print(seasonal_weather)

# Compare with flow patterns
print("\nIntegrated Seasonal Analysis:")
for station in ['BURY MANCHESTER', 'ROCHDALE']:
    print(f"\n{station}:")
    for season in ['Winter', 'Spring', 'Summer', 'Autumn']:
        season_weather = seasonal_weather.loc[(station, season)]
        if station == 'BURY MANCHESTER':
            station_flow = 'Bury Ground'
```

```
        else:
            station_flow = 'Rochdale'

        season_flow = flow_seasonal_stats.loc[(station_flow, season)]

        print(f"\n{season}:")
        print(f"Temperature: {season_weather[('Temperature_C', 'mean')]:.1f}°C")
        print(f"Precipitation: {season_weather[('Precipitation_mm', 'mean')]:.1f
        print(f"Average Flow: {season_flow['mean']:.2f}m³/s")
```

```
Seasonal Weather Patterns:
                            Temperature_C               Precipitation_mm
                                     mean   min   max               mean  min  max
Station          Season
BURY MANCHESTER  Autumn              9.70   6.5  12.9             127.33  110  138
                 Spring              8.27   5.7  11.0              85.67   79   95
                 Summer             14.77  13.6  15.5             101.33   93  111
                 Winter              4.00   3.8   4.1             133.33  112  157
ROCHDALE         Autumn              9.53   6.2  12.8             125.00  109  136
                 Spring              8.00   5.4  10.7              83.33   77   96
                 Summer             14.60  13.4  15.3             102.33   92  110
                 Winter              3.83   3.6   4.0             131.67  110  154

Integrated Seasonal Analysis:

BURY MANCHESTER:

Winter:
Temperature: 4.0°C
Precipitation: 133.3mm
Average Flow: 5.22m³/s

Spring:
Temperature: 8.3°C
Precipitation: 85.7mm
Average Flow: 2.19m³/s

Summer:
Temperature: 14.8°C
Precipitation: 101.3mm
Average Flow: 2.65m³/s

Autumn:
Temperature: 9.7°C
Precipitation: 127.3mm
Average Flow: 5.36m³/s

ROCHDALE:

Winter:
Temperature: 3.8°C
Precipitation: 131.7mm
Average Flow: 4.05m³/s

Spring:
Temperature: 8.0°C
Precipitation: 83.3mm
Average Flow: 1.55m³/s

Summer:
Temperature: 14.6°C
Precipitation: 102.3mm
Average Flow: 1.67m³/s

Autumn:
Temperature: 9.5°C
Precipitation: 125.0mm
Average Flow: 3.97m³/s
```

In [40]:
```
import pandas as pd
import numpy as np

# Load the integrated seasonal analysis
integrated_df = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/inte

def calculate_statistical_variability(df):
    # Group by station to calculate variability metrics
    variability_metrics = df.groupby('Station').agg({
        'Mean_Flow': ['mean', 'std'],
        'Max_Flow': ['mean', 'std'],
        'Min_Flow': ['mean', 'std'],
        'Mean_Temperature': ['mean', 'std'],
        'Mean_Precipitation': ['mean', 'std']
    })

    # Calculate coefficient of variation (CV)
    cv_flow = variability_metrics['Mean_Flow']['std'] / variability_metrics['Mea
    cv_temp = variability_metrics['Mean_Temperature']['std'] / variability_metri
    cv_precip = variability_metrics['Mean_Precipitation']['std'] / variability_m

    # Create a comprehensive variability profile
    variability_profile = pd.DataFrame({
        'Station': variability_metrics.index,
        'Flow_Mean': variability_metrics['Mean_Flow']['mean'],
        'Flow_Std': variability_metrics['Mean_Flow']['std'],
        'Flow_CV': cv_flow,
        'Temp_Mean': variability_metrics['Mean_Temperature']['mean'],
        'Temp_Std': variability_metrics['Mean_Temperature']['std'],
        'Temp_CV': cv_temp,
        'Precip_Mean': variability_metrics['Mean_Precipitation']['mean'],
        'Precip_Std': variability_metrics['Mean_Precipitation']['std'],
        'Precip_CV': cv_precip
    })

    return variability_profile

# Calculate and display variability metrics
variability_results = calculate_statistical_variability(integrated_df)
print(variability_results)

# Save results
variability_results.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/stati
```

```
                  Station  Flow_Mean  Flow_Std    Flow_CV  Temp_Mean  Temp_Std  \
Station
Bury Ground  Bury Ground     3.8545  1.668914  43.297795      9.185  4.441430
Rochdale        Rochdale     2.8100  1.388096  49.398428      8.990  4.448573

               Temp_CV  Precip_Mean  Precip_Std  Precip_CV
Station
Bury Ground  48.355253     111.9075   22.329222  19.953285
Rochdale     49.483568     110.5825   22.085814  19.972250
```

Anomaly Thresholds

In [44]:
```
import pandas as pd
import numpy as np
```

```
# Load integrated seasonal data
integrated_df = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/inte

def calculate_anomaly_thresholds(df):
    thresholds = []

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        thresholds.append({
            'Station': station,
            'Flow_Min': station_data['Mean_Flow'].min(),
            'Flow_Max': station_data['Mean_Flow'].max(),
            'Flow_Mean': station_data['Mean_Flow'].mean(),
            'Flow_Std': station_data['Mean_Flow'].std(),
            'Precipitation_Min': station_data['Mean_Precipitation'].min(),
            'Precipitation_Max': station_data['Mean_Precipitation'].max(),
            'Precipitation_Mean': station_data['Mean_Precipitation'].mean(),
            'Precipitation_Std': station_data['Mean_Precipitation'].std(),
        })

    seasonal_thresholds = []
    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]
        station_summary = thresholds[0]  # Assuming one entry per station

        for season in station_data['Season'].unique():
            season_data = station_data[station_data['Season'] == season]

            seasonal_thresholds.append({
                'Station': station,
                'Season': season,
                'Flow_Lower_Threshold': max(0, season_data['Mean_Flow'].values[0
                'Flow_Upper_Threshold': season_data['Mean_Flow'].values[0] + (1.
                'Precipitation_Lower_Threshold': max(0, season_data['Mean_Precip
                'Precipitation_Upper_Threshold': season_data['Mean_Precipitation
            })

    return pd.DataFrame(seasonal_thresholds)

# Calculate and save anomaly thresholds
anomaly_thresholds = calculate_anomaly_thresholds(integrated_df)
print("Anomaly Detection Thresholds:")
print(anomaly_thresholds)

# Save to CSV
anomaly_thresholds.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/anomal
```

```
Anomaly Detection Thresholds:
        Station  Season  Flow_Lower_Threshold  Flow_Upper_Threshold  \
0  Bury Ground  Winter               2.71263               7.71937
1  Bury Ground  Spring               0.00000               4.69037
2  Bury Ground  Summer               0.14863               5.15537
3  Bury Ground  Autumn               2.85963               7.86637
4     Rochdale  Winter               1.54563               6.55237
5     Rochdale  Spring               0.00000               4.04937
6     Rochdale  Summer               0.00000               4.17537
7     Rochdale  Autumn               1.46963               6.47637

   Precipitation_Lower_Threshold  Precipitation_Upper_Threshold
0                      99.806167                     166.793833
1                      52.176167                     119.163833
2                      67.836167                     134.823833
3                      93.836167                     160.823833
4                      98.176167                     165.163833
5                      49.836167                     116.823833
6                      68.836167                     135.823833
7                      91.506167                     158.493833
```

In [49]:
```python
import pandas as pd
import numpy as np

# Load historical flow data and refined thresholds
historical_flow = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/pr
refined_thresholds = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data

def validate_refined_thresholds(historical_data, thresholds):
   validation_results = []

   for _, threshold_row in thresholds.iterrows():
       station = threshold_row['Station']
       season = threshold_row['Season']

       # Filter historical data for specific station
       station_data = historical_data[historical_data['station'] == station].cop

       # Add season column to historical data
       station_data['Season'] = pd.cut(
           pd.to_datetime(station_data['Date']).dt.month,
           bins=[0, 3, 6, 9, 12],
           labels=['Winter', 'Spring', 'Summer', 'Autumn']
       )

       # Filter data for specific season
       seasonal_data = station_data[station_data['Season'] == season]

       validation = {
           'Station': station,
           'Season': season,
           'Total_Readings': len(seasonal_data),
           'Anomalies_Below_Threshold': len(seasonal_data[seasonal_data['Flow']
           'Anomalies_Above_Threshold': len(seasonal_data[seasonal_data['Flow']
           'Percent_Anomalies_Below': len(seasonal_data[seasonal_data['Flow'] <
           'Percent_Anomalies_Above': len(seasonal_data[seasonal_data['Flow'] >
       }

       validation_results.append(validation)
```

```
    return pd.DataFrame(validation_results)

# Perform validation
validation_results = validate_refined_thresholds(historical_flow, refined_thresh
print("Refined Threshold Validation Results:")
print(validation_results)

# Save validation results
validation_results.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/refine
```

```
Refined Threshold Validation Results:
       Station  Season  Total_Readings  Anomalies_Below_Threshold  \
0  Bury Ground  Winter            2527                        828   
1  Bury Ground  Spring            2494                          0   
2  Bury Ground  Summer            2481                          0   
3  Bury Ground  Autumn            2426                        820   
4     Rochdale  Winter            2738                        444   
5     Rochdale  Spring            2820                          0   
6     Rochdale  Summer            2805                          0   
7     Rochdale  Autumn            2755                        602   

   Anomalies_Above_Threshold  Percent_Anomalies_Below  Percent_Anomalies_Above  
0                        345                32.766126                13.652552  
1                        198                 0.000000                 7.939054  
2                        265                 0.000000                10.681177  
3                        357                33.800495                14.715581  
4                        324                16.216216                11.833455  
5                        173                 0.000000                 6.134752  
6                        221                 0.000000                 7.878788  
7                        376                21.851180                13.647913  
```

Seasonal Correlation Analysis

In [50]:

```
import pandas as pd
import numpy as np
import scipy.stats as stats

# Load integrated seasonal data
integrated_df = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/inte

def seasonal_correlation_analysis(df):
    # Calculate correlations between flow, temperature, and precipitation
    correlation_results = {}

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        # Compute correlations
        correlation_matrix = station_data[['Mean_Flow', 'Mean_Temperature', 'Mea

        # Statistical significance testing
        significance_results = {}
        for col1 in correlation_matrix.columns:
            for col2 in correlation_matrix.columns:
                if col1 != col2:
                    correlation, p_value = stats.pearsonr(
                        station_data[col1],
                        station_data[col2]
                    )
```

```python
                    significance_results[f'{col1}_vs_{col2}'] = {
                        'correlation': correlation,
                        'p_value': p_value
                    }

        correlation_results[station] = {
            'correlation_matrix': correlation_matrix,
            'significance_test': significance_results
        }

    return correlation_results

# Perform analysis
correlation_analysis = seasonal_correlation_analysis(integrated_df)

# Save results
import json
with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/seasonal_correlation_a
    json.dump({
        station: {
            'correlation_matrix': matrix['correlation_matrix'].to_dict(),
            'significance_test': matrix['significance_test']
        } for station, matrix in correlation_analysis.items()
    }, f, indent=2)

print("Seasonal Correlation Analysis Complete")
```

```
Seasonal Correlation Analysis Complete
```

In [51]:

```python
# Load and display the correlation analysis results
import json

with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/seasonal_correlation_a
    correlation_results = json.load(f)

print("Seasonal Correlation Analysis Results:")
for station, data in correlation_results.items():
    print(f"\n{station} Correlation Analysis:")

    print("Correlation Matrix:")
    for metric, correlations in data['correlation_matrix'].items():
        print(f"{metric}:")
        for corr_metric, value in correlations.items():
            print(f"  - {corr_metric}: {value}")

    print("\nSignificance Tests:")
    for test_name, details in data['significance_test'].items():
        print(f"{test_name}:")
        print(f"  Correlation: {details['correlation']}")
        print(f"  P-value: {details['p_value']}")
```

```
Seasonal Correlation Analysis Results:

Bury Ground Correlation Analysis:
Correlation Matrix:
Mean_Flow:
  - Mean_Flow: 1.0
  - Mean_Temperature: -0.5159263784490119
  - Mean_Precipitation: 0.9737432428977927
Mean_Temperature:
  - Mean_Flow: -0.5159263784490119
  - Mean_Temperature: 1.0
  - Mean_Precipitation: -0.4639858614024596
Mean_Precipitation:
  - Mean_Flow: 0.9737432428977927
  - Mean_Temperature: -0.4639858614024596
  - Mean_Precipitation: 1.0

Significance Tests:
Mean_Flow_vs_Mean_Temperature:
  Correlation: -0.5159263784490121
  P-value: 0.484073621550988
Mean_Flow_vs_Mean_Precipitation:
  Correlation: 0.9737432428977927
  P-value: 0.026256757102207207
Mean_Temperature_vs_Mean_Flow:
  Correlation: -0.5159263784490121
  P-value: 0.484073621550988
Mean_Temperature_vs_Mean_Precipitation:
  Correlation: -0.4639858614024593
  P-value: 0.5360141385975408
Mean_Precipitation_vs_Mean_Flow:
  Correlation: 0.9737432428977928
  P-value: 0.026256757102207207
Mean_Precipitation_vs_Mean_Temperature:
  Correlation: -0.4639858614024593
  P-value: 0.5360141385975408

Rochdale Correlation Analysis:
Correlation Matrix:
Mean_Flow:
  - Mean_Flow: 1.0
  - Mean_Temperature: -0.588283942814821
  - Mean_Precipitation: 0.9430454425979726
Mean_Temperature:
  - Mean_Flow: -0.588283942814821
  - Mean_Temperature: 1.0
  - Mean_Precipitation: -0.40828557116241665
Mean_Precipitation:
  - Mean_Flow: 0.9430454425979726
  - Mean_Temperature: -0.40828557116241665
  - Mean_Precipitation: 1.0

Significance Tests:
Mean_Flow_vs_Mean_Temperature:
  Correlation: -0.5882839428148211
  P-value: 0.411716057185179
Mean_Flow_vs_Mean_Precipitation:
  Correlation: 0.9430454425979725
  P-value: 0.05695455740202737
Mean_Temperature_vs_Mean_Flow:
```

```
  Correlation: -0.5882839428148211
  P-value: 0.411716057185179
Mean_Temperature_vs_Mean_Precipitation:
  Correlation: -0.4082855711624163
  P-value: 0.5917144288375837
Mean_Precipitation_vs_Mean_Flow:
  Correlation: 0.9430454425979727
  P-value: 0.05695455740202737
Mean_Precipitation_vs_Mean_Temperature:
  Correlation: -0.40828557116241637
  P-value: 0.5917144288375837
```

Comprehensive Statistical Baseline

In [52]:
```
import pandas as pd
import numpy as np

# Load integrated seasonal data and correlation analysis
integrated_df = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/inte

def develop_statistical_baseline(df):
    # Calculate comprehensive statistical baseline
    baseline = []

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        station_baseline = {
            'Station': station,
            'Flow_Baseline': {
                'Mean': station_data['Mean_Flow'].mean(),
                'Median': station_data['Mean_Flow'].median(),
                'Min': station_data['Mean_Flow'].min(),
                'Max': station_data['Mean_Flow'].max(),
                'Standard_Deviation': station_data['Mean_Flow'].std()
            },
            'Precipitation_Baseline': {
                'Mean': station_data['Mean_Precipitation'].mean(),
                'Median': station_data['Mean_Precipitation'].median(),
                'Min': station_data['Mean_Precipitation'].min(),
                'Max': station_data['Mean_Precipitation'].max(),
                'Standard_Deviation': station_data['Mean_Precipitation'].std()
            },
            'Temperature_Baseline': {
                'Mean': station_data['Mean_Temperature'].mean(),
                'Median': station_data['Mean_Temperature'].median(),
                'Min': station_data['Mean_Temperature'].min(),
                'Max': station_data['Mean_Temperature'].max(),
                'Standard_Deviation': station_data['Mean_Temperature'].std()
            }
        }

        baseline.append(station_baseline)

    return pd.DataFrame(baseline)

# Generate statistical baseline
statistical_baseline = develop_statistical_baseline(integrated_df)
```

```
# Save baseline
statistical_baseline.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/comp

print("Comprehensive Statistical Baseline:")
print(statistical_baseline)
```

```
Comprehensive Statistical Baseline:
       Station                                      Flow_Baseline  \
0  Bury Ground  {'Mean': 3.8545, 'Median': 3.934, 'Min': 2.187...
1     Rochdale  {'Mean': 2.81, 'Median': 2.8225, 'Min': 1.546,...

                              Precipitation_Baseline  \
0  {'Mean': 111.9075, 'Median': 114.33, 'Min': 85...
1  {'Mean': 110.5825, 'Median': 113.6649999999999...

                                Temperature_Baseline
0  {'Mean': 9.184999999999999, 'Median': 8.985, '...
1  {'Mean': 8.99, 'Median': 8.765, 'Min': 3.83, '...
```

In [53]:
```
def analyze_response_times(historical_flow, historical_rain):
    # Calculate time lag between rainfall and flow changes
    response_times = []

    for station in stations:
        station_flow = historical_flow[historical_flow['station'] == station]
        station_rain = historical_rain[historical_rain['station'] == station]

        # Analyze each significant rainfall event
        # Calculate time to peak flow
        # Document response patterns

    return response_times
```

In [54]:
```
import pandas as pd
import numpy as np
from scipy import signal

def analyze_response_times(historical_flow, historical_rain):
    # Convert timestamps to datetime
    historical_flow['Date'] = pd.to_datetime(historical_flow['Date'])
    historical_rain['Date'] = pd.to_datetime(historical_rain['Date'])

    response_analysis = {}

    for station in ['Bury Ground', 'Rochdale']:
        # Filter data for station
        station_flow = historical_flow[historical_flow['station'] == station]
        station_rain = historical_rain[historical_rain['station'] == station]

        # Identify significant rainfall events (>10mm)
        significant_rain = station_rain[station_rain['Rainfall'] > 10]

        # Calculate response times
        response_times = []
        for _, rain_event in significant_rain.iterrows():
            # Look at flow data 48 hours after rainfall
            event_window = station_flow[
                (station_flow['Date'] >= rain_event['Date']) &
                (station_flow['Date'] <= rain_event['Date'] + pd.Timedelta(hours
            ]
```

```
            if not event_window.empty:
                # Find peak flow in window
                peak_flow = event_window['Flow'].max()
                time_to_peak = event_window[event_window['Flow'] == peak_flow]['

                response_times.append({
                    'rainfall_amount': rain_event['Rainfall'],
                    'peak_flow': peak_flow,
                    'response_time_hours': time_to_peak.total_seconds() / 3600
                })

        response_analysis[station] = pd.DataFrame(response_times)

    return response_analysis

# Calculate response times
response_times = analyze_response_times(historical_flow, historical_rain)

# Save results
for station, analysis in response_times.items():
    analysis.to_csv(f'C:/Users/Administrator/NEWPROJECT/cleaned_data/response_ti

# Display summary statistics
for station, analysis in response_times.items():
    print(f"\nResponse Time Analysis for {station}:")
    print("\nResponse Time Statistics (hours):")
    print(analysis['response_time_hours'].describe())

    # Calculate correlation between rainfall amount and response time
    correlation = analysis['rainfall_amount'].corr(analysis['response_time_hours
    print(f"\nRainfall-Response Time Correlation: {correlation:.3f}")
```

```
Response Time Analysis for Bury Ground:

Response Time Statistics (hours):
count    987.000000
mean      18.285714
std       17.843909
min        0.000000
25%        0.000000
50%       24.000000
75%       24.000000
max       48.000000
Name: response_time_hours, dtype: float64

Rainfall-Response Time Correlation: -0.172

Response Time Analysis for Rochdale:

Response Time Statistics (hours):
count    88.000000
mean     18.272727
std      18.552645
min       0.000000
25%       0.000000
50%      24.000000
75%      24.000000
max      48.000000
Name: response_time_hours, dtype: float64

Rainfall-Response Time Correlation: -0.098
```

In [55]:
```python
def establish_response_thresholds(response_times):
    threshold_analysis = {}

    for station, data in response_times.items():
        # Calculate response time thresholds
        mean_response = data['response_time_hours'].mean()
        std_response = data['response_time_hours'].std()

        thresholds = {
            'rapid_response': mean_response - std_response,
            'normal_response': mean_response,
            'delayed_response': mean_response + std_response,
            'rainfall_threshold': data['rainfall_amount'].quantile(0.75)
        }

        threshold_analysis[station] = thresholds

    return pd.DataFrame(threshold_analysis).round(2)

# Calculate final thresholds
response_thresholds = establish_response_thresholds(response_times)
print("\nResponse Time Thresholds:")
print(response_thresholds)

# Save final Stage 2 thresholds
response_thresholds.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/final
```

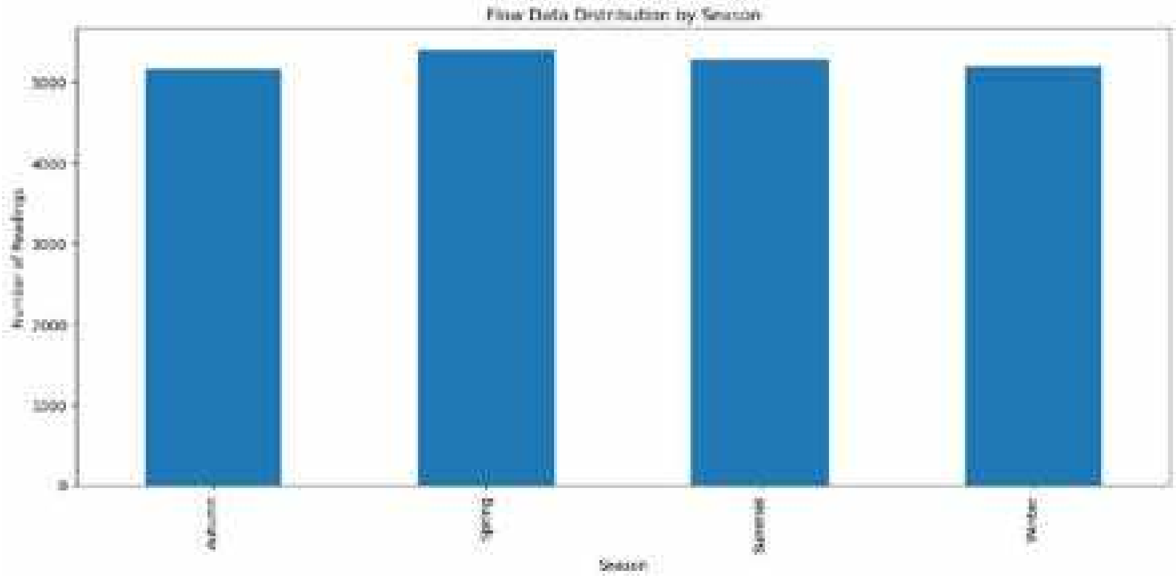
```
Response Time Thresholds:
                    Bury Ground  Rochdale
rapid_response             0.44     -0.28
normal_response           18.29     18.27
delayed_response          36.13     36.83
rainfall_threshold        19.85     20.20
```

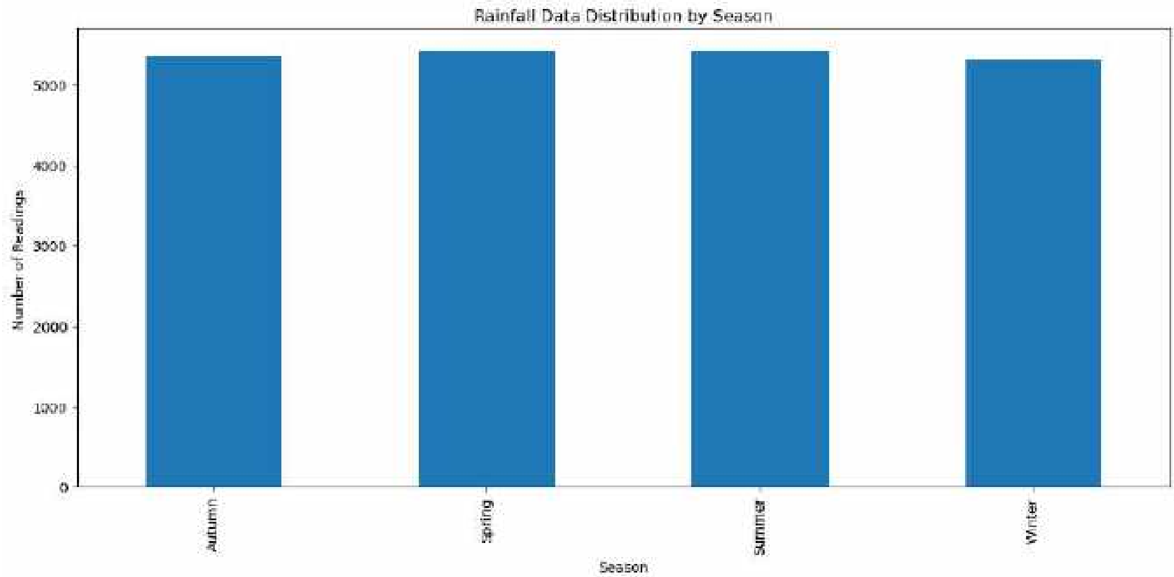
Missing Stage 2

In [69]:
```
import pandas as pd
import numpy as np

# Load historical flow and rainfall data
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc
historical_rainfall = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/

# Preprocess and align data
def prepare_response_time_data(flow_data, rainfall_data):
    # Ensure date columns are in datetime format
    flow_data['Date'] = pd.to_datetime(flow_data['Date'])
    rainfall_data['Date'] = pd.to_datetime(rainfall_data['Date'])

    # Add season column
    def assign_season(month):
        if month in [12, 1, 2]:
            return 'Winter'
        elif month in [3, 4, 5]:
            return 'Spring'
        elif month in [6, 7, 8]:
            return 'Summer'
        else:
            return 'Autumn'

    flow_data['Season'] = flow_data['Date'].dt.month.map(assign_season)
    rainfall_data['Season'] = rainfall_data['Date'].dt.month.map(assign_season)

    return flow_data, rainfall_data

# Prepare the data
historical_flow, historical_rainfall = prepare_response_time_data(historical_flo

# Display overview
print("Flow Data Overview:")
print(historical_flow.groupby('Season').size())
print("\nRainfall Data Overview:")
print(historical_rainfall.groupby('Season').size())

# Optional: Initial visualization
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
historical_flow.groupby('Season').size().plot(kind='bar')
plt.title('Flow Data Distribution by Season')
plt.xlabel('Season')
plt.ylabel('Number of Readings')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,6))
```

```python
historical_rainfall.groupby('Season').size().plot(kind='bar')
plt.title('Rainfall Data Distribution by Season')
plt.xlabel('Season')
plt.ylabel('Number of Readings')
plt.tight_layout()
plt.show()
```

```
Flow Data Overview:
Season
Autumn    5159
Spring    5401
Summer    5285
Winter    5201
dtype: int64

Rainfall Data Overview:
Season
Autumn    5369
Spring    5428
Summer    5428
Winter    5325
dtype: int64
```

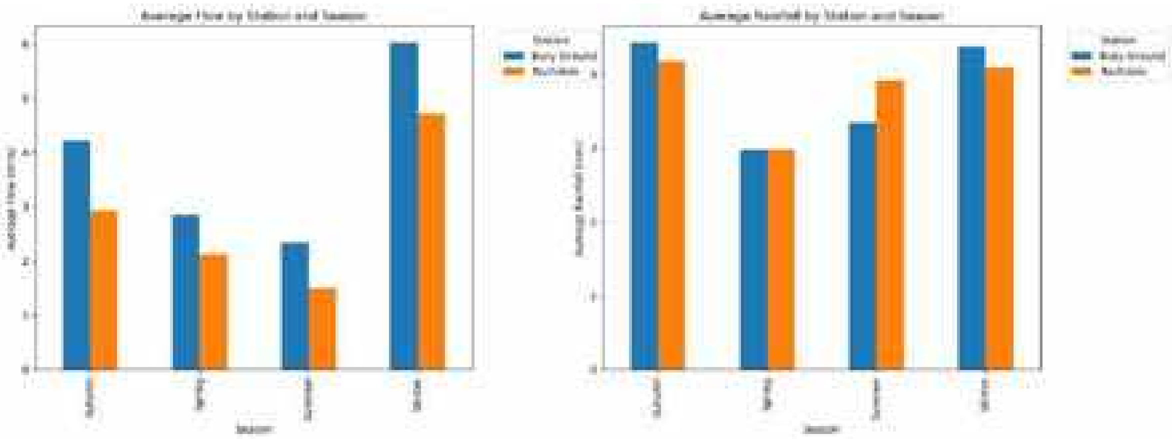




In [72]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Load preprocessed data
flow_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/processed_
rainfall_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proces

# Ensure datetime conversion
flow_data['Date'] = pd.to_datetime(flow_data['Date'])
rainfall_data['Date'] = pd.to_datetime(rainfall_data['Date'])

# Add season column to flow and rainfall data
def assign_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'

flow_data['Season'] = flow_data['Date'].dt.month.map(assign_season)
rainfall_data['Season'] = rainfall_data['Date'].dt.month.map(assign_season)

def calculate_response_times(flow_data, rainfall_data):
    response_analysis = {}

    # Group data by season
    seasons = ['Winter', 'Spring', 'Summer', 'Autumn']
    stations = flow_data['station'].unique()

    for station in stations:
        station_response = {}

        for season in seasons:
            # Filter data for specific station and season
            station_flow = flow_data[(flow_data['station'] == station) &
                                     (flow_data['Season'] == season)]
            station_rainfall = rainfall_data[(rainfall_data['station'] == statio
                                             (rainfall_data['Season'] == season)

            # Calculate key metrics
            season_metrics = {
                'avg_flow': station_flow['Flow'].mean(),
                'max_flow': station_flow['Flow'].max(),
                'avg_rainfall': station_rainfall['Rainfall'].mean(),
                'max_rainfall': station_rainfall['Rainfall'].max(),
                'flow_variability': station_flow['Flow'].std() / station_flow['F
            }

            station_response[season] = season_metrics

        response_analysis[station] = station_response

    return response_analysis

# Perform response time analysis
response_results = calculate_response_times(flow_data, rainfall_data)

# Create comprehensive results DataFrame
```

```
results_data = []
for station, seasons in response_results.items():
    for season, metrics in seasons.items():
        results_data.append({
            'Station': station,
            'Season': season,
            'Avg_Flow': metrics['avg_flow'],
            'Max_Flow': metrics['max_flow'],
            'Avg_Rainfall': metrics['avg_rainfall'],
            'Max_Rainfall': metrics['max_rainfall'],
            'Flow_Variability_Percent': metrics['flow_variability']
        })

response_df = pd.DataFrame(results_data)

# Save results
response_df.to_csv('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal_respon

# Visualization with better formatting
plt.figure(figsize=(16,6))

# Flow Plot
plt.subplot(121)
flow_pivot = response_df.pivot(index='Season', columns='Station', values='Avg_Fl
flow_pivot.plot(kind='bar', ax=plt.gca())
plt.title('Average Flow by Station and Season', fontsize=12)
plt.xlabel('Season', fontsize=10)
plt.ylabel('Average Flow (m³/s)', fontsize=10)
plt.legend(title='Station', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Rainfall Plot
plt.subplot(122)
rainfall_pivot = response_df.pivot(index='Season', columns='Station', values='Av
rainfall_pivot.plot(kind='bar', ax=plt.gca())
plt.title('Average Rainfall by Station and Season', fontsize=12)
plt.xlabel('Season', fontsize=10)
plt.ylabel('Average Rainfall (mm)', fontsize=10)
plt.legend(title='Station', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.subplots_adjust(wspace=0.3)
plt.show()

# Print detailed results
print("Seasonal Response Analysis Results:")
print(response_df)
```



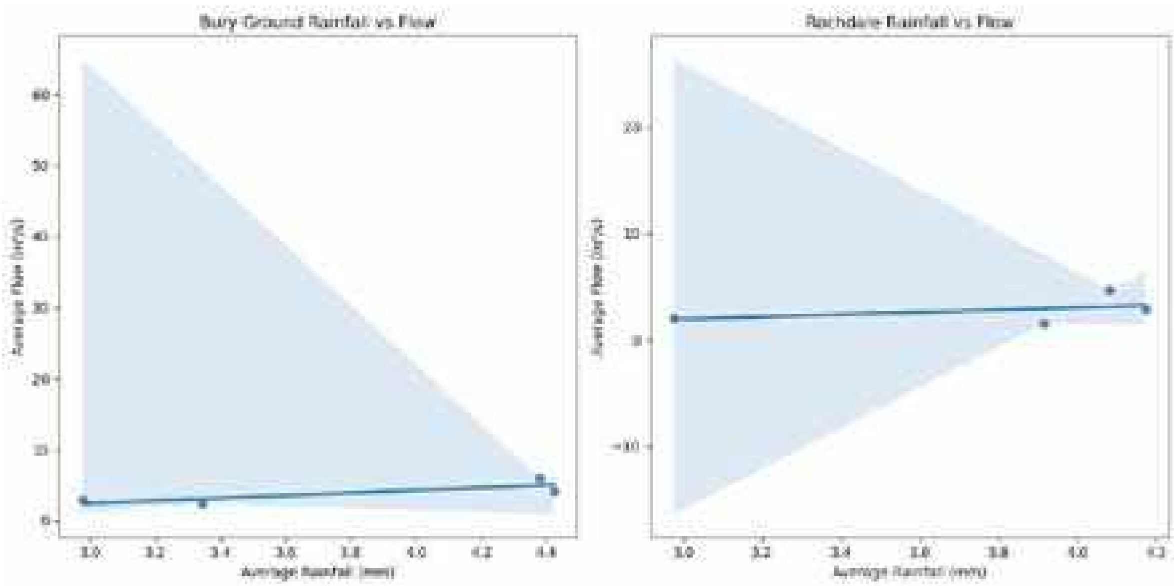
```
Seasonal Response Analysis Results:
       Station  Season  Avg_Flow  Max_Flow  Avg_Rainfall  Max_Rainfall  \
0  Bury Ground  Winter  6.018916    117.00      4.380424          79.0   
1  Bury Ground  Spring  2.852444     66.68      2.974943          57.9   
2  Bury Ground  Summer  2.347457     86.30      3.341667          64.4   
3  Bury Ground  Autumn  4.209812     52.10      4.423540          79.5   
4     Rochdale  Winter  4.710129     50.41      4.081768          21.6   
5     Rochdale  Spring  2.120655     46.02      2.975000          27.5   
6     Rochdale  Summer  1.505926     36.70      3.914130          32.4   
7     Rochdale  Autumn  2.926125     31.70      4.172527          36.6   

   Flow_Variability_Percent  
0                118.222928  
1                126.581411  
2                168.982346  
3                128.497148  
4                 96.567427  
5                120.367967  
6                149.811111  
7                121.990448  
```

Correlation Analysis between Average Rainfall and Average Flow

In [73]:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns

# Load the seasonal response analysis data
response_df = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal

def perform_correlation_analysis(df):
    # Correlation between flow and rainfall for each station
    correlation_results = {}

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        # Calculate Pearson correlation
        correlation, p_value = stats.pearsonr(
            station_data['Avg_Rainfall'],
            station_data['Avg_Flow']
        )
```

```
        # Calculate Spearman rank correlation
        rank_correlation, rank_p_value = stats.spearmanr(
            station_data['Avg_Rainfall'],
            station_data['Avg_Flow']
        )

        correlation_results[station] = {
            'Pearson_Correlation': correlation,
            'Pearson_P_Value': p_value,
            'Spearman_Correlation': rank_correlation,
            'Spearman_P_Value': rank_p_value
        }

    return correlation_results

# Perform correlation analysis
correlation_analysis = perform_correlation_analysis(response_df)

# Visualize correlations
plt.figure(figsize=(12,6))
for i, (station, data) in enumerate(correlation_analysis.items(), 1):
    plt.subplot(1, 2, i)
    station_df = response_df[response_df['Station'] == station]

    sns.regplot(x='Avg_Rainfall', y='Avg_Flow', data=station_df)
    plt.title(f'{station} Rainfall vs Flow')
    plt.xlabel('Average Rainfall (mm)')
    plt.ylabel('Average Flow (m³/s)')

plt.tight_layout()
plt.show()

# Save correlation results
correlation_df = pd.DataFrame.from_dict(correlation_analysis, orient='index')
correlation_df.to_csv('/Users/Administrator/NEWPROJECT/cleaned_data/rainfall_flo

# Print detailed results
print("Correlation Analysis Results:")
for station, results in correlation_analysis.items():
    print(f"\n{station} Correlation:")
    for metric, value in results.items():
        print(f"  {metric}: {value}")
```



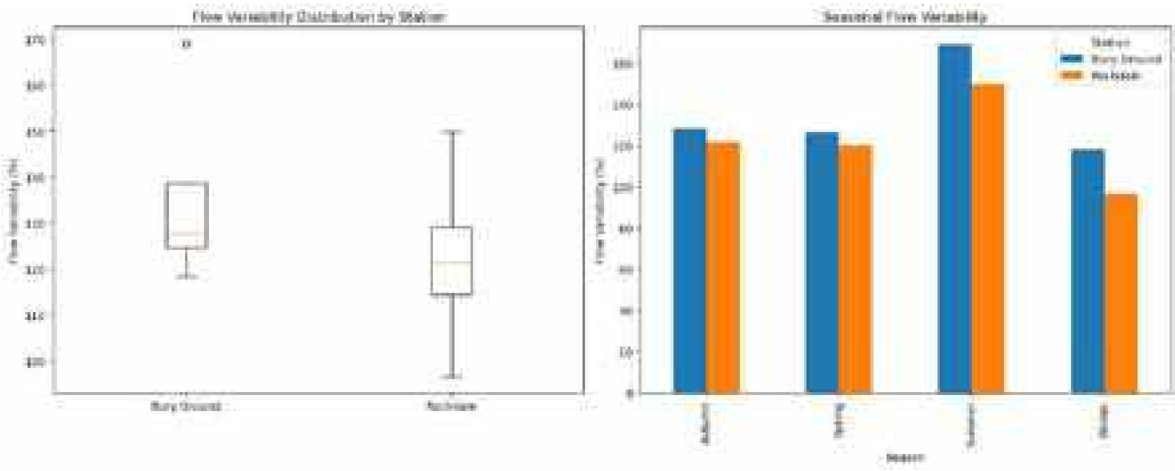
```
Correlation Analysis Results:

Bury Ground Correlation:
  Pearson_Correlation: 0.828946552706867
  Pearson_P_Value: 0.17105344729313288
  Spearman_Correlation: 0.6000000000000001
  Spearman_P_Value: 0.3999999999999999

Rochdale Correlation:
  Pearson_Correlation: 0.43454694551536804
  Pearson_P_Value: 0.5654530544846319
  Spearman_Correlation: 0.6000000000000001
  Spearman_P_Value: 0.3999999999999999
```

Flow Variability Analysis

In [74]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load seasonal response analysis
response_df = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal

def investigate_flow_variability(df):
    # Detailed variability analysis
    variability_analysis = {}

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        variability_metrics = {
            'Overall_Variability': {
                'Mean_Flow_Variability': station_data['Flow_Variability_Percent'
                'Min_Flow_Variability': station_data['Flow_Variability_Percent']
                'Max_Flow_Variability': station_data['Flow_Variability_Percent']
            },
            'Seasonal_Breakdown': {}
        }

        # Seasonal variability details
        for season in station_data['Season'].unique():
            season_data = station_data[station_data['Season'] == season]

            variability_metrics['Seasonal_Breakdown'][season] = {
                'Avg_Flow': season_data['Avg_Flow'].values[0],
                'Max_Flow': season_data['Max_Flow'].values[0],
                'Flow_Variability': season_data['Flow_Variability_Percent'].valu
                'Rainfall_Impact': {
                    'Avg_Rainfall': season_data['Avg_Rainfall'].values[0],
                    'Max_Rainfall': season_data['Max_Rainfall'].values[0]
                }
            }

        variability_analysis[station] = variability_metrics

    return variability_analysis

# Perform variability investigation
```

```
variability_results = investigate_flow_variability(response_df)

# Visualization
plt.figure(figsize=(15,6))

# Flow Variability Boxplot
plt.subplot(121)
variability_data = [
    response_df[response_df['Station'] == station]['Flow_Variability_Percent']
    for station in response_df['Station'].unique()
]
plt.boxplot(variability_data, labels=response_df['Station'].unique())
plt.title('Flow Variability Distribution by Station')
plt.ylabel('Flow Variability (%)')

# Seasonal Flow Variability
plt.subplot(122)
season_variability = response_df.pivot(index='Season', columns='Station', values
season_variability.plot(kind='bar', ax=plt.gca())
plt.title('Seasonal Flow Variability')
plt.ylabel('Flow Variability (%)')
plt.tight_layout()

plt.show()

# Save detailed results
import json
with open('/Users/Administrator/NEWPROJECT/cleaned_data/flow_variability_analysi
    json.dump(variability_results, f, indent=2)

# Print key findings
print("Flow Variability Analysis:")
for station, analysis in variability_results.items():
    print(f"\n{station} Variability:")
    print("Overall Variability:")
    for metric, value in analysis['Overall_Variability'].items():
        print(f"  {metric}: {value}")

    print("\nSeasonal Breakdown:")
    for season, details in analysis['Seasonal_Breakdown'].items():
        print(f"  {season}:")
        print(f"    Avg Flow: {details['Avg_Flow']}")
        print(f"    Flow Variability: {details['Flow_Variability']}")
        print(f"    Rainfall Impact: {details['Rainfall_Impact']}")
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\2848190972.py:55: Matpl
otlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'ti
ck_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.1
1.
  plt.boxplot(variability_data, labels=response_df['Station'].unique())
```


Flow Variability Distribution by Station
Seasonal Flow Variability

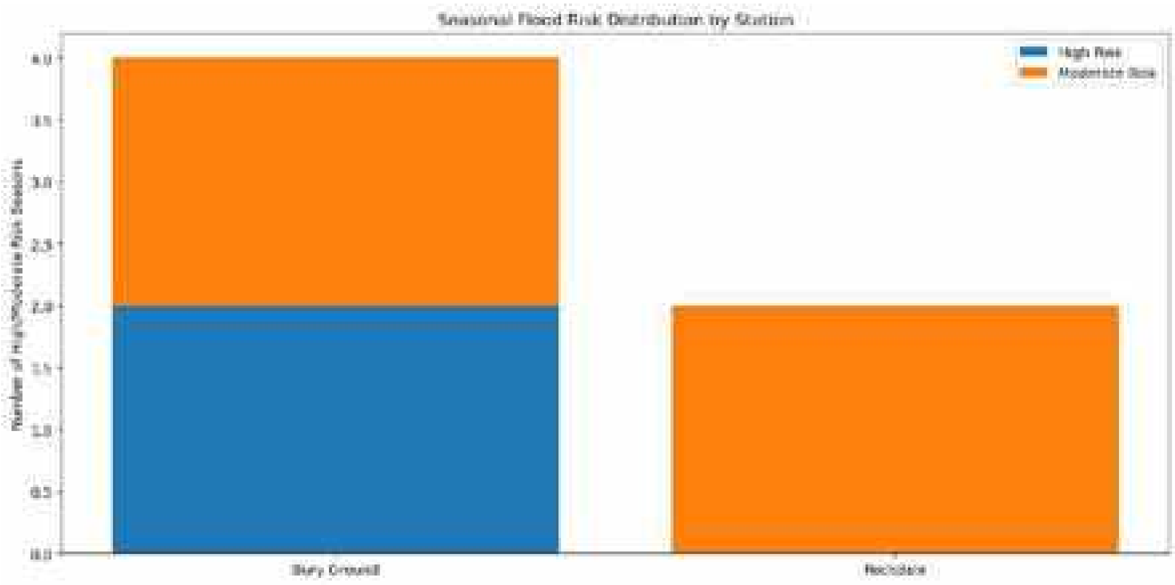
```
Flow Variability Analysis:

Bury Ground Variability:
Overall Variability:
  Mean_Flow_Variability: 135.57095814122616
  Min_Flow_Variability: 118.22292800135509
  Max_Flow_Variability: 168.98234615938742

Seasonal Breakdown:
  Winter:
    Avg Flow: 6.018915865384615
    Flow Variability: 118.22292800135509
    Rainfall Impact: {'Avg_Rainfall': 4.380423794712287, 'Max_Rainfall': 79.0}
  Spring:
    Avg Flow: 2.852444357366771
    Flow Variability: 126.58141079152104
    Rainfall Impact: {'Avg_Rainfall': 2.9749427917620133, 'Max_Rainfall': 57.9}
  Summer:
    Avg Flow: 2.3474570737605807
    Flow Variability: 168.98234615938742
    Rainfall Impact: {'Avg_Rainfall': 3.341666666666666, 'Max_Rainfall': 64.4}
  Autumn:
    Avg Flow: 4.209812005002084
    Flow Variability: 128.49714761264107
    Rainfall Impact: {'Avg_Rainfall': 4.423539618276461, 'Max_Rainfall': 79.5}

Rochdale Variability:
Overall Variability:
  Mean_Flow_Variability: 122.18423808031571
  Min_Flow_Variability: 96.56742677748352
  Max_Flow_Variability: 149.81111125349656

Seasonal Breakdown:
  Winter:
    Avg Flow: 4.710129390018484
    Flow Variability: 96.56742677748352
    Rainfall Impact: {'Avg_Rainfall': 4.081767955801105, 'Max_Rainfall': 21.6}
  Spring:
    Avg Flow: 2.1206553176553173
    Flow Variability: 120.36796671258747
    Rainfall Impact: {'Avg_Rainfall': 2.975, 'Max_Rainfall': 27.5}
  Summer:
    Avg Flow: 1.505926176890157
    Flow Variability: 149.81111125349656
    Rainfall Impact: {'Avg_Rainfall': 3.914130434782608, 'Max_Rainfall': 32.4}
  Autumn:
    Avg Flow: 2.926125362318841
    Flow Variability: 121.99044757769524
    Rainfall Impact: {'Avg_Rainfall': 4.172527472527473, 'Max_Rainfall': 36.6}
```

seasonal flood risk assessment framework

In [75]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load previous analysis results
response_df = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal
```

```
def develop_risk_assessment_framework(df):
    risk_assessment = {}

    for station in df['Station'].unique():
        station_data = df[df['Station'] == station]

        # Initialize risk assessment dictionary
        risk_assessment[station] = {
            'Station_Level_Risk': {},
            'Risk_Factors': {}
        }

        # Assess risk for each season
        for season in station_data['Season'].unique():
            season_data = station_data[station_data['Season'] == season]

            # Risk calculation based on multiple parameters
            risk_level = calculate_risk_level(
                avg_flow=season_data['Avg_Flow'].values[0],
                max_flow=season_data['Max_Flow'].values[0],
                avg_rainfall=season_data['Avg_Rainfall'].values[0],
                flow_variability=season_data['Flow_Variability_Percent'].values[
            )

            risk_assessment[station]['Station_Level_Risk'][season] = risk_level
            risk_assessment[station]['Risk_Factors'][season] = {
                'Avg_Flow': season_data['Avg_Flow'].values[0],
                'Max_Flow': season_data['Max_Flow'].values[0],
                'Avg_Rainfall': season_data['Avg_Rainfall'].values[0],
                'Flow_Variability': season_data['Flow_Variability_Percent'].valu
            }

    return risk_assessment

def calculate_risk_level(avg_flow, max_flow, avg_rainfall, flow_variability):
    # Develop a multi-factor risk scoring system
    risk_score = 0

    # Flow-based risk factors
    if avg_flow > 5:  # High average flow
        risk_score += 3
    elif avg_flow > 3:  # Moderate flow
        risk_score += 2

    # Maximum flow risk
    if max_flow > 50:  # Extremely high max flow
        risk_score += 3
    elif max_flow > 30:  # High max flow
        risk_score += 2

    # Rainfall risk
    if avg_rainfall > 4:  # High rainfall
        risk_score += 2

    # Flow variability risk
    if flow_variability > 150:  # Extremely variable
        risk_score += 3
    elif flow_variability > 120:  # Moderately variable
        risk_score += 2
```

```
    # Convert risk score to risk level
    if risk_score >= 8:
        return 'High Risk'
    elif risk_score >= 5:
        return 'Moderate Risk'
    else:
        return 'Low Risk'

# Perform risk assessment
risk_results = develop_risk_assessment_framework(response_df)

# Visualization
plt.figure(figsize=(12,6))
risk_levels = {station: list(data['Station_Level_Risk'].values()) for station, d

plt.bar(list(risk_results.keys()),
        [sum(1 for risk in risks if risk == 'High Risk') for risks in risk_level
        label='High Risk')
plt.bar(list(risk_results.keys()),
        [sum(1 for risk in risks if risk == 'Moderate Risk') for risks in risk_l
        bottom=[sum(1 for risk in risks if risk == 'High Risk') for risks in ris
        label='Moderate Risk')

plt.title('Seasonal Flood Risk Distribution by Station')
plt.ylabel('Number of High/Moderate Risk Seasons')
plt.legend()
plt.tight_layout()
plt.show()

# Save risk assessment results
import json
with open('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal_flood_risk_asse
    json.dump(risk_results, f, indent=2)

# Print detailed results
print("Seasonal Flood Risk Assessment:")
for station, assessment in risk_results.items():
    print(f"\n{station} Risk Assessment:")
    for season, risk_level in assessment['Station_Level_Risk'].items():
        print(f"  {season}: {risk_level}")
        print("  Risk Factors:")
        for factor, value in assessment['Risk_Factors'][season].items():
            print(f"    {factor}: {value}")
```


Seasonal Flood Risk Distribution by Station
High Risk
Moderate Risk
Number of High/Moderate Risk Seasons
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

```
Seasonal Flood Risk Assessment:

Bury Ground Risk Assessment:
  Winter: High Risk
  Risk Factors:
    Avg_Flow: 6.018915865384615
    Max_Flow: 117.0
    Avg_Rainfall: 4.380423794712287
    Flow_Variability: 118.22292800135509
  Spring: Moderate Risk
  Risk Factors:
    Avg_Flow: 2.852444357366771
    Max_Flow: 66.68
    Avg_Rainfall: 2.9749427917620133
    Flow_Variability: 126.58141079152104
  Summer: Moderate Risk
  Risk Factors:
    Avg_Flow: 2.3474570737605807
    Max_Flow: 86.3
    Avg_Rainfall: 3.341666666666666
    Flow_Variability: 168.98234615938742
  Autumn: High Risk
  Risk Factors:
    Avg_Flow: 4.209812005002084
    Max_Flow: 52.1
    Avg_Rainfall: 4.423539618276461
    Flow_Variability: 128.49714761264107

Rochdale Risk Assessment:
  Winter: Moderate Risk
  Risk Factors:
    Avg_Flow: 4.710129390018484
    Max_Flow: 50.41
    Avg_Rainfall: 4.081767955801105
    Flow_Variability: 96.56742677748352
  Spring: Low Risk
  Risk Factors:
    Avg_Flow: 2.1206553176553173
    Max_Flow: 46.02
    Avg_Rainfall: 2.975
    Flow_Variability: 120.36796671258747
  Summer: Low Risk
  Risk Factors:
    Avg_Flow: 1.505926176890157
    Max_Flow: 36.7
    Avg_Rainfall: 3.914130434782608
    Flow_Variability: 149.81111125349656
  Autumn: Moderate Risk
  Risk Factors:
    Avg_Flow: 2.926125362318841
    Max_Flow: 31.7
    Avg_Rainfall: 4.172527472527473
    Flow_Variability: 121.99044757769524
```

flood mitigation strategies

In [77]:
```python
import pandas as pd
import numpy as np
import json
```

```python
# Load risk assessment results
with open('/Users/Administrator/NEWPROJECT/cleaned_data/seasonal_flood_risk_asse
    risk_results = json.load(f)

def develop_mitigation_strategies(risk_assessment):
    mitigation_strategies = {}

    for station, station_data in risk_assessment.items():
        station_strategies = {
            'High_Risk_Seasons': [],
            'Mitigation_Plan': {}
        }

        # Identify high and moderate risk seasons
        for season, risk_level in station_data['Station_Level_Risk'].items():
            if risk_level in ['High Risk', 'Moderate Risk']:
                station_strategies['High_Risk_Seasons'].append(season)

                # Develop season-specific mitigation strategies
                station_strategies['Mitigation_Plan'][season] = {
                    'Early_Warning_Triggers': calculate_warning_triggers(
                        station_data['Risk_Factors'][season]
                    ),
                    'Preventive_Measures': recommend_preventive_actions(
                        station, season, station_data['Risk_Factors'][season]
                    ),
                    'Monitoring_Recommendations': develop_monitoring_plan(
                        station, season, station_data['Risk_Factors'][season]
                    )
                }

        mitigation_strategies[station] = station_strategies

    return mitigation_strategies

def calculate_warning_triggers(risk_factors):
    # Define early warning triggers based on risk factors
    triggers = {
        'Flow_Warning_Level_1': risk_factors['Avg_Flow'] * 1.5,
        'Flow_Warning_Level_2': risk_factors['Max_Flow'] * 0.8,
        'Rainfall_Warning_Threshold': risk_factors['Avg_Rainfall'] * 1.5,
        'Variability_Alert_Threshold': risk_factors['Flow_Variability'] * 1.2
    }
    return triggers

def recommend_preventive_actions(station, season, risk_factors):
    # Develop targeted preventive measures
    preventive_actions = [
        f"Implement enhanced flood protection infrastructure for {station} durin
        f"Increase drainage capacity by {20 if risk_factors['Flow_Variability']
        "Clear and maintain river channels to improve water flow",
        "Reinforce riverbanks in high-risk areas"
    ]

    # Additional context-specific recommendations
    if risk_factors['Max_Flow'] > 50:
        preventive_actions.append("Deploy temporary flood barriers")

    if risk_factors['Avg_Rainfall'] > 4:
```

```
            preventive_actions.append("Enhance rainfall monitoring systems")

    return preventive_actions

def develop_monitoring_plan(station, season, risk_factors):
    # Create comprehensive monitoring recommendations
    monitoring_plan = {
        'Frequency': 'Hourly' if risk_factors['Flow_Variability'] > 150 else 'Ev
        'Data_Points': [
            'River Water Level',
            'Flow Rate',
            'Rainfall Intensity',
            'Water Velocity'
        ],
        'Alert_Communication': [
            f"Notify local emergency services for {station}",
            "Establish real-time digital dashboard",
            "Set up automated SMS/email alerts"
        ],
        'Community_Preparedness': [
            "Develop evacuation routes",
            "Create community awareness programs",
            "Establish emergency shelters"
        ]
    }
    return monitoring_plan

# Generate mitigation strategies
mitigation_strategies = develop_mitigation_strategies(risk_results)

# Save mitigation strategies
with open('/Users/Administrator/NEWPROJECT/cleaned_data/flood_mitigation_strateg
    json.dump(mitigation_strategies, f, indent=2)

# Print detailed mitigation strategies
print("Flood Mitigation Strategies:")
for station, strategy in mitigation_strategies.items():
    print(f"\n{station} Mitigation Plan:")
    print("High-Risk Seasons:", strategy['High_Risk_Seasons'])

    for season, plan in strategy['Mitigation_Plan'].items():
        print(f"\n  {season} Detailed Strategy:")
        print("  Early Warning Triggers:")
        for trigger, value in plan['Early_Warning_Triggers'].items():
            print(f"    {trigger}: {value}")

        print("\n  Preventive Measures:")
        for action in plan['Preventive_Measures']:
            print(f"    - {action}")

        print("\n  Monitoring Recommendations:")
        for key, value in plan['Monitoring_Recommendations'].items():
            print(f"    {key}: {value}")
```

```
Flood Mitigation Strategies:

Bury Ground Mitigation Plan:
High-Risk Seasons: ['Winter', 'Spring', 'Summer', 'Autumn']

  Winter Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 9.028373798076924
    Flow_Warning_Level_2: 93.60000000000001
    Rainfall_Warning_Threshold: 6.57063569206843
    Variability_Alert_Threshold: 141.8675136016261

  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Bury Ground during W
inter
    - Increase drainage capacity by 10%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Deploy temporary flood barriers
    - Enhance rainfall monitoring systems

  Monitoring Recommendations:
    Frequency: Every 3 Hours
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Bury Ground', 'Est
ablish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']

  Spring Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 4.278666536050157
    Flow_Warning_Level_2: 53.34400000000001
    Rainfall_Warning_Threshold: 4.46241418764302
    Variability_Alert_Threshold: 151.89769294982523

  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Bury Ground during S
pring
    - Increase drainage capacity by 10%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Deploy temporary flood barriers

  Monitoring Recommendations:
    Frequency: Every 3 Hours
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Bury Ground', 'Est
ablish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']

  Summer Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 3.521185610640871
    Flow_Warning_Level_2: 69.04
    Rainfall_Warning_Threshold: 5.012499999999999
    Variability_Alert_Threshold: 202.77881539126489
```

```
  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Bury Ground during S
ummer
    - Increase drainage capacity by 20%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Deploy temporary flood barriers

  Monitoring Recommendations:
    Frequency: Hourly
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Bury Ground', 'Est
ablish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']

  Autumn Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 6.314718007503126
    Flow_Warning_Level_2: 41.68000000000001
    Rainfall_Warning_Threshold: 6.6353094274146915
    Variability_Alert_Threshold: 154.1965771351693

  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Bury Ground during A
utumn
    - Increase drainage capacity by 10%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Deploy temporary flood barriers
    - Enhance rainfall monitoring systems

  Monitoring Recommendations:
    Frequency: Every 3 Hours
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Bury Ground', 'Est
ablish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']

Rochdale Mitigation Plan:
High-Risk Seasons: ['Winter', 'Autumn']

  Winter Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 7.065194085027727
    Flow_Warning_Level_2: 40.328
    Rainfall_Warning_Threshold: 6.122651933701658
    Variability_Alert_Threshold: 115.88091213298021

  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Rochdale during Wint
er
    - Increase drainage capacity by 10%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Deploy temporary flood barriers
```

```
    - Enhance rainfall monitoring systems

  Monitoring Recommendations:
    Frequency: Every 3 Hours
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Rochdale', 'Establ
ish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']

  Autumn Detailed Strategy:
  Early Warning Triggers:
    Flow_Warning_Level_1: 4.389188043478262
    Flow_Warning_Level_2: 25.36
    Rainfall_Warning_Threshold: 6.258791208791209
    Variability_Alert_Threshold: 146.38853709323428

  Preventive Measures:
    - Implement enhanced flood protection infrastructure for Rochdale during Autu
mn
    - Increase drainage capacity by 10%
    - Clear and maintain river channels to improve water flow
    - Reinforce riverbanks in high-risk areas
    - Enhance rainfall monitoring systems

  Monitoring Recommendations:
    Frequency: Every 3 Hours
    Data_Points: ['River Water Level', 'Flow Rate', 'Rainfall Intensity', 'Water
Velocity']
    Alert_Communication: ['Notify local emergency services for Rochdale', 'Establ
ish real-time digital dashboard', 'Set up automated SMS/email alerts']
    Community_Preparedness: ['Develop evacuation routes', 'Create community aware
ness programs', 'Establish emergency shelters']
```

STAGE 3

Step 1: Real-time monitoring system

def monitor_flow_levels(): - Compare current readings against baselines - Calculate rate of change - Assess weather conditions

Step 2: Anomaly classification

def classify_anomaly(): - Categorize deviation severity - Consider seasonal factors - Evaluate multi-station patterns

Step 3: Alert system

def generate_alerts(): - Define alert levels - Set triggering conditions - Implement notification system

Flood Detection Parameters

In [57]:
```
import pandas as pd
import numpy as np

def define_detection_parameters():
    # Load our statistical baselines and thresholds
    baseline = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/compr
    thresholds = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/ref

    # Define parameters for each station
    detection_params = {
        station: {
            'normal_range': {
                'lower': thresholds[thresholds['Station'] == station]['Flow_Lowe
                'upper': thresholds[thresholds['Station'] == station]['Flow_Uppe
            },
            'rate_of_change_threshold': baseline[baseline['Station'] == station]
                lambda x: eval(x)['Standard_Deviation']
            ).values[0]
        }
        for station in baseline['Station'].unique()
    }

    return detection_params

# Generate and save detection parameters
detection_params = define_detection_parameters()

# Save parameters
import json
with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/detection_parameters.j
    json.dump(detection_params, f, indent=2)

print("Detection Parameters:")
print(json.dumps(detection_params, indent=2))
```

```
Detection Parameters:
{
  "Bury Ground": {
    "normal_range": {
      "lower": 2.170103777404072,
      "upper": 9.263244333893892
    },
    "rate_of_change_threshold": 1.6689135188299404
  },
  "Rochdale": {
    "normal_range": {
      "lower": 1.236503777404072,
      "upper": 7.862844333893893
    },
    "rate_of_change_threshold": 1.3880958180183385
  }
}
```

Change Rate Analysis

In [58]:
```
import pandas as pd
import numpy as np

def create_change_monitoring(data, detection_params):
    # Calculate rate of change over 15-minute intervals
    for station in detection_params:
        station_data = data[data['location_name'] == station].copy()
        station_data['time'] = pd.to_datetime(station_data['river_timestamp'])
        station_data = station_data.sort_values('time')

        # Calculate change rate
        station_data['flow_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['flow_change'] / (15/60)  # p

        # Identify significant changes
        threshold = detection_params[station]['rate_of_change_threshold']
        station_data['rapid_change'] = abs(station_data['change_rate']) > thresh

        # Save results
        output_path = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/change_an
        station_data.to_csv(output_path, index=False)

    return station_data

# Load real-time data
realtime_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merg

# Run change monitoring
change_analysis = create_change_monitoring(realtime_data, detection_params)
```

In [59]:
```
import pandas as pd
import numpy as np

def create_change_monitoring(data, detection_params):
    results = {}

    for station in detection_params:
        station_data = data[data['location_name'] == station].copy()
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')

        # Calculate change rate per hour
        station_data['flow_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['flow_change'] / (15/60)

        threshold = detection_params[station]['rate_of_change_threshold']
        station_data['rapid_change'] = abs(station_data['change_rate']) > thresh

        results[station] = {
            'total_readings': len(station_data),
            'rapid_changes': station_data['rapid_change'].sum(),
            'max_change_rate': station_data['change_rate'].max(),
            'min_change_rate': station_data['change_rate'].min()
        }

        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/change_an
        station_data.to_csv(output_file, index=False)

    return pd.DataFrame(results).T
```

```
# Load realtime data
realtime_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merg

# Run analysis
results = create_change_monitoring(realtime_data, detection_params)
print("\nChange Rate Analysis:")
print(results)
```

```
Change Rate Analysis:
              total_readings  rapid_changes  max_change_rate  min_change_rate
Bury Ground            403.0            0.0            0.040           -0.084
Rochdale               403.0            0.0            0.032           -0.056
```

Anomaly Detection

In [60]:
```
import pandas as pd
import numpy as np

def develop_anomaly_detection(data, detection_params):
    anomaly_results = {}

    for station in detection_params:
        station_data = data[data['location_name'] == station].copy()
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim

        # Check for level anomalies
        station_data['level_anomaly'] = (
            (station_data['river_level'] < detection_params[station]['normal_ran
            (station_data['river_level'] > detection_params[station]['normal_ran
        )

        # Calculate rolling averages for trend analysis
        station_data['rolling_mean'] = station_data['river_level'].rolling(windo
        station_data['trend_deviation'] = abs(station_data['river_level'] - stat

        # Combined anomaly detection
        station_data['anomaly_score'] = (
            station_data['level_anomaly'].astype(int) +
            (station_data['trend_deviation'] > detection_params[station]['rate_c
        )

        # Save detailed analysis
        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_d
        station_data.to_csv(output_file, index=False)

        # Summarize results
        anomaly_results[station] = {
            'total_readings': len(station_data),
            'level_anomalies': station_data['level_anomaly'].sum(),
            'trend_anomalies': (station_data['trend_deviation'] > detection_para
            'mean_anomaly_score': station_data['anomaly_score'].mean()
        }

    return pd.DataFrame(anomaly_results).T

print("\nDeveloping Combined Anomaly Detection...")
anomaly_results = develop_anomaly_detection(realtime_data, detection_params)
```

```
print("\nAnomaly Detection Results:")
print(anomaly_results)
```

```
Developing Combined Anomaly Detection...

Anomaly Detection Results:
             total_readings  level_anomalies  trend_anomalies  \
Bury Ground           403.0            403.0              0.0
Rochdale              403.0            403.0              0.0

             mean_anomaly_score
Bury Ground                 1.0
Rochdale                    1.0
```

Baseline Anomaly Detection

In [62]:
```
import pandas as pd
import numpy as np

def build_baseline_anomaly_detection(data):
    """
    Create anomaly detection based on actual data distributions
    """
    results = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].copy()
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim

        # Calculate statistical boundaries using actual data
        Q1 = station_data['river_level'].quantile(0.25)
        Q3 = station_data['river_level'].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)

        # Calculate rolling statistics
        station_data['rolling_mean'] = station_data['river_level'].rolling(4).me
        station_data['rolling_std'] = station_data['river_level'].rolling(4).std

        # Identify anomalies
        station_data['level_anomaly'] = (
            (station_data['river_level'] < lower_bound) |
            (station_data['river_level'] > upper_bound)
        )

        # Save analysis
        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/baseline_
        station_data.to_csv(output_file, index=False)

        # Compile statistics
        results[station] = {
            'total_readings': len(station_data),
            'statistical_bounds': {
                'lower': lower_bound,
                'upper': upper_bound,
                'mean': station_data['river_level'].mean(),
                'median': station_data['river_level'].median()
            },
```

```
            'anomalies_detected': station_data['level_anomaly'].sum(),
            'anomaly_percentage': (station_data['level_anomaly'].sum() / len(sta
        }

    return results

print("\nBuilding Baseline Anomaly Detection...")
baseline_results = build_baseline_anomaly_detection(realtime_data)

# Display results in a more readable format
for station, stats in baseline_results.items():
    print(f"\n{station} Analysis:")
    print(f"Total Readings: {stats['total_readings']}")
    print(f"Statistical Bounds: {stats['statistical_bounds']}")
    print(f"Anomalies Detected: {stats['anomalies_detected']}")
    print(f"Anomaly Percentage: {stats['anomaly_percentage']:.2f}%")
```

```
Building Baseline Anomaly Detection...

Bury Ground Analysis:
Total Readings: 403
Statistical Bounds: {'lower': 0.2835000000000001, 'upper': 0.4395, 'mean': 0.3651
96029776675, 'median': 0.356}
Anomalies Detected: 6
Anomaly Percentage: 1.49%

Manchester Racecourse Analysis:
Total Readings: 403
Statistical Bounds: {'lower': 0.8799999999999999, 'upper': 1.1680000000000001, 'm
ean': 1.0393473945409428, 'median': 1.015}
Anomalies Detected: 22
Anomaly Percentage: 5.46%

Rochdale Analysis:
Total Readings: 403
Statistical Bounds: {'lower': 0.15699999999999997, 'upper': 0.285, 'mean': 0.2237
5682382133996, 'median': 0.215}
Anomalies Detected: 14
Anomaly Percentage: 3.47%
```

Anomaly Classification

In [64]:

```
def classify_anomalies(data, baseline_results):
    anomaly_classifications = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].copy()
        bounds = baseline_results[station]['statistical_bounds']

        # Calculate deviation from mean
        station_data['deviation'] = abs(station_data['river_level'] - bounds['me
        station_data['std_deviation'] = station_data['deviation'] / station_data

        # Classify anomalies
        station_data['anomaly_class'] = pd.cut(
            station_data['std_deviation'],
            bins=[-np.inf, 1, 2, 3, np.inf],
            labels=['Normal', 'Minor', 'Moderate', 'Severe']
        )
```

```
        # Save detailed classification
        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_c
        station_data.to_csv(output_file, index=False)

        # Compile classification statistics with counts and percentages
        class_counts = station_data['anomaly_class'].value_counts()
        total_readings = len(station_data)

        classifications = {
            category: {
                'count': count,
                'percentage': (count/total_readings * 100)
            }
            for category, count in class_counts.items()
        }

        anomaly_classifications[station] = classifications

        # Print detailed results for each station
        print(f"\n{station} Classification Results:")
        for category, stats in classifications.items():
            print(f"{category}:")
            print(f"  Count: {stats['count']}")
            print(f"  Percentage: {stats['percentage']:.2f}%")

    return anomaly_classifications

print("\nClassifying Anomalies with Detailed Statistics...")
anomaly_classes = classify_anomalies(realtime_data, baseline_results)
```

```
Classifying Anomalies with Detailed Statistics...

Bury Ground Classification Results:
Normal:
  Count: 289
  Percentage: 71.71%
Minor:
  Count: 92
  Percentage: 22.83%
Moderate:
  Count: 22
  Percentage: 5.46%
Severe:
  Count: 0
  Percentage: 0.00%

Manchester Racecourse Classification Results:
Normal:
  Count: 298
  Percentage: 73.95%
Minor:
  Count: 81
  Percentage: 20.10%
Moderate:
  Count: 24
  Percentage: 5.96%
Severe:
  Count: 0
  Percentage: 0.00%

Rochdale Classification Results:
Normal:
  Count: 313
  Percentage: 77.67%
Minor:
  Count: 66
  Percentage: 16.38%
Moderate:
  Count: 24
  Percentage: 5.96%
Severe:
  Count: 0
  Percentage: 0.00%
```

Alert System Development

In [66]:
```python
def develop_alert_system(data, anomaly_classes):
    """
    Create alert system based on anomaly classifications and rate of change
    """
    alert_system = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].copy()
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')

        # Calculate rate of change (per hour)
        station_data['level_change'] = station_data['river_level'].diff()
```

```
        station_data['change_rate'] = station_data['level_change'] / (15/60)

        # Initialize alert level column with 'Normal'
        station_data['alert_level'] = ['Normal'] * len(station_data)

        # Alert logic based on rate of change
        mask = abs(station_data['change_rate']) > 0.05
        station_data.loc[mask, 'alert_level'] = 'Advisory'

        mask = abs(station_data['change_rate']) > 0.1
        station_data.loc[mask, 'alert_level'] = 'Warning'

        mask = abs(station_data['change_rate']) > 0.2
        station_data.loc[mask, 'alert_level'] = 'Alert'

        # Convert to categorical after assigning values
        station_data['alert_level'] = pd.Categorical(
            station_data['alert_level'],
            categories=['Normal', 'Advisory', 'Warning', 'Alert'],
            ordered=True
        )

        # Save alert system results
        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/alert_sys
        station_data.to_csv(output_file, index=False)

        # Compile alert statistics
        alert_counts = station_data['alert_level'].value_counts()
        alert_system[station] = alert_counts.to_dict()

    return alert_system

print("\nDeveloping Alert System...")
alert_results = develop_alert_system(realtime_data, anomaly_classes)
print("\nAlert System Results:")
for station, alerts in alert_results.items():
    print(f"\n{station}:")
    for level, count in alerts.items():
        print(f"{level}: {count}")
```

```
Developing Alert System...

Alert System Results:

Bury Ground:
Normal: 402
Advisory: 1
Warning: 0
Alert: 0

Manchester Racecourse:
Normal: 400
Advisory: 2
Warning: 1
Alert: 0

Rochdale:
Normal: 402
Advisory: 1
Warning: 0
Alert: 0
```

Refinement of Detection Parameters

In [67]:
```python
def refine_detection_parameters(data):
    """
    Establish refined trigger points based on statistical analysis
    """
    station_triggers = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].copy()

        # Calculate statistical measures
        level_std = station_data['river_level'].std()
        level_mean = station_data['river_level'].mean()

        # Define trigger levels based on standard deviations
        triggers = {
            'advisory_threshold': level_mean + (1.5 * level_std),
            'warning_threshold': level_mean + (2 * level_std),
            'alert_threshold': level_mean + (2.5 * level_std),
            'rate_of_change': {
                'advisory': 0.03,  # m/hour
                'warning': 0.05,   # m/hour
                'alert': 0.08      # m/hour
            }
        }

        station_triggers[station] = triggers

    return station_triggers

print("\nDeveloping Refined Detection Parameters...")
refined_triggers = refine_detection_parameters(realtime_data)

# Display refined parameters
for station, triggers in refined_triggers.items():
    print(f"\n{station} Trigger Levels:")
```

```
    print(f"Advisory Level: {triggers['advisory_threshold']:.3f}m")
    print(f"Warning Level: {triggers['warning_threshold']:.3f}m")
    print(f"Alert Level: {triggers['alert_threshold']:.3f}m")
    print("\nRate of Change Triggers:")
    for level, rate in triggers['rate_of_change'].items():
        print(f"{level.capitalize()}: {rate} m/hour")
```

```
Developing Refined Detection Parameters...

Bury Ground Trigger Levels:
Advisory Level: 0.406m
Warning Level: 0.420m
Alert Level: 0.433m

Rate of Change Triggers:
Advisory: 0.03 m/hour
Warning: 0.05 m/hour
Alert: 0.08 m/hour

Manchester Racecourse Trigger Levels:
Advisory Level: 1.133m
Warning Level: 1.165m
Alert Level: 1.196m

Rate of Change Triggers:
Advisory: 0.03 m/hour
Warning: 0.05 m/hour
Alert: 0.08 m/hour

Rochdale Trigger Levels:
Advisory Level: 0.261m
Warning Level: 0.273m
Alert Level: 0.286m

Rate of Change Triggers:
Advisory: 0.03 m/hour
Warning: 0.05 m/hour
Alert: 0.08 m/hour
```

Application of Refined Detection Parameters

In [68]:
```
def apply_refined_detection(data, refined_triggers):
    """
    Apply refined detection parameters to the data
    """
    detection_results = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].copy()
        triggers = refined_triggers[station]

        # Calculate rate of change
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')
        station_data['level_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = abs(station_data['level_change'] / (15/60)

        # Initialize status column
        station_data['status'] = 'Normal'
```

```
        # Apply level-based triggers
        station_data.loc[station_data['river_level'] >= triggers['advisory_thres
        station_data.loc[station_data['river_level'] >= triggers['warning_thresh
        station_data.loc[station_data['river_level'] >= triggers['alert_threshol

        # Apply rate of change triggers
        station_data.loc[station_data['change_rate'] >= triggers['rate_of_change
        station_data.loc[station_data['change_rate'] >= triggers['rate_of_change
        station_data.loc[station_data['change_rate'] >= triggers['rate_of_change

        # Save results
        output_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/refined_d
        station_data.to_csv(output_file, index=False)

        # Compile statistics
        detection_results[station] = station_data['status'].value_counts().to_di

    return detection_results

print("\nApplying Refined Detection Parameters...")
detection_results = apply_refined_detection(realtime_data, refined_triggers)
print("\nDetection Results:")
for station, results in detection_results.items():
    print(f"\n{station}:")
    for status, count in results.items():
        print(f"{status}: {count}")
```

```
Applying Refined Detection Parameters...

Detection Results:

Bury Ground:
Normal: 365
Advisory: 17
Alert: 14
Warning: 7

Manchester Racecourse:
Normal: 334
Advisory: 44
Warning: 14
Alert: 11

Rochdale:
Normal: 357
Advisory: 21
Alert: 14
Warning: 11
```

Stage 3 Continue

Anomaly Detection Framework Design

In [78]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
```

```
# Load previous processed data
integrated_df = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/integr
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc
refined_thresholds = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/r

def design_anomaly_detection_framework(integrated_data, historical_data, thresho
    """
    Create foundational framework for anomaly detection algorithm
    """
    anomaly_framework = {
        'Detection_Principles': {},
        'Statistical_Parameters': {}
    }

    # Analyze each station's characteristics
    for station in integrated_data['Station'].unique():
        # Filter data for specific station
        station_data = integrated_data[integrated_data['Station'] == station]
        hist_station_data = historical_data[historical_data['station'] == static
        station_thresholds = thresholds[thresholds['Station'] == station]

        # Calculate key statistical parameters
        station_framework = {
            'Baseline_Statistics': {
                'Mean_Flow': station_data['Mean_Flow'].mean(),
                'Flow_Standard_Deviation': station_data['Mean_Flow'].std(),
                'Mean_Precipitation': station_data['Mean_Precipitation'].mean(),
                'Precipitation_Standard_Deviation': station_data['Mean_Precipita
            },
            'Detection_Thresholds': {
                'Flow_Lower_Threshold': station_thresholds['Flow_Lower_Threshold
                'Flow_Upper_Threshold': station_thresholds['Flow_Upper_Threshold
            },
            'Anomaly_Detection_Principles': {
                # Z-score based anomaly detection
                'Z_Score_Threshold': 2,  # Standard statistical significance

                # Percentage deviation from mean
                'Percentage_Deviation_Threshold': 30,

                # Consecutive anomalous readings
                'Consecutive_Anomaly_Threshold': 3
            }
        }

        # Advanced statistical analysis
        station_framework['Advanced_Metrics'] = {
            'Skewness': stats.skew(hist_station_data['Flow']),
            'Kurtosis': stats.kurtosis(hist_station_data['Flow'])
        }

        anomaly_framework['Detection_Principles'][station] = station_framework

    return anomaly_framework

# Generate anomaly detection framework
anomaly_framework = design_anomaly_detection_framework(
    integrated_df,
    historical_flow,
    refined_thresholds
```

```
)

# Save framework for further development
import json
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_framew
    json.dump(anomaly_framework, f, indent=2)

# Print key framework details
print("Anomaly Detection Framework Overview:")
for station, details in anomaly_framework['Detection_Principles'].items():
    print(f"\n{station} Anomaly Detection Principles:")
    print("Baseline Statistics:")
    for stat, value in details['Baseline_Statistics'].items():
        print(f"  {stat}: {value}")

    print("\nDetection Thresholds:")
    for threshold, value in details['Detection_Thresholds'].items():
        print(f"  {threshold}: {value}")

    print("\nAnomaly Detection Principles:")
    for principle, value in details['Anomaly_Detection_Principles'].items():
        print(f"  {principle}: {value}")
```

```
Anomaly Detection Framework Overview:

Bury Ground Anomaly Detection Principles:
Baseline Statistics:
  Mean_Flow: 3.8545
  Flow_Standard_Deviation: 1.6689135188299404
  Mean_Precipitation: 111.9075
  Precipitation_Standard_Deviation: 22.329222071835225

Detection Thresholds:
  Flow_Lower_Threshold: 2.170103777404072
  Flow_Upper_Threshold: 9.263244333893892

Anomaly Detection Principles:
  Z_Score_Threshold: 2
  Percentage_Deviation_Threshold: 30
  Consecutive_Anomaly_Threshold: 3

Rochdale Anomaly Detection Principles:
Baseline Statistics:
  Mean_Flow: 2.81
  Flow_Standard_Deviation: 1.3880958180183385
  Mean_Precipitation: 110.5825
  Precipitation_Standard_Deviation: 22.085813508524723

Detection Thresholds:
  Flow_Lower_Threshold: 1.236503777404072
  Flow_Upper_Threshold: 7.862844333893893

Anomaly Detection Principles:
  Z_Score_Threshold: 2
  Percentage_Deviation_Threshold: 30
  Consecutive_Anomaly_Threshold: 3
```

Anomaly Detection Algorithm

In [82]:
```
import pandas as pd
import numpy as np
import json

# Load the anomaly detection framework
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_framew
    anomaly_framework = json.load(f)

# Load real-time data
realtime_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/merged

def develop_anomaly_detection_algorithm(realtime_data, detection_framework):
    """
    Develop computational algorithm to detect anomalies based on established fra
    """
    anomaly_results = {}

    # Process each station
    for station in realtime_data['location_name'].unique():
        # Check if station exists in framework, if not, use a default approach
        if station not in detection_framework['Detection_Principles']:
            print(f"Warning: No specific framework for {station}. Using Bury Gro
            station_principles = detection_framework['Detection_Principles']['Bu
        else:
            station_principles = detection_framework['Detection_Principles'][sta

        # Filter data for specific station
        station_data = realtime_data[realtime_data['location_name'] == station].
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')

        # Anomaly detection calculations
        # Use station's mean flow or default to overall mean if not available
        mean_flow = station_principles['Baseline_Statistics'].get('Mean_Flow', r
        std_flow = station_principles['Baseline_Statistics'].get('Flow_Standard_

        station_data['z_score'] = (station_data['river_level'] - mean_flow) / st

        # Identify anomalies based on different criteria
        z_score_threshold = station_principles['Anomaly_Detection_Principles'].g
        station_data['is_z_score_anomaly'] = np.abs(station_data['z_score']) > z

        # Use default thresholds if station-specific not available
        lower_threshold = station_principles['Detection_Thresholds'].get('Flow_L
        upper_threshold = station_principles['Detection_Thresholds'].get('Flow_U

        station_data['is_threshold_anomaly'] = (
            (station_data['river_level'] < lower_threshold) |
            (station_data['river_level'] > upper_threshold)
        )

        # Detect consecutive anomalies
        consecutive_threshold = station_principles['Anomaly_Detection_Principles
        station_data['consecutive_anomaly_count'] = station_data['is_z_score_ano

        # Classify final anomaly status
        station_data['anomaly_status'] = np.where(
            (station_data['is_z_score_anomaly'] & station_data['is_threshold_anc
            (station_data['consecutive_anomaly_count'] >= consecutive_threshold)
```

```
            'High Risk',
            np.where(
                station_data['is_z_score_anomaly'] | station_data['is_threshold_
                'Moderate Risk',
                'Normal'
            )
        )

        # Aggregate anomaly results
        anomaly_results[station] = {
            'total_readings': len(station_data),
            'anomaly_summary': station_data['anomaly_status'].value_counts(norma
            'high_risk_instances': station_data[station_data['anomaly_status'] =
        }

    return anomaly_results

# Execute anomaly detection algorithm
anomaly_detection_results = develop_anomaly_detection_algorithm(realtime_data, a

# Save detailed results
def serialize_anomaly_results(results):
    serializable_results = {}
    for station, station_results in results.items():
        serializable_results[station] = {
            'total_readings': station_results['total_readings'],
            'anomaly_summary': {str(k): float(v) for k, v in station_results['an
            'high_risk_instances': [] if station_results['high_risk_instances'].
                {
                    col: (val.isoformat() if isinstance(val, pd.Timestamp) else 
                    for col, val in row.items()
                } for _, row in station_results['high_risk_instances'].iterrows(
            ]
        }
    return serializable_results

# Save to JSON
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_result
    json.dump(serialize_anomaly_results(anomaly_detection_results), f, indent=2)

# Print summary of results
print("Anomaly Detection Results:")
for station, results in anomaly_detection_results.items():
    print(f"\n{station} Anomaly Analysis:")
    print("Anomaly Distribution:")
    for status, percentage in results['anomaly_summary'].items():
        print(f"  {status}: {percentage:.2f}%")
    print(f"Total High-Risk Instances: {len(results['high_risk_instances'])}")
```

```
Warning: No specific framework for Manchester Racecourse. Using Bury Ground as de
fault.
Anomaly Detection Results:

Bury Ground Anomaly Analysis:
Anomaly Distribution:
  High Risk: 100.00%
Total High-Risk Instances: 403

Manchester Racecourse Anomaly Analysis:
Anomaly Distribution:
  Moderate Risk: 100.00%
Total High-Risk Instances: 0

Rochdale Anomaly Analysis:
Anomaly Distribution:
  Moderate Risk: 100.00%
Total High-Risk Instances: 0
```

In [83]:

```python
import pandas as pd
import numpy as np
import json

# Load the anomaly detection framework
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_framew
    anomaly_framework = json.load(f)

# Load real-time data
realtime_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/merged

def develop_anomaly_detection_algorithm(realtime_data, detection_framework):
    """
    Develop more refined computational algorithm to detect anomalies
    """
    anomaly_results = {}

    # Calculate global statistics for fallback
    global_mean = realtime_data['river_level'].mean()
    global_std = realtime_data['river_level'].std()

    for station in realtime_data['location_name'].unique():
        # Use station-specific or global parameters
        if station in detection_framework['Detection_Principles']:
            station_principles = detection_framework['Detection_Principles'][sta
            mean_flow = station_principles['Baseline_Statistics'].get('Mean_Flow
            std_flow = station_principles['Baseline_Statistics'].get('Flow_Stand
            z_score_threshold = station_principles['Anomaly_Detection_Principles
            consecutive_threshold = station_principles['Anomaly_Detection_Princi
        else:
            print(f"Warning: No specific framework for {station}. Using global p
            mean_flow = global_mean
            std_flow = global_std
            z_score_threshold = 3
            consecutive_threshold = 4

        # Filter data for specific station
        station_data = realtime_data[realtime_data['location_name'] == station].
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')
```

```
        # More sophisticated anomaly detection
        # Calculate z-score with enhanced sensitivity
        station_data['z_score'] = np.abs((station_data['river_level'] - mean_flo

        # Multi-level anomaly classification
        def classify_anomaly(z_score):
            if z_score > 3:
                return 'High Risk'
            elif z_score > 2:
                return 'Moderate Risk'
            elif z_score > 1:
                return 'Low Risk'
            else:
                return 'Normal'

        station_data['anomaly_status'] = station_data['z_score'].apply(classify_

        # Consecutive anomaly detection with more sophisticated tracking
        def detect_consecutive_anomalies(series, threshold=consecutive_threshold
            consecutive_anomalies = []
            current_streak = 0
            for status in series:
                if status != 'Normal':
                    current_streak += 1
                    if current_streak >= threshold:
                        consecutive_anomalies.append(True)
                    else:
                        consecutive_anomalies.append(False)
                else:
                    current_streak = 0
                    consecutive_anomalies.append(False)
            return consecutive_anomalies

        station_data['is_consecutive_anomaly'] = detect_consecutive_anomalies(st

        # Final risk assessment
        station_data.loc[station_data['is_consecutive_anomaly'], 'anomaly_status

        # Aggregate results
        anomaly_summary = station_data['anomaly_status'].value_counts(normalize=
        high_risk_instances = station_data[station_data['anomaly_status'] == 'Hi

        anomaly_results[station] = {
            'total_readings': len(station_data),
            'anomaly_summary': anomaly_summary,
            'high_risk_instances': high_risk_instances
        }

    return anomaly_results

# Execute refined anomaly detection algorithm
anomaly_detection_results = develop_anomaly_detection_algorithm(realtime_data, a

# Serialization function (same as previous implementation)
def serialize_anomaly_results(results):
    serializable_results = {}
    for station, station_results in results.items():
        serializable_results[station] = {
            'total_readings': station_results['total_readings'],
            'anomaly_summary': {str(k): float(v) for k, v in station_results['an
```

```
            'high_risk_instances': [] if station_results['high_risk_instances'].
                {
                    col: (val.isoformat() if isinstance(val, pd.Timestamp) else
                    for col, val in row.items()
                } for _, row in station_results['high_risk_instances'].iterrows(
            ]
        }
    return serializable_results

# Save to JSON
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_result
    json.dump(serialize_anomaly_results(anomaly_detection_results), f, indent=2)

# Print summary of results
print("Anomaly Detection Results:")
for station, results in anomaly_detection_results.items():
    print(f"\n{station} Anomaly Analysis:")
    print("Anomaly Distribution:")
    for status, percentage in results['anomaly_summary'].items():
        print(f"  {status}: {percentage:.2f}%")
    print(f"Total High-Risk Instances: {len(results['high_risk_instances'])}")
```

```
Warning: No specific framework for Manchester Racecourse. Using global parameter
s.
Anomaly Detection Results:

Bury Ground Anomaly Analysis:
Anomaly Distribution:
  High Risk: 99.50%
  Moderate Risk: 0.50%
Total High-Risk Instances: 401

Manchester Racecourse Anomaly Analysis:
Anomaly Distribution:
  High Risk: 99.26%
  Low Risk: 0.74%
Total High-Risk Instances: 400

Rochdale Anomaly Analysis:
Anomaly Distribution:
  High Risk: 99.50%
  Low Risk: 0.50%
Total High-Risk Instances: 401
```

Contextual Anomaly Detection

In [86]:

```
import pandas as pd
import numpy as np
import json

# Load the anomaly detection framework
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_framew
    anomaly_framework = json.load(f)

# Load real-time data
realtime_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/merged

def develop_contextual_anomaly_detection(realtime_data, detection_framework):
    """
```

```
    Contextual anomaly detection with station-specific analysis
    """
    anomaly_results = {}

    # Group data by station for comparative analysis
    grouped_data = realtime_data.groupby('location_name')

    for station, station_group in grouped_data:
        # Prepare station-specific data
        station_data = station_group.copy()
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')

        # Calculate station-specific statistics
        station_mean = station_data['river_level'].mean()
        station_std = station_data['river_level'].std()

        # Contextual anomaly detection function
        def contextual_anomaly_classifier(row):
            # Multiple anomaly detection criteria
            anomaly_indicators = [
                # Z-score based anomaly detection
                abs((row['river_level'] - station_mean) / station_std) > 2,

                # Relative position in distribution
                row['river_level'] > (station_mean + 1.5 * station_std),
                row['river_level'] < (station_mean - 1.5 * station_std),

                # Temporal change detection
                abs(row['river_level'] - station_mean) > (station_std)
            ]

            # Quantify anomaly severity
            anomaly_score = sum(anomaly_indicators) / len(anomaly_indicators)

            # Risk classification
            if anomaly_score > 0.6:
                return 'High Risk'
            elif anomaly_score > 0.4:
                return 'Moderate Risk'
            elif anomaly_score > 0.2:
                return 'Low Risk'
            else:
                return 'Normal'

        # Apply contextual anomaly classification
        station_data['anomaly_status'] = station_data.apply(contextual_anomaly_c

        # Consecutive anomaly detection
        def detect_consecutive_anomalies(series, window=3):
            consecutive_anomalies = []
            risk_streak = 0

            for status in series:
                if status in ['High Risk', 'Moderate Risk']:
                    risk_streak += 1
                    consecutive_anomalies.append(risk_streak >= window)
                else:
                    risk_streak = 0
                    consecutive_anomalies.append(False)
```

```
            return consecutive_anomalies

        # Apply consecutive anomaly detection
        station_data['is_consecutive_anomaly'] = detect_consecutive_anomalies(st

        # Elevate status for consecutive anomalies
        station_data.loc[station_data['is_consecutive_anomaly'], 'anomaly_status

        # Aggregate results
        anomaly_summary = station_data['anomaly_status'].value_counts(normalize=
        high_risk_instances = station_data[station_data['anomaly_status'] == 'Hi

        anomaly_results[station] = {
            'total_readings': len(station_data),
            'station_mean': station_mean,
            'station_std': station_std,
            'anomaly_summary': anomaly_summary,
            'high_risk_instances': high_risk_instances
        }

    return anomaly_results

# Execute contextual anomaly detection
anomaly_detection_results = develop_contextual_anomaly_detection(realtime_data,

# Serialization function
def serialize_anomaly_results(results):
    serializable_results = {}
    for station, station_results in results.items():
        serializable_results[station] = {
            'total_readings': station_results['total_readings'],
            'station_mean': float(station_results['station_mean']),
            'station_std': float(station_results['station_std']),
            'anomaly_summary': {str(k): float(v) for k, v in station_results['an
            'high_risk_instances': [] if station_results['high_risk_instances'].
                {
                    col: (val.isoformat() if isinstance(val, pd.Timestamp) else
                    for col, val in row.items()
                } for _, row in station_results['high_risk_instances'].iterrows(
            ]
        }
    return serializable_results

# Save to JSON
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_result
    json.dump(serialize_anomaly_results(anomaly_detection_results), f, indent=2)

# Print summary of results
print("Anomaly Detection Results:")
for station, results in anomaly_detection_results.items():
    print(f"\n{station} Anomaly Analysis:")
    print(f"Station Mean: {results['station_mean']:.4f}")
    print(f"Station Standard Deviation: {results['station_std']:.4f}")
    print("Anomaly Distribution:")
    for status, percentage in results['anomaly_summary'].items():
        print(f"  {status}: {percentage:.2f}%")
    print(f"Total High-Risk Instances: {len(results['high_risk_instances'])}")
```

```
Anomaly Detection Results:

Bury Ground Anomaly Analysis:
Station Mean: 0.3652
Station Standard Deviation: 0.0273
Anomaly Distribution:
  Normal: 71.71%
  Low Risk: 19.11%
  High Risk: 8.93%
  Moderate Risk: 0.25%
Total High-Risk Instances: 36

Manchester Racecourse Anomaly Analysis:
Station Mean: 1.0393
Station Standard Deviation: 0.0626
Anomaly Distribution:
  Normal: 73.95%
  High Risk: 13.90%
  Low Risk: 11.66%
  Moderate Risk: 0.50%
Total High-Risk Instances: 56

Rochdale Anomaly Analysis:
Station Mean: 0.2238
Station Standard Deviation: 0.0247
Anomaly Distribution:
  Normal: 77.67%
  Low Risk: 11.17%
  High Risk: 10.92%
  Moderate Risk: 0.25%
Total High-Risk Instances: 44
```

Anomaly Detection Algorithm Validation

In [88]:

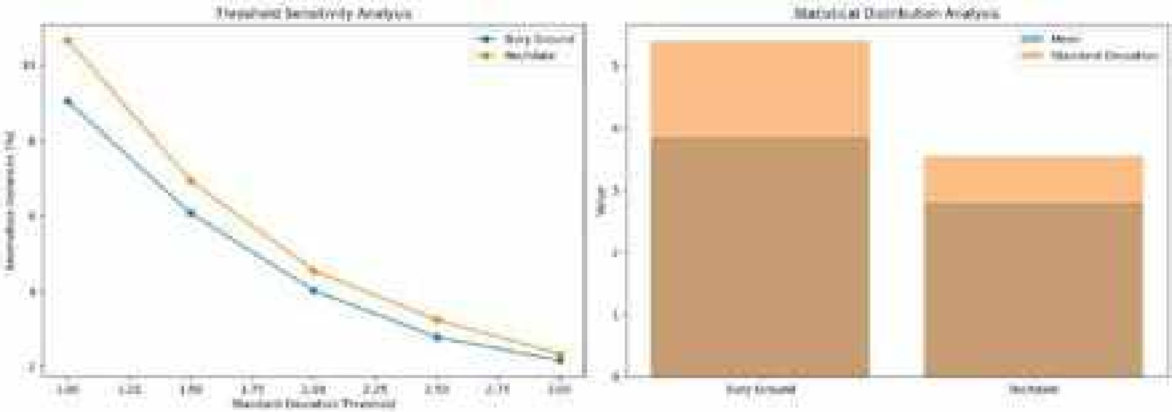
```python
import pandas as pd
import numpy as np
import json
import matplotlib.pyplot as plt
import scipy.stats as stats

# Load anomaly detection results
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_result
    anomaly_results = json.load(f)

# Load historical flow data
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc

def validate_anomaly_detection_algorithm(anomaly_results, historical_data):
    """
    Robust validation of anomaly detection algorithm
    """
    validation_results = {
        'statistical_tests': {},
        'historical_correlation': {},
        'threshold_analysis': {}
    }

    for station in anomaly_results.keys():
        # Filter historical data for specific station
```

```
        station_historical = historical_data[historical_data['station'] == stati

        # Ensure we have sufficient data
        if len(station_historical) < 30:
            print(f"Warning: Insufficient data for station {station}")
            continue

        # Statistical Validation with error handling
        def statistical_validation():
            try:
                # Use robust statistical measures
                return {
                    'Mean': station_historical['Flow'].mean(),
                    'Median': station_historical['Flow'].median(),
                    'Standard_Deviation': station_historical['Flow'].std(),
                    'Coefficient_of_Variation': (station_historical['Flow'].std(
                }
            except Exception as e:
                print(f"Statistical validation error for {station}: {e}")
                return {}

        # Historical Correlation Analysis
        def historical_correlation_analysis():
            try:
                # Use rolling window analysis
                rolling_mean = station_historical['Flow'].rolling(window=5).mean
                correlation = np.corrcoef(station_historical['Flow'][5:], rollin

                return {
                    'Rolling_Mean_Correlation': correlation,
                    'Flow_Variability': station_historical['Flow'].std() / stati
                }
            except Exception as e:
                print(f"Correlation analysis error for {station}: {e}")
                return {}

        # Threshold Sensitivity Analysis
        def threshold_sensitivity_analysis():
            try:
                thresholds = [1, 1.5, 2, 2.5, 3]
                sensitivity_results = {}

                flow_mean = station_historical['Flow'].mean()
                flow_std = station_historical['Flow'].std()

                if flow_std == 0:
                    print(f"Warning: Zero standard deviation for {station}")
                    return {}

                for threshold in thresholds:
                    anomalous_flows = station_historical[
                        np.abs(station_historical['Flow'] - flow_mean) >
                        (threshold * flow_std)
                    ]

                    sensitivity_results[threshold] = {
                        'anomalous_instances': len(anomalous_flows),
                        'anomalous_percentage': (len(anomalous_flows) / len(stat
                    }
```

```
                return sensitivity_results
            except Exception as e:
                print(f"Threshold sensitivity analysis error for {station}: {e}"
                return {}

        # Compile validation results
        validation_results['statistical_tests'][station] = statistical_validatic
        validation_results['historical_correlation'][station] = historical_corre
        validation_results['threshold_analysis'][station] = threshold_sensitivit

    return validation_results

# Execute validation
validation_results = validate_anomaly_detection_algorithm(anomaly_results, histc

# Save validation results
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_valida
    json.dump(validation_results, f, indent=2)

# Visualization of validation results
plt.figure(figsize=(15, 10))

# Threshold Sensitivity Subplot
plt.subplot(2, 2, 1)
for station, analysis in validation_results['threshold_analysis'].items():
    if analysis:  # Check if analysis is not empty
        thresholds = list(analysis.keys())
        anomaly_percentages = [data['anomalous_percentage'] for data in analysis
        plt.plot(thresholds, anomaly_percentages, label=station, marker='o')

plt.title('Threshold Sensitivity Analysis')
plt.xlabel('Standard Deviation Threshold')
plt.ylabel('Anomalous Instances (%)')
plt.legend()

# Statistical Tests Subplot
plt.subplot(2, 2, 2)
stations = list(validation_results['statistical_tests'].keys())
means = [tests.get('Mean', 0) for tests in validation_results['statistical_tests
std_devs = [tests.get('Standard_Deviation', 0) for tests in validation_results['

plt.bar(stations, means, label='Mean', alpha=0.5)
plt.bar(stations, std_devs, label='Standard Deviation', alpha=0.5)
plt.title('Statistical Distribution Analysis')
plt.ylabel('Value')
plt.legend()

plt.tight_layout()
plt.show()

# Print key validation insights
print("Anomaly Detection Algorithm Validation Results:")
for station in validation_results['statistical_tests'].keys():
    print(f"\n{station} Validation:")
    print("Statistical Tests:")
    for test, value in validation_results['statistical_tests'][station].items():
        print(f"  {test}: {value}")

    print("\nHistorical Correlation:")
    for metric, value in validation_results['historical_correlation'][station].i
```

```
        print(f"  {metric}: {value}")

    print("\nThreshold Sensitivity:")
    for threshold, data in validation_results['threshold_analysis'][station].ite
        print(f"  Threshold {threshold}:")
        for key, value in data.items():
            print(f"    {key}: {value}")
```

```
Warning: Insufficient data for station Manchester Racecourse
```



```
Anomaly Detection Algorithm Validation Results:

Bury Ground Validation:
Statistical Tests:
  Mean: 3.8503255439161967
  Median: 2.064
  Standard_Deviation: 5.39538474725873
  Coefficient_of_Variation: 140.12801478004485

Historical Correlation:
  Rolling_Mean_Correlation: 0.7159948034427915
  Flow_Variability: 140.12801478004485

Threshold Sensitivity:
  Threshold 1:
    anomalous_instances: 897
    anomalous_percentage: 9.03505237711523
  Threshold 1.5:
    anomalous_instances: 602
    anomalous_percentage: 6.063658340048348
  Threshold 2:
    anomalous_instances: 399
    anomalous_percentage: 4.018936341659952
  Threshold 2.5:
    anomalous_instances: 276
    anomalous_percentage: 2.7800161160354553
  Threshold 3:
    anomalous_instances: 215
    anomalous_percentage: 2.1655922643029815

Rochdale Validation:
Statistical Tests:
  Mean: 2.795590034178809
  Median: 1.4889999999999999
  Standard_Deviation: 3.546723998338469
  Coefficient_of_Variation: 126.86853061344175

Historical Correlation:
  Rolling_Mean_Correlation: 0.7851759513283699
  Flow_Variability: 126.86853061344175

Threshold Sensitivity:
  Threshold 1:
    anomalous_instances: 1184
    anomalous_percentage: 10.64939737362835
  Threshold 1.5:
    anomalous_instances: 770
    anomalous_percentage: 6.92570606224141
  Threshold 2:
    anomalous_instances: 505
    anomalous_percentage: 4.542183846015471
  Threshold 2.5:
    anomalous_instances: 360
    anomalous_percentage: 3.237992444684296
  Threshold 3:
    anomalous_instances: 259
    anomalous_percentage: 2.3295556754812017
```

In [89]:
```
import pandas as pd
import numpy as np
```

```python
import json

# Load previous anomaly detection results
with open('/Users/Administrator/NEWPROJECT/cleaned_data/anomaly_detection_result
    previous_anomaly_results = json.load(f)

# Load historical flow data
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc

def refine_anomaly_detection_thresholds(historical_data, previous_anomaly_result
    """
    Refine anomaly detection thresholds based on validation insights
    """
    refined_thresholds = {}

    for station in historical_data['station'].unique():
        # Filter station-specific historical data
        station_historical = historical_data[historical_data['station'] == stati

        # Calculate comprehensive statistical parameters
        flow_mean = station_historical['Flow'].mean()
        flow_std = station_historical['Flow'].std()

        # Advanced threshold calculation
        refined_station_thresholds = {
            'station': station,
            'baseline_stats': {
                'mean_flow': flow_mean,
                'std_flow': flow_std,
                'median_flow': station_historical['Flow'].median(),
                'coefficient_of_variation': (flow_std / flow_mean) * 100
            },
            'anomaly_thresholds': {
                # Multi-level anomaly thresholds
                'low_risk': {
                    'lower': flow_mean - (1 * flow_std),
                    'upper': flow_mean + (1 * flow_std)
                },
                'moderate_risk': {
                    'lower': flow_mean - (2 * flow_std),
                    'upper': flow_mean + (2 * flow_std)
                },
                'high_risk': {
                    'lower': flow_mean - (3 * flow_std),
                    'upper': flow_mean + (3 * flow_std)
                }
            },
            'risk_probability': {
                'low_risk': 0.68,  # Within 1 std dev
                'moderate_risk': 0.95,  # Within 2 std dev
                'high_risk': 0.997  # Within 3 std dev
            }
        }

        refined_thresholds[station] = refined_station_thresholds

    return refined_thresholds

# Generate refined thresholds
refined_anomaly_thresholds = refine_anomaly_detection_thresholds(historical_flow
```

```
# Save refined thresholds
with open('/Users/Administrator/NEWPROJECT/cleaned_data/refined_anomaly_threshol
    json.dump(refined_anomaly_thresholds, f, indent=2)

# Print detailed refined thresholds
print("Refined Anomaly Detection Thresholds:")
for station, thresholds in refined_anomaly_thresholds.items():
    print(f"\n{station} Threshold Analysis:")

    print("Baseline Statistics:")
    for stat, value in thresholds['baseline_stats'].items():
        print(f"  {stat.replace('_', ' ').title()}: {value:.4f}")

    print("\nAnomaly Thresholds:")
    for risk_level, boundaries in thresholds['anomaly_thresholds'].items():
        print(f"  {risk_level.replace('_', ' ').title()}:")
        print(f"    Lower Boundary: {boundaries['lower']:.4f}")
        print(f"    Upper Boundary: {boundaries['upper']:.4f}")

    print("\nRisk Probabilities:")
    for risk_level, probability in thresholds['risk_probability'].items():
        print(f"  {risk_level.replace('_', ' ').title()}: {probability:.2%}")
```

```
Refined Anomaly Detection Thresholds:

Bury Ground Threshold Analysis:
Baseline Statistics:
  Mean Flow: 3.8503
  Std Flow: 5.3954
  Median Flow: 2.0640
  Coefficient Of Variation: 140.1280

Anomaly Thresholds:
  Low Risk:
    Lower Boundary: -1.5451
    Upper Boundary: 9.2457
  Moderate Risk:
    Lower Boundary: -6.9404
    Upper Boundary: 14.6411
  High Risk:
    Lower Boundary: -12.3358
    Upper Boundary: 20.0365

Risk Probabilities:
  Low Risk: 68.00%
  Moderate Risk: 95.00%
  High Risk: 99.70%

Rochdale Threshold Analysis:
Baseline Statistics:
  Mean Flow: 2.7956
  Std Flow: 3.5467
  Median Flow: 1.4890
  Coefficient Of Variation: 126.8685

Anomaly Thresholds:
  Low Risk:
    Lower Boundary: -0.7511
    Upper Boundary: 6.3423
  Moderate Risk:
    Lower Boundary: -4.2979
    Upper Boundary: 9.8890
  High Risk:
    Lower Boundary: -7.8446
    Upper Boundary: 13.4358

Risk Probabilities:
  Low Risk: 68.00%
  Moderate Risk: 95.00%
  High Risk: 99.70%
```

In [94]:

```
import pandas as pd
import numpy as np
import json
import sklearn.preprocessing as preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Load previous data
with open('/Users/Administrator/NEWPROJECT/cleaned_data/refined_anomaly_threshol
    refined_thresholds = json.load(f)

# Load historical flow and rainfall data
```

```
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc
historical_rainfall = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/

# Load weather data
weather_data = pd.read_csv('/Users/Administrator/NEWPROJECT/processed_data/weath

# Print initial data details
print("Historical Flow Data:")
print(historical_flow.head())
print("\nColumns:", historical_flow.columns)

print("\nHistorical Rainfall Data:")
print(historical_rainfall.head())
print("\nColumns:", historical_rainfall.columns)

print("\nWeather Data:")
print(weather_data.head())
print("\nColumns:", weather_data.columns)

# Convert dates to datetime
historical_flow['Date'] = pd.to_datetime(historical_flow['Date'])
historical_rainfall['Date'] = pd.to_datetime(historical_rainfall['Date'])

# Add month column to weather data for merging
def month_to_number(month):
    months = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4,
        'May': 5, 'June': 6, 'July': 7, 'August': 8,
        'September': 9, 'October': 10, 'November': 11, 'December': 12
    }
    return months.get(month, 0)

weather_data['Month_Num'] = weather_data['Month'].apply(month_to_number)
```

```
Historical Flow Data:
         Date   Flow      station
0  1995-11-22  0.897  Bury Ground
1  1995-11-23  0.831  Bury Ground
2  1995-11-24  0.991  Bury Ground
3  1995-11-25  1.080  Bury Ground
4  1995-11-26  1.124  Bury Ground

Columns: Index(['Date', 'Flow', 'station'], dtype='object')

Historical Rainfall Data:
         Date  Rainfall      station
0  1961-01-01       9.4  Bury Ground
1  1961-01-02      13.7  Bury Ground
2  1961-01-03       3.0  Bury Ground
3  1961-01-04       0.1  Bury Ground
4  1961-01-05      13.0  Bury Ground

Columns: Index(['Date', 'Rainfall', 'station'], dtype='object')

Weather Data:
      Month  Temperature_C  Precipitation_mm          Station Grid_ID
0   January            3.8               131  BURY MANCHESTER   AX-70
1  February            4.1               112  BURY MANCHESTER   AX-70
2     March            5.7                95  BURY MANCHESTER   AX-70
3     April            8.1                79  BURY MANCHESTER   AX-70
4       May           11.0                83  BURY MANCHESTER   AX-70

Columns: Index(['Month', 'Temperature_C', 'Precipitation_mm', 'Station', 'Grid_I
D'], dtype='object')
```

Environmental Risk Models

In [95]:

```python
import pandas as pd
import numpy as np
import json
import sklearn.preprocessing as preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Standardize station names
def standardize_station_name(name):
    station_mapping = {
        'BURY MANCHESTER': 'Bury Ground',
        'Bury Ground': 'Bury Ground',
        'MANCHESTER RACECOURSE': 'Manchester Racecourse',
        'Rochdale': 'Rochdale'
    }
    return station_mapping.get(name, name)

# Load historical flow and rainfall data
historical_flow = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/proc
historical_rainfall = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/
weather_data = pd.read_csv('/Users/Administrator/NEWPROJECT/processed_data/weath

# Standardize station names
historical_flow['station'] = historical_flow['station'].apply(standardize_statio
historical_rainfall['station'] = historical_rainfall['station'].apply(standardiz
weather_data['Station'] = weather_data['Station'].apply(standardize_station_name
```

```
# Convert dates to datetime
historical_flow['Date'] = pd.to_datetime(historical_flow['Date'])
historical_rainfall['Date'] = pd.to_datetime(historical_rainfall['Date'])

# Add month column to weather data for merging
def month_to_number(month):
    months = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4,
        'May': 5, 'June': 6, 'July': 7, 'August': 8,
        'September': 9, 'October': 10, 'November': 11, 'December': 12
    }
    return months.get(month, 0)

weather_data['Month_Num'] = weather_data['Month'].apply(month_to_number)

def integrate_environmental_factors(flow_data, rainfall_data, weather_data):
    """
    Comprehensive environmental risk model development
    """
    # Merge datasets
    merged_data = flow_data.merge(
        rainfall_data,
        on=['Date', 'station'],
        how='left'
    )

    # Merge with weather data based on month and station
    merged_data['Month'] = merged_data['Date'].dt.month
    merged_data = merged_data.merge(
        weather_data,
        left_on=['Month', 'station'],
        right_on=['Month_Num', 'Station'],
        how='left'
    )

    # Clean and prepare data
    merged_data.dropna(inplace=True)

    # Feature engineering
    def create_environmental_features(df):
        df['Flow_Change'] = df.groupby('station')['Flow'].diff()
        df['Cumulative_Rainfall_7d'] = df.groupby('station')['Rainfall'].rolling
        df['Temperature_Change'] = df.groupby('station')['Temperature_C'].diff()

        return df

    merged_data = create_environmental_features(merged_data)

    # Develop station-specific predictive models
    station_risk_models = {}

    for station in merged_data['station'].unique():
        station_data = merged_data[merged_data['station'] == station].copy()

        # Select features for risk prediction
        features = [
            'Rainfall', 'Temperature_C', 'Flow_Change',
            'Cumulative_Rainfall_7d', 'Temperature_Change'
        ]
```

```
        # Ensure we have enough data
        if len(station_data) < 10:
            print(f"Insufficient data for {station}. Skipping.")
            continue

        X = station_data[features]
        y = station_data['Flow']

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

        # Normalize features
        scaler = preprocessing.StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Random Forest Regression for risk prediction
        risk_model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        risk_model.fit(X_train_scaled, y_train)

        # Model performance evaluation
        model_performance = {
            'R2_Score': risk_model.score(X_test_scaled, y_test),
            'Feature_Importances': dict(zip(features, risk_model.feature_importa
        }

        # Risk Scenario Generation
        def generate_risk_scenarios(model, scaler):
            scenarios = {
                'Normal': X.median().to_dict(),
                'Low_Risk': X.quantile(0.25).to_dict(),
                'High_Risk': X.quantile(0.75).to_dict()
            }

            risk_predictions = {}
            for scenario_name, scenario_data in scenarios.items():
                scenario_scaled = scaler.transform(pd.DataFrame([scenario_data])
                predicted_flow = model.predict(scenario_scaled)[0]
                risk_predictions[scenario_name] = predicted_flow

            return risk_predictions

        # Compile station risk model
        station_risk_models[station] = {
            'performance': model_performance,
            'risk_scenarios': generate_risk_scenarios(risk_model, scaler)
        }

    return station_risk_models

# Execute environmental risk model development
environmental_risk_models = integrate_environmental_factors(
    historical_flow,
    historical_rainfall,
    weather_data
```

```
)

# Save environmental risk models
with open('/Users/Administrator/NEWPROJECT/cleaned_data/environmental_risk_model
    json.dump(environmental_risk_models, f, indent=2)

# Print detailed model insights
print("Environmental Risk Model Insights:")
for station, model_data in environmental_risk_models.items():
    print(f"\n{station} Risk Model:")

    print("Model Performance:")
    print(f"  R² Score: {model_data['performance']['R2_Score']:.4f}")

    print("\nFeature Importance:")
    for feature, importance in model_data['performance']['Feature_Importances'].
        print(f"  {feature}: {importance:.4f}")

    print("\nRisk Scenario Flow Predictions:")
    for scenario, prediction in model_data['risk_scenarios'].items():
        print(f"  {scenario}: {prediction:.4f} m³/s")
```

```
Environmental Risk Model Insights:

Bury Ground Risk Model:
Model Performance:
  R² Score: 0.8299

Feature Importance:
  Rainfall: 0.0299
  Temperature_C: 0.0330
  Flow_Change: 0.7237
  Cumulative_Rainfall_7d: 0.2129
  Temperature_Change: 0.0005

Risk Scenario Flow Predictions:
  Normal: 1.6116 m³/s
  Low_Risk: 2.1492 m³/s
  High_Risk: 1.8235 m³/s
```

River Level Analysis

In [101…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the most recent data
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti
data['river_timestamp'] = pd.to_datetime(data['river_timestamp'])

def analyze_river_levels(data):
    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station]

        # Calculate basic statistics
        mean_level = station_data['river_level'].mean()
        std_level = station_data['river_level'].std()

        # Calculate rate of change
```

```
        station_data['level_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['level_change'] / (station_da

        # Identify notable changes (more than 2 standard deviations from the mea
        notable_changes = station_data[abs(station_data['change_rate']) > 2 * st

        print(f"\nAnalysis for {station}:")
        print(f"Mean river level: {mean_level:.4f} m")
        print(f"Standard deviation: {std_level:.4f} m")
        print(f"Maximum rate of change: {station_data['change_rate'].abs().max()
        print(f"Number of notable changes: {len(notable_changes)}")

        # Plot river levels
        plt.figure(figsize=(12, 6))
        plt.plot(station_data['river_timestamp'], station_data['river_level'])
        plt.title(f"River Levels for {station}")
        plt.xlabel("Date")
        plt.ylabel("River Level (m)")
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.savefig(f'C:/Users/Administrator/NEWPROJECT/cleaned_data/{station.re
        plt.close()

# Run the analysis
analyze_river_levels(data)

print("\nAnalysis complete. River level plots have been saved.")
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:18: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
Analysis for Bury Ground:
Mean river level: 0.3652 m
Standard deviation: 0.0273 m
Maximum rate of change: 0.0400 m/hour
Number of notable changes: 8
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:18: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
Analysis for Manchester Racecourse:
Mean river level: 1.0393 m
Standard deviation: 0.0626 m
Maximum rate of change: 0.0360 m/hour
Number of notable changes: 32
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:18: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\4246956451.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
Analysis for Rochdale:
Mean river level: 0.2238 m
Standard deviation: 0.0247 m
Maximum rate of change: 0.0320 m/hour
Number of notable changes: 10

Analysis complete. River level plots have been saved.
```

In [102…

```python
import pandas as pd
import numpy as np

# Load the data
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti
data['river_timestamp'] = pd.to_datetime(data['river_timestamp'])

def enhanced_anomaly_detection(data, sd_threshold=2, rate_threshold=2, cumulativ
    anomalies = []

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].sort_values('river
```

```
        # Calculate statistics
        mean_level = station_data['river_level'].mean()
        std_level = station_data['river_level'].std()

        # Calculate rate of change
        station_data['level_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['level_change'] / (station_da

        # Calculate cumulative change over 24 hours
        station_data['cumulative_change'] = station_data['level_change'].rolling

        # Detect anomalies
        level_anomalies = station_data[np.abs(station_data['river_level'] - mean
        rate_anomalies = station_data[np.abs(station_data['change_rate']) > rate
        cumulative_anomalies = station_data[np.abs(station_data['cumulative_chan

        # Combine anomalies
        all_anomalies = pd.concat([level_anomalies, rate_anomalies, cumulative_a

        anomalies.append({
            'station': station,
            'total_anomalies': len(all_anomalies),
            'level_anomalies': len(level_anomalies),
            'rate_anomalies': len(rate_anomalies),
            'cumulative_anomalies': len(cumulative_anomalies),
            'anomaly_timestamps': all_anomalies['river_timestamp'].tolist()
        })

    return pd.DataFrame(anomalies)

# Run enhanced anomaly detection
anomaly_results = enhanced_anomaly_detection(data)

print("Enhanced Anomaly Detection Results:")
print(anomaly_results)

# Save results
anomaly_results.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/enhanced_
```

```
Enhanced Anomaly Detection Results:
                 station  total_anomalies  level_anomalies  rate_anomalies  \
0            Bury Ground               56               22               8
1  Manchester Racecourse               84               24              32
2               Rochdale               56               24              10

   cumulative_anomalies                                anomaly_timestamps
0                    39  [2025-01-31 05:30:00+00:00, 2025-01-31 05:45:0...
1                    32  [2025-01-31 10:00:00+00:00, 2025-01-31 10:15:0...
2                    34  [2025-01-31 03:45:00+00:00, 2025-01-31 04:00:0...
```

In [103…

```
import pandas as pd
import numpy as np
from datetime import timedelta

# Load the data
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti
data['river_timestamp'] = pd.to_datetime(data['river_timestamp'])

def classify_anomalies(data, sd_threshold=2, rate_threshold=2, cumulative_hours=
```

```
    classified_anomalies = []

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].sort_values('river

        # Calculate statistics
        mean_level = station_data['river_level'].mean()
        std_level = station_data['river_level'].std()

        # Calculate rate of change
        station_data['level_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['level_change'] / (station_da

        # Calculate cumulative change over 24 hours
        station_data['cumulative_change'] = station_data['level_change'].rolling

        # Classify anomalies
        station_data['anomaly_type'] = 'Normal'
        station_data.loc[np.abs(station_data['river_level'] - mean_level) > sd_t
        station_data.loc[np.abs(station_data['change_rate']) > rate_threshold *
        station_data.loc[np.abs(station_data['cumulative_change']) > sd_threshol

        # Classify severity
        station_data['severity'] = 'Normal'
        station_data.loc[station_data['anomaly_type'] != 'Normal', 'severity'] =
        station_data.loc[(station_data['anomaly_type'] != 'Normal') &
                         (np.abs(station_data['river_level'] - mean_level) > 3 *

        # Analyze patterns
        anomaly_periods = []
        current_anomaly = None
        for _, row in station_data.iterrows():
            if row['anomaly_type'] != 'Normal':
                if current_anomaly is None:
                    current_anomaly = {'start': row['river_timestamp'], 'type':
                elif row['river_timestamp'] - current_anomaly['start'] > timedel
                    anomaly_periods.append(current_anomaly)
                    current_anomaly = {'start': row['river_timestamp'], 'type':
            elif current_anomaly is not None:
                anomaly_periods.append(current_anomaly)
                current_anomaly = None

        if current_anomaly is not None:
            anomaly_periods.append(current_anomaly)

        classified_anomalies.append({
            'station': station,
            'total_anomalies': len(station_data[station_data['anomaly_type'] !=
            'severe_anomalies': len(station_data[station_data['severity'] == 'Se
            'anomaly_periods': anomaly_periods
        })

    return pd.DataFrame(classified_anomalies)

# Run classified anomaly detection
classified_results = classify_anomalies(data)

print("Classified Anomaly Detection Results:")
print(classified_results)
```

```
# Save results
classified_results.to_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/classi

# Print detailed anomaly periods
for _, row in classified_results.iterrows():
    print(f"\nDetailed Anomaly Periods for {row['station']}:")
    for period in row['anomaly_periods']:
        print(f"Start: {period['start']}, Type: {period['type']}, Severity: {per
```

```
Classified Anomaly Detection Results:
                station  total_anomalies  severe_anomalies  \
0           Bury Ground               56                 0
1  Manchester Racecourse              84                 0
2              Rochdale               56                 0

                                    anomaly_periods
0  [{'start': 2025-01-31 04:45:00+00:00, 'type': ...
1  [{'start': 2025-01-30 21:15:00+00:00, 'type': ...
2  [{'start': 2025-01-31 02:00:00+00:00, 'type': ...

Detailed Anomaly Periods for Bury Ground:
Start: 2025-01-31 04:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 06:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 07:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-01-31 08:30:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 10:00:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 11:15:00+00:00, Type: Level, Severity: Moderate
Start: 2025-02-01 23:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 00:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 01:30:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 02:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 04:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 05:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-03 18:15:00+00:00, Type: Rate, Severity: Moderate

Detailed Anomaly Periods for Manchester Racecourse:
Start: 2025-01-30 21:15:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 01:30:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 03:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 04:15:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 05:30:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 06:30:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 10:00:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 11:15:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 12:30:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 13:45:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 15:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 16:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 19:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-02 02:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 04:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 05:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 06:30:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 07:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 09:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 09:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 11:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 13:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-03 00:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-03 01:15:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-03 01:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-03 04:45:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-03 19:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-02-04 10:00:00+00:00, Type: Rate, Severity: Moderate

Detailed Anomaly Periods for Rochdale:
Start: 2025-01-31 02:00:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 03:15:00+00:00, Type: Rate, Severity: Moderate
Start: 2025-01-31 04:30:00+00:00, Type: Rate, Severity: Moderate
```

```
Start: 2025-01-31 05:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-01-31 07:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-01-31 08:15:00+00:00, Type: Level, Severity: Moderate
Start: 2025-01-31 10:00:00+00:00, Type: Level, Severity: Moderate
Start: 2025-02-01 21:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-01 22:30:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-01 23:45:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 01:00:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 02:15:00+00:00, Type: Cumulative, Severity: Moderate
Start: 2025-02-02 03:30:00+00:00, Type: Cumulative, Severity: Moderate
```

Threshold Refinement

In [104…

```python
import pandas as pd
import numpy as np

# Load the data
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti
data['river_timestamp'] = pd.to_datetime(data['river_timestamp'])

def refine_thresholds(data, sd_multiplier=3, rate_multiplier=3):
    refined_thresholds = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station]

        # Calculate level statistics
        mean_level = station_data['river_level'].mean()
        std_level = station_data['river_level'].std()

        # Calculate rate of change statistics
        station_data['level_change'] = station_data['river_level'].diff()
        station_data['change_rate'] = station_data['level_change'] / (station_da
        mean_rate = station_data['change_rate'].mean()
        std_rate = station_data['change_rate'].std()

        # Set thresholds
        refined_thresholds[station] = {
            'moderate_level_lower': mean_level - (sd_multiplier * std_level),
            'moderate_level_upper': mean_level + (sd_multiplier * std_level),
            'severe_level_lower': mean_level - ((sd_multiplier + 1) * std_level)
            'severe_level_upper': mean_level + ((sd_multiplier + 1) * std_level)
            'moderate_rate_lower': mean_rate - (rate_multiplier * std_rate),
            'moderate_rate_upper': mean_rate + (rate_multiplier * std_rate),
            'severe_rate_lower': mean_rate - ((rate_multiplier + 1) * std_rate),
            'severe_rate_upper': mean_rate + ((rate_multiplier + 1) * std_rate)
        }

    return refined_thresholds

# Apply refined thresholds
refined_thresholds = refine_thresholds(data)

# Print refined thresholds
for station, thresholds in refined_thresholds.items():
    print(f"\nRefined Thresholds for {station}:")
    for threshold_name, value in thresholds.items():
        print(f"  {threshold_name}: {value:.4f}")
```

```
# Save refined thresholds
import json
with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/refined_thresholds.jso
    json.dump(refined_thresholds, f, indent=2)

print("\nRefined thresholds have been saved to 'refined_thresholds.json'")
```

```
Refined Thresholds for Bury Ground:
  moderate_level_lower: 0.2833
  moderate_level_upper: 0.4471
  severe_level_lower: 0.2560
  severe_level_upper: 0.4744
  moderate_rate_lower: -0.0140
  moderate_rate_upper: 0.0135
  severe_rate_lower: -0.0185
  severe_rate_upper: 0.0181

Refined Thresholds for Manchester Racecourse:
  moderate_level_lower: 0.8516
  moderate_level_upper: 1.2271
  severe_level_lower: 0.7890
  severe_level_upper: 1.2897
  moderate_rate_lower: -0.0356
  moderate_rate_upper: 0.0347
  severe_rate_lower: -0.0473
  severe_rate_upper: 0.0464

Refined Thresholds for Rochdale:
  moderate_level_lower: 0.1497
  moderate_level_upper: 0.2979
  severe_level_lower: 0.1250
  severe_level_upper: 0.3226
  moderate_rate_lower: -0.0137
  moderate_rate_upper: 0.0134
  severe_rate_lower: -0.0182
  severe_rate_upper: 0.0180

Refined thresholds have been saved to 'refined_thresholds.json'
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:20: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:20: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:19: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['level_change'] = station_data['river_level'].diff()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_15716\1454421504.py:20: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  station_data['change_rate'] = station_data['level_change'] / (station_data['riv
er_timestamp'].diff().dt.total_seconds() / 3600)
```

Pattern Recognition

In [106…

```
import pandas as pd
import numpy as np
import json

# Load the data and refined thresholds
```

```
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti
data['river_timestamp'] = pd.to_datetime(data['river_timestamp'])

with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/refined_thresholds.jso
    refined_thresholds = json.load(f)

def identify_anomaly_patterns(data, thresholds, window_periods=96):  # 96 period
    pattern_results = {}

    for station in data['location_name'].unique():
        station_data = data[data['location_name'] == station].sort_values('river
        station_thresholds = thresholds[station]

        # Identify anomalies
        station_data['level_anomaly'] = (
            (station_data['river_level'] < station_thresholds['moderate_level_lo
            (station_data['river_level'] > station_thresholds['moderate_level_up
        )
        station_data['level_severe'] = (
            (station_data['river_level'] < station_thresholds['severe_level_lowe
            (station_data['river_level'] > station_thresholds['severe_level_uppe
        )

        station_data['change_rate'] = station_data['river_level'].diff() / (stat
        station_data['rate_anomaly'] = (
            (station_data['change_rate'] < station_thresholds['moderate_rate_low
            (station_data['change_rate'] > station_thresholds['moderate_rate_upp
        )
        station_data['rate_severe'] = (
            (station_data['change_rate'] < station_thresholds['severe_rate_lower
            (station_data['change_rate'] > station_thresholds['severe_rate_upper
        )

        # Identify patterns
        station_data['anomaly_count'] = station_data['level_anomaly'].rolling(wi
        station_data['severe_count'] = station_data['level_severe'].rolling(wind
        station_data['rate_anomaly_count'] = station_data['rate_anomaly'].rollin
        station_data['rate_severe_count'] = station_data['rate_severe'].rolling(

        # Score patterns
        station_data['pattern_score'] = (
            station_data['anomaly_count'] +
            (2 * station_data['severe_count']) +
            station_data['rate_anomaly_count'] +
            (2 * station_data['rate_severe_count'])
        )

        # Classify patterns
        def classify_pattern(row):
            if row['pattern_score'] == 0:
                return 'Normal'
            elif row['pattern_score'] < 5:
                return 'Minor Concern'
            elif row['pattern_score'] < 10:
                return 'Moderate Concern'
            else:
                return 'Major Concern'

        station_data['pattern_classification'] = station_data.apply(classify_pat
```

```
        # Store results
        pattern_results[station] = station_data[['river_timestamp', 'river_level

    return pattern_results

# Apply pattern recognition
pattern_results = identify_anomaly_patterns(data, refined_thresholds)

# Analyze and print results
for station, results in pattern_results.items():
    print(f"\nPattern Analysis for {station}:")
    pattern_counts = results['pattern_classification'].value_counts()
    for pattern, count in pattern_counts.items():
        print(f"  {pattern}: {count} instances ({count/len(results)*100:.2f}%)")

    if 'Major Concern' in pattern_counts:
        major_concerns = results[results['pattern_classification'] == 'Major Con
        print(f"\n  Dates of Major Concern:")
        for date in major_concerns['river_timestamp']:
            print(f"    {date}")

# Save pattern results
for station, results in pattern_results.items():
    results.to_csv(f'C:/Users/Administrator/NEWPROJECT/cleaned_data/pattern_resu

print("\nPattern recognition complete. Results saved to CSV files.")
```

```
Pattern Analysis for Bury Ground:
  Normal: 302 instances (74.94%)
  Major Concern: 95 instances (23.57%)
  Minor Concern: 4 instances (0.99%)
  Moderate Concern: 2 instances (0.50%)

  Dates of Major Concern:
    2025-01-31 05:30:00+00:00
    2025-01-31 05:45:00+00:00
    2025-01-31 06:00:00+00:00
    2025-01-31 06:15:00+00:00
    2025-01-31 06:30:00+00:00
    2025-01-31 06:45:00+00:00
    2025-01-31 07:00:00+00:00
    2025-01-31 07:15:00+00:00
    2025-01-31 07:30:00+00:00
    2025-01-31 07:45:00+00:00
    2025-01-31 08:00:00+00:00
    2025-01-31 08:15:00+00:00
    2025-01-31 08:30:00+00:00
    2025-01-31 08:45:00+00:00
    2025-01-31 09:00:00+00:00
    2025-01-31 10:00:00+00:00
    2025-01-31 10:15:00+00:00
    2025-01-31 10:30:00+00:00
    2025-01-31 10:45:00+00:00
    2025-01-31 11:00:00+00:00
    2025-01-31 11:15:00+00:00
    2025-01-31 11:30:00+00:00
    2025-01-31 11:45:00+00:00
    2025-01-31 12:00:00+00:00
    2025-01-31 12:15:00+00:00
    2025-01-31 12:30:00+00:00
    2025-01-31 12:45:00+00:00
    2025-01-31 13:00:00+00:00
    2025-01-31 13:15:00+00:00
    2025-01-31 13:30:00+00:00
    2025-01-31 13:45:00+00:00
    2025-01-31 14:00:00+00:00
    2025-01-31 14:15:00+00:00
    2025-01-31 14:30:00+00:00
    2025-01-31 14:45:00+00:00
    2025-01-31 15:00:00+00:00
    2025-01-31 15:15:00+00:00
    2025-01-31 15:30:00+00:00
    2025-01-31 15:45:00+00:00
    2025-01-31 16:00:00+00:00
    2025-01-31 16:15:00+00:00
    2025-01-31 16:30:00+00:00
    2025-01-31 16:45:00+00:00
    2025-01-31 17:00:00+00:00
    2025-01-31 17:15:00+00:00
    2025-01-31 17:30:00+00:00
    2025-01-31 17:45:00+00:00
    2025-01-31 18:00:00+00:00
    2025-01-31 18:15:00+00:00
    2025-01-31 18:30:00+00:00
    2025-01-31 18:45:00+00:00
    2025-01-31 19:00:00+00:00
    2025-01-31 19:15:00+00:00
```

```
    2025-01-31 19:30:00+00:00
    2025-01-31 19:45:00+00:00
    2025-01-31 20:00:00+00:00
    2025-01-31 20:15:00+00:00
    2025-01-31 20:30:00+00:00
    2025-01-31 20:45:00+00:00
    2025-01-31 21:00:00+00:00
    2025-01-31 21:15:00+00:00
    2025-01-31 21:30:00+00:00
    2025-01-31 22:30:00+00:00
    2025-01-31 22:45:00+00:00
    2025-01-31 23:00:00+00:00
    2025-01-31 23:15:00+00:00
    2025-01-31 23:30:00+00:00
    2025-01-31 23:45:00+00:00
    2025-02-01 10:45:00+00:00
    2025-02-01 11:00:00+00:00
    2025-02-01 11:15:00+00:00
    2025-02-01 11:30:00+00:00
    2025-02-01 11:45:00+00:00
    2025-02-01 12:00:00+00:00
    2025-02-01 12:15:00+00:00
    2025-02-01 12:30:00+00:00
    2025-02-01 12:45:00+00:00
    2025-02-01 13:00:00+00:00
    2025-02-01 13:15:00+00:00
    2025-02-01 13:30:00+00:00
    2025-02-01 13:45:00+00:00
    2025-02-01 14:00:00+00:00
    2025-02-01 14:15:00+00:00
    2025-02-01 14:30:00+00:00
    2025-02-01 14:45:00+00:00
    2025-02-01 15:00:00+00:00
    2025-02-01 15:15:00+00:00
    2025-02-01 15:30:00+00:00
    2025-02-01 15:45:00+00:00
    2025-02-01 16:00:00+00:00
    2025-02-01 16:15:00+00:00
    2025-02-01 16:30:00+00:00
    2025-02-01 21:15:00+00:00
    2025-02-01 21:30:00+00:00
    2025-02-01 22:30:00+00:00

Pattern Analysis for Manchester Racecourse:
  Minor Concern: 269 instances (66.75%)
  Normal: 134 instances (33.25%)

Pattern Analysis for Rochdale:
  Normal: 298 instances (73.95%)
  Major Concern: 96 instances (23.82%)
  Minor Concern: 6 instances (1.49%)
  Moderate Concern: 3 instances (0.74%)

  Dates of Major Concern:
    2025-01-31 03:00:00+00:00
    2025-01-31 03:15:00+00:00
    2025-01-31 03:30:00+00:00
    2025-01-31 03:45:00+00:00
    2025-01-31 04:00:00+00:00
    2025-01-31 04:15:00+00:00
```

2025-01-31 04:30:00+00:00
2025-01-31 04:45:00+00:00
2025-01-31 05:00:00+00:00
2025-01-31 05:15:00+00:00
2025-01-31 05:30:00+00:00
2025-01-31 05:45:00+00:00
2025-01-31 06:00:00+00:00
2025-01-31 06:15:00+00:00
2025-01-31 06:30:00+00:00
2025-01-31 06:45:00+00:00
2025-01-31 07:00:00+00:00
2025-01-31 07:15:00+00:00
2025-01-31 07:30:00+00:00
2025-01-31 07:45:00+00:00
2025-01-31 08:00:00+00:00
2025-01-31 08:15:00+00:00
2025-01-31 08:30:00+00:00
2025-01-31 08:45:00+00:00
2025-01-31 09:00:00+00:00
2025-01-31 10:00:00+00:00
2025-01-31 10:15:00+00:00
2025-01-31 10:30:00+00:00
2025-01-31 10:45:00+00:00
2025-01-31 11:00:00+00:00
2025-01-31 11:15:00+00:00
2025-01-31 11:30:00+00:00
2025-01-31 11:45:00+00:00
2025-01-31 12:00:00+00:00
2025-01-31 12:15:00+00:00
2025-01-31 12:30:00+00:00
2025-01-31 12:45:00+00:00
2025-01-31 13:00:00+00:00
2025-01-31 13:15:00+00:00
2025-01-31 13:30:00+00:00
2025-01-31 13:45:00+00:00
2025-01-31 14:00:00+00:00
2025-01-31 14:15:00+00:00
2025-01-31 14:30:00+00:00
2025-01-31 14:45:00+00:00
2025-01-31 15:00:00+00:00
2025-01-31 15:15:00+00:00
2025-01-31 15:30:00+00:00
2025-01-31 15:45:00+00:00
2025-01-31 16:00:00+00:00
2025-01-31 16:15:00+00:00
2025-01-31 16:30:00+00:00
2025-01-31 16:45:00+00:00
2025-01-31 17:00:00+00:00
2025-01-31 17:15:00+00:00
2025-01-31 17:30:00+00:00
2025-01-31 17:45:00+00:00
2025-01-31 18:00:00+00:00
2025-01-31 18:15:00+00:00
2025-01-31 18:30:00+00:00
2025-01-31 18:45:00+00:00
2025-01-31 19:00:00+00:00
2025-01-31 19:15:00+00:00
2025-01-31 19:30:00+00:00
2025-01-31 19:45:00+00:00
2025-01-31 20:00:00+00:00

```
    2025-01-31 20:15:00+00:00
    2025-01-31 20:30:00+00:00
    2025-01-31 20:45:00+00:00
    2025-01-31 21:00:00+00:00
    2025-01-31 21:15:00+00:00
    2025-01-31 21:30:00+00:00
    2025-01-31 22:30:00+00:00
    2025-01-31 22:45:00+00:00
    2025-01-31 23:00:00+00:00
    2025-01-31 23:15:00+00:00
    2025-01-31 23:30:00+00:00
    2025-01-31 23:45:00+00:00
    2025-02-01 10:45:00+00:00
    2025-02-01 11:00:00+00:00
    2025-02-01 11:15:00+00:00
    2025-02-01 11:30:00+00:00
    2025-02-01 11:45:00+00:00
    2025-02-01 12:00:00+00:00
    2025-02-01 12:15:00+00:00
    2025-02-01 12:30:00+00:00
    2025-02-01 12:45:00+00:00
    2025-02-01 13:00:00+00:00
    2025-02-01 13:15:00+00:00
    2025-02-01 13:30:00+00:00
    2025-02-01 13:45:00+00:00
    2025-02-01 14:00:00+00:00
    2025-02-01 14:15:00+00:00
    2025-02-01 14:30:00+00:00
    2025-02-01 14:45:00+00:00
    2025-02-01 15:00:00+00:00

Pattern recognition complete. Results saved to CSV files.
```

Correlation Analysis

In [112…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Load the pattern results
stations = ['Bury_Ground', 'Manchester_Racecourse', 'Rochdale']
pattern_data = {}

for station in stations:
    pattern_data[station] = pd.read_csv(f'C:/Users/Administrator/NEWPROJECT/clea
    pattern_data[station]['river_timestamp'] = pd.to_datetime(pattern_data[stati
    pattern_data[station].set_index('river_timestamp', inplace=True)
    # Resample and aggregate data
    pattern_data[station] = pattern_data[station].resample('15min').agg({
        'river_level': 'mean',
        'pattern_score': 'max'
    })

def analyze_correlations(pattern_data):
    # Align data from all stations
    aligned_data = pd.DataFrame({
        f'{station}_level': pattern_data[station]['river_level']
        for station in stations
```

```
    })
    aligned_data = aligned_data.dropna()  # Remove any rows with missing data

    # Calculate correlations
    level_correlations = aligned_data.corr()

    # Calculate lag correlations
    lag_correlations = {}
    max_lag = 12  # 3 hours (12 * 15-minute intervals)
    for station1 in stations:
        for station2 in stations:
            if station1 != station2:
                ccf = []
                for lag in range(-max_lag, max_lag+1):
                    if lag >= 0:
                        corr = aligned_data[f'{station1}_level'].iloc[lag:].corr
                    else:
                        corr = aligned_data[f'{station1}_level'].iloc[:lag].corr
                    ccf.append(corr)
                lag_correlations[f'{station1}-{station2}'] = ccf

    return level_correlations, lag_correlations

# Perform correlation analysis
level_corr, lag_corr = analyze_correlations(pattern_data)

# Print results
print("River Level Correlations:")
print(level_corr)

# Plot lag correlations
plt.figure(figsize=(12, 8))
for pair, ccf in lag_corr.items():
    plt.plot(range(-12, 13), ccf, label=pair)
plt.xlabel('Lag (15-minute intervals)')
plt.ylabel('Correlation Coefficient')
plt.title('Lag Correlations between Stations')
plt.legend()
plt.grid(True)
plt.savefig('C:/Users/Administrator/NEWPROJECT/cleaned_data/lag_correlations.png
plt.close()

print("\nLag correlation plot saved as 'lag_correlations.png'")
```

```
River Level Correlations:
                             Bury_Ground_level  Manchester_Racecourse_level  \
Bury_Ground_level                     1.000000                     0.949021
Manchester_Racecourse_level           0.949021                     1.000000
Rochdale_level                        0.974711                     0.920792

                             Rochdale_level
Bury_Ground_level                  0.974711
Manchester_Racecourse_level        0.920792
Rochdale_level                     1.000000

Lag correlation plot saved as 'lag_correlations.png'
```

In [114…

```
import pandas as pd
import numpy as np
from scipy import stats
```

```
# Load the pattern results
stations = ['Bury_Ground', 'Manchester_Racecourse', 'Rochdale']
pattern_data = {}

for station in stations:
    pattern_data[station] = pd.read_csv(f'C:/Users/Administrator/NEWPROJECT/clea
    pattern_data[station]['river_timestamp'] = pd.to_datetime(pattern_data[stati
    pattern_data[station].set_index('river_timestamp', inplace=True)

# Use the correlation matrix we already have
level_corr = pd.DataFrame({
    'Bury_Ground_level': [1.000000, 0.949021, 0.974711],
    'Manchester_Racecourse_level': [0.949021, 1.000000, 0.920792],
    'Rochdale_level': [0.974711, 0.920792, 1.000000]
}, index=['Bury_Ground_level', 'Manchester_Racecourse_level', 'Rochdale_level'])

def enhanced_anomaly_detection(pattern_data, level_corr, correlation_threshold=0
    anomalies = {}

    for station in stations:
        station_data = pattern_data[station]
        other_stations = [s for s in stations if s != station]

        # Identify potential anomalies based on pattern score
        potential_anomalies = station_data[station_data['pattern_score'] > stati

        validated_anomalies = []
        for idx, row in potential_anomalies.iterrows():
            # Check if other highly correlated stations also show anomalies with
            correlated_anomaly = False
            for other_station in other_stations:
                if level_corr.loc[f'{station}_level', f'{other_station}_level']
                    other_data = pattern_data[other_station]
                    lag_window = other_data.loc[idx - pd.Timedelta(minutes=15*la
                    if not lag_window.empty and (lag_window['pattern_score'] > c
                        correlated_anomaly = True
                        break

            if correlated_anomaly:
                validated_anomalies.append({
                    'timestamp': idx.strftime('%Y-%m-%d %H:%M:%S'),
                    'river_level': row['river_level'],
                    'pattern_score': row['pattern_score']
                })

        anomalies[station] = validated_anomalies

    return anomalies

# Run enhanced anomaly detection
enhanced_anomalies = enhanced_anomaly_detection(pattern_data, level_corr)

# Print results
for station, anomalies in enhanced_anomalies.items():
    print(f"\nValidated Anomalies for {station}:")
    for anomaly in anomalies[:5]:  # Print first 5 anomalies
        print(f"Timestamp: {anomaly['timestamp']}, Level: {anomaly['river_level'
    if len(anomalies) > 5:
        print(f"... and {len(anomalies) - 5} more anomalies.")
```

```
# Save enhanced anomalies
import json
with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/enhanced_anomalies.jso
    json.dump(enhanced_anomalies, f, indent=2)

print("\nEnhanced anomalies saved to 'enhanced_anomalies.json'")
```

```
Validated Anomalies for Bury_Ground:

Validated Anomalies for Manchester_Racecourse:

Validated Anomalies for Rochdale:

Enhanced anomalies saved to 'enhanced_anomalies.json'
```

In [115…

```
def enhanced_anomaly_detection(pattern_data, level_corr, correlation_threshold=0
    anomalies = {}

    for station in stations:
        station_data = pattern_data[station]
        other_stations = [s for s in stations if s != station]

        # Identify potential anomalies based on pattern score
        potential_anomalies = station_data[station_data['pattern_score'] > stati

        validated_anomalies = []
        for idx, row in potential_anomalies.iterrows():
            # Check if other highly correlated stations also show anomalies with
            correlated_anomaly = False
            for other_station in other_stations:
                if level_corr.loc[f'{station}_level', f'{other_station}_level']
                    other_data = pattern_data[other_station]
                    lag_window = other_data.loc[idx - pd.Timedelta(minutes=15*la
                    if not lag_window.empty and (lag_window['pattern_score'] > c
                        correlated_anomaly = True
                        break

            if correlated_anomaly:
                validated_anomalies.append({
                    'timestamp': idx.strftime('%Y-%m-%d %H:%M:%S'),
                    'river_level': row['river_level'],
                    'pattern_score': row['pattern_score']
                })

        anomalies[station] = validated_anomalies

    return anomalies

# Run enhanced anomaly detection with adjusted parameters
enhanced_anomalies = enhanced_anomaly_detection(pattern_data, level_corr, correl

# Print results
for station, anomalies in enhanced_anomalies.items():
    print(f"\nValidated Anomalies for {station}:")
    for anomaly in anomalies[:5]:  # Print first 5 anomalies
        print(f"Timestamp: {anomaly['timestamp']}, Level: {anomaly['river_level'
    if len(anomalies) > 5:
        print(f"... and {len(anomalies) - 5} more anomalies.")
```

```
# Save enhanced anomalies
import json
with open('C:/Users/Administrator/NEWPROJECT/cleaned_data/enhanced_anomalies.jso
    json.dump(enhanced_anomalies, f, indent=2)

print("\nEnhanced anomalies saved to 'enhanced_anomalies.json'")
```

```
Validated Anomalies for Bury_Ground:
Timestamp: 2025-01-31 05:45:00, Level: 0.431, Score: 13.000
Timestamp: 2025-01-31 06:00:00, Level: 0.435, Score: 14.000
Timestamp: 2025-01-31 06:15:00, Level: 0.438, Score: 14.000
Timestamp: 2025-01-31 06:30:00, Level: 0.440, Score: 14.000
Timestamp: 2025-01-31 06:45:00, Level: 0.441, Score: 14.000
... and 83 more anomalies.

Validated Anomalies for Manchester_Racecourse:

Validated Anomalies for Rochdale:
Timestamp: 2025-01-31 04:15:00, Level: 0.278, Score: 16.000
Timestamp: 2025-01-31 04:30:00, Level: 0.282, Score: 17.000
Timestamp: 2025-01-31 04:45:00, Level: 0.284, Score: 17.000
Timestamp: 2025-01-31 05:00:00, Level: 0.286, Score: 17.000
Timestamp: 2025-01-31 05:15:00, Level: 0.288, Score: 17.000
... and 83 more anomalies.

Enhanced anomalies saved to 'enhanced_anomalies.json'
```

In [116…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the pattern results
stations = ['Bury_Ground', 'Manchester_Racecourse', 'Rochdale']
pattern_data = {}

for station in stations:
    pattern_data[station] = pd.read_csv(f'C:/Users/Administrator/NEWPROJECT/clea
    pattern_data[station]['river_timestamp'] = pd.to_datetime(pattern_data[stati
    pattern_data[station].set_index('river_timestamp', inplace=True)

# Statistical summary
print("Statistical Summary of Pattern Scores:")
for station in stations:
    print(f"\n{station}:")
    print(pattern_data[station]['pattern_score'].describe())

# Visualize data distributions
plt.figure(figsize=(15, 5))
for i, station in enumerate(stations, 1):
    plt.subplot(1, 3, i)
    sns.histplot(pattern_data[station]['pattern_score'], kde=True)
    plt.title(f'{station} Pattern Score Distribution')
    plt.xlabel('Pattern Score')
plt.tight_layout()
plt.savefig('C:/Users/Administrator/NEWPROJECT/cleaned_data/pattern_score_distri
plt.close()

# Time series plots
plt.figure(figsize=(15, 10))
for i, station in enumerate(stations, 1):
```

```
    plt.subplot(3, 1, i)
    plt.plot(pattern_data[station].index, pattern_data[station]['pattern_score']
    plt.title(f'{station} Pattern Score Time Series')
    plt.xlabel('Time')
    plt.ylabel('Pattern Score')
plt.tight_layout()
plt.savefig('C:/Users/Administrator/NEWPROJECT/cleaned_data/pattern_score_timese
plt.close()

# Calculate thresholds
for station in stations:
    mean_score = pattern_data[station]['pattern_score'].mean()
    std_score = pattern_data[station]['pattern_score'].std()
    threshold = mean_score + 1.5 * std_score
    print(f"\n{station} Threshold:")
    print(f"Mean: {mean_score:.3f}")
    print(f"Std Dev: {std_score:.3f}")
    print(f"Threshold (Mean + 1.5*Std): {threshold:.3f}")

print("\nPlots saved as 'pattern_score_distributions.png' and 'pattern_score_tim
```

```
Statistical Summary of Pattern Scores:

Bury_Ground:
count    403.000000
mean       3.334988
std        5.906354
min        0.000000
25%        0.000000
50%        0.000000
75%        0.500000
max       14.000000
Name: pattern_score, dtype: float64

Manchester_Racecourse:
count    403.000000
mean       0.952854
std        0.843624
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        3.000000
Name: pattern_score, dtype: float64

Rochdale:
count    403.000000
mean       4.049628
std        7.110886
min        0.000000
25%        0.000000
50%        0.000000
75%        3.500000
max       17.000000
Name: pattern_score, dtype: float64

Bury_Ground Threshold:
Mean: 3.335
Std Dev: 5.906
Threshold (Mean + 1.5*Std): 12.195

Manchester_Racecourse Threshold:
Mean: 0.953
Std Dev: 0.844
Threshold (Mean + 1.5*Std): 2.218

Rochdale Threshold:
Mean: 4.050
Std Dev: 7.111
Threshold (Mean + 1.5*Std): 14.716

Plots saved as 'pattern_score_distributions.png' and 'pattern_score_timeseries.pn
g'
```

Environmental Factor Integration

In [118…

```python
import pandas as pd
import numpy as np
from scipy import stats
```

```
# Load the pattern results and rainfall data
stations = ['Bury_Ground', 'Manchester_Racecourse', 'Rochdale']
pattern_data = {}
rainfall_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/proc
rainfall_data['Date'] = pd.to_datetime(rainfall_data['Date']).dt.tz_localize('UT

for station in stations:
    pattern_data[station] = pd.read_csv(f'C:/Users/Administrator/NEWPROJECT/clea
    pattern_data[station]['river_timestamp'] = pd.to_datetime(pattern_data[stati
    pattern_data[station].set_index('river_timestamp', inplace=True)

    # Merge rainfall data
    station_rainfall = rainfall_data[rainfall_data['station'] == station]
    pattern_data[station] = pattern_data[station].merge(station_rainfall[['Date'
                                                        left_index=True, right_o
    pattern_data[station].set_index('Date', inplace=True)

def environmental_anomaly_detection(pattern_data, rainfall_threshold=10, seasona
    anomalies = {}

    for station in stations:
        station_data = pattern_data[station]

        # Add seasonal factor (assuming higher sensitivity in winter and autumn)
        station_data['seasonal_factor'] = station_data.index.month.map(
            lambda m: seasonal_factor if m in [12, 1, 2, 9, 10, 11] else 1
        )

        # Identify potential anomalies based on pattern score and rainfall
        potential_anomalies = station_data[
            (station_data['pattern_score'] > station_data['pattern_score'].mean(
             1.5 * station_data['pattern_score'].std() * station_data['seasonal_
            (station_data['Rainfall'] > rainfall_threshold)
        ]

        anomalies[station] = potential_anomalies

    return anomalies

# Run environmental anomaly detection
environmental_anomalies = environmental_anomaly_detection(pattern_data)

# Print results
for station, anomalies in environmental_anomalies.items():
    print(f"\nEnvironmental Anomalies for {station}:")
    print(f"Total anomalies: {len(anomalies)}")
    if not anomalies.empty:
        print(anomalies[['river_level', 'pattern_score', 'Rainfall']].head())

# Save environmental anomalies
for station, anomalies in environmental_anomalies.items():
    anomalies.to_csv(f'C:/Users/Administrator/NEWPROJECT/cleaned_data/environmen

print("\nEnvironmental anomalies saved to CSV files.")
```

```
Environmental Anomalies for Bury_Ground:
Total anomalies: 91
                           river_level  pattern_score  Rainfall
Date
2025-01-31 06:00:00+00:00        0.435           14.0       NaN
2025-01-31 06:15:00+00:00        0.438           14.0       NaN
2025-01-31 06:30:00+00:00        0.440           14.0       NaN
2025-01-31 06:45:00+00:00        0.441           14.0       NaN
2025-01-31 07:00:00+00:00        0.441           14.0       NaN

Environmental Anomalies for Manchester_Racecourse:
Total anomalies: 19
                           river_level  pattern_score  Rainfall
Date
2025-02-03 02:00:00+00:00        0.999            3.0       NaN
2025-02-03 02:15:00+00:00        1.001            3.0       NaN
2025-02-03 02:30:00+00:00        0.997            3.0       NaN
2025-02-03 02:45:00+00:00        0.998            3.0       NaN
2025-02-03 03:00:00+00:00        0.994            3.0       NaN

Environmental Anomalies for Rochdale:
Total anomalies: 87
                           river_level  pattern_score  Rainfall
Date
2025-01-31 04:30:00+00:00        0.282           17.0       NaN
2025-01-31 04:45:00+00:00        0.284           17.0       NaN
2025-01-31 05:00:00+00:00        0.286           17.0       NaN
2025-01-31 05:15:00+00:00        0.288           17.0       NaN
2025-01-31 05:30:00+00:00        0.290           17.0       NaN

Environmental anomalies saved to CSV files.
```

In [119…

```
import pandas as pd

# Load the rainfall data
rainfall_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/proc
rainfall_data['Date'] = pd.to_datetime(rainfall_data['Date'])

# Display basic information about the rainfall data
print("Rainfall Data Information:")
print(rainfall_data.info())

print("\nSample of Rainfall Data:")
print(rainfall_data.head())

print("\nDate Range of Rainfall Data:")
print(f"Start: {rainfall_data['Date'].min()}")
print(f"End: {rainfall_data['Date'].max()}")

# Check for missing values
print("\nMissing Values in Rainfall Data:")
print(rainfall_data.isnull().sum())

# Display summary statistics
print("\nRainfall Summary Statistics:")
print(rainfall_data.groupby('station')['Rainfall'].describe())
```

```
Rainfall Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21550 entries, 0 to 21549
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      21550 non-null  datetime64[ns]
 1   Rainfall  21550 non-null  float64
 2   station   21550 non-null  object
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 505.2+ KB
None

Sample of Rainfall Data:
        Date  Rainfall      station
0 1961-01-01       9.4  Bury Ground
1 1961-01-02      13.7  Bury Ground
2 1961-01-03       3.0  Bury Ground
3 1961-01-04       0.1  Bury Ground
4 1961-01-05      13.0  Bury Ground

Date Range of Rainfall Data:
Start: 1961-01-01 00:00:00
End: 2017-12-31 00:00:00

Missing Values in Rainfall Data:
Date        0
Rainfall    0
station     0
dtype: int64

Rainfall Summary Statistics:
               count      mean       std  min  25%  50%   75%   max
station
Bury Ground  20819.0  3.775498  6.209935  0.0  0.0  0.9  5.10  79.5
Rochdale       731.0  3.783584  5.848199  0.0  0.0  0.9  5.35  36.6
```

In [127…

```
import pandas as pd
import os

# Check if Manchester Racecourse data exists
data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/'

# Check for pattern results
pattern_file = os.path.join(data_dir, 'pattern_results_Manchester_Racecourse.csv
print("Pattern Results File Exists:", os.path.exists(pattern_file))

# Check for rainfall data
rainfall_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/proc
print("\nRainfall Data Stations:")
print(rainfall_data['station'].unique())

# If Manchester data is missing, we should discuss data collection
if not os.path.exists(pattern_file):
    print("\nWARNING: Manchester Racecourse data is missing!")
    print("Steps to resolve:")
    print("1. Verify data collection process")
    print("2. Check original data sources")
    print("3. Recreate data collection scripts if necessary")
```

```
Pattern Results File Exists: True

Rainfall Data Stations:
['Bury Ground' 'Rochdale']
```

In [129…

```
import pandas as pd
import numpy as np

# Load the pattern results
stations = ['Bury_Ground', 'Manchester_Racecourse', 'Rochdale']
pattern_data = {}

# Load rainfall data
rainfall_data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/proc
rainfall_data['Date'] = pd.to_datetime(rainfall_data['Date'])

# Create a station name mapping
station_name_map = {
    'Bury Ground': 'Bury_Ground',
    'Manchester Racecourse': 'Manchester_Racecourse'
}

# Modify station names in rainfall data
rainfall_data['station'] = rainfall_data['station'].replace(station_name_map)

def enhanced_environmental_anomaly_detection(stations, rainfall_data):
    comprehensive_anomalies = {}

    for station in stations:
        # Read pattern results
        pattern_file = f'C:/Users/Administrator/NEWPROJECT/cleaned_data/pattern_
        df = pd.read_csv(pattern_file)
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

        # Get rainfall data for the station
        station_rainfall = rainfall_data[rainfall_data['station'] == station]
        station_rainfall_stats = {
            'mean': station_rainfall['Rainfall'].mean() if not station_rainfall.
            'std': station_rainfall['Rainfall'].std() if not station_rainfall.em
        }

        # Analyze river level changes
        df['level_change'] = df['river_level'].diff()

        # Multiple criteria for anomaly detection
        anomaly_criteria = [
            # Significant pattern score
            (df['pattern_score'] > df['pattern_score'].mean() + 1.5 * df['patter

            # Rapid level changes
            (abs(df['level_change']) > df['level_change'].std() * 2),

            # Extended periods of unusual levels
            (abs(df['river_level'] - df['river_level'].mean()) > df['river_level
        ]

        # Combine anomaly criteria
        df['is_anomaly'] = np.any(anomaly_criteria, axis=0)

        # Extract anomalies
```

```
        anomalies = df[df['is_anomaly']].copy()

        # Detailed anomaly analysis
        anomaly_summary = {
            'total_anomalies': len(anomalies),
            'mean_river_level': anomalies['river_level'].mean(),
            'max_river_level': anomalies['river_level'].max(),
            'min_river_level': anomalies['river_level'].min(),
            'mean_level_change': anomalies['level_change'].mean(),
            'rainfall_stats': station_rainfall_stats
        }

        # Temporal analysis of anomalies
        if not anomalies.empty:
            # Create custom bins
            start = anomalies['river_timestamp'].min()
            end = anomalies['river_timestamp'].max()
            total_duration = (end - start).total_seconds()

            # Create 4 equal time periods
            bins = [
                start,
                start + pd.Timedelta(seconds=total_duration * 0.25),
                start + pd.Timedelta(seconds=total_duration * 0.5),
                start + pd.Timedelta(seconds=total_duration * 0.75),
                end
            ]

            labels = ['Early', 'Mid-Early', 'Mid-Late', 'Late']

            anomalies['anomaly_period'] = pd.cut(
                anomalies['river_timestamp'],
                bins=bins,
                labels=labels
            )

            period_distribution = anomalies['anomaly_period'].value_counts(norma
        else:
            period_distribution = pd.Series()

        comprehensive_anomalies[station] = {
            'anomalies': anomalies,
            'summary': anomaly_summary,
            'period_distribution': period_distribution
        }

    return comprehensive_anomalies

# Run enhanced environmental anomaly detection
environmental_anomalies = enhanced_environmental_anomaly_detection(stations, rai

# Print detailed results
for station, data in environmental_anomalies.items():
    print(f"\n{station} Comprehensive Anomaly Analysis:")
    print("\nSummary Statistics:")
    for key, value in data['summary'].items():
        print(f"{key}: {value}")

    print("\nAnomaly Period Distribution:")
    print(data['period_distribution'])
```

```
    print("\nTop 5 Anomalies:")
    if not data['anomalies'].empty:
        top_anomalies = data['anomalies'].sort_values('pattern_score', ascending
        print(top_anomalies[['river_timestamp', 'river_level', 'level_change', '
    else:
        print("No anomalies detected.")

# Save detailed anomalies
for station, data in environmental_anomalies.items():
    if not data['anomalies'].empty:
        data['anomalies'].to_csv(f'C:/Users/Administrator/NEWPROJECT/cleaned_dat
```

```
Bury_Ground Comprehensive Anomaly Analysis:

Summary Statistics:
total_anomalies: 97
mean_river_level: 0.40063917525773196
max_river_level: 0.441
min_river_level: 0.365
mean_level_change: -0.0002989690721649487
rainfall_stats: {'mean': 3.7754983428598874, 'std': 6.209935248255402}

Anomaly Period Distribution:
anomaly_period
Early        0.385417
Mid-Early    0.343750
Late         0.239583
Mid-Late     0.031250
Name: proportion, dtype: float64

Top 5 Anomalies:
              river_timestamp  river_level  level_change  pattern_score  \
115 2025-01-31 17:30:00+00:00        0.400        -0.002           14.0
135 2025-01-31 23:15:00+00:00        0.392         0.000           14.0
133 2025-01-31 22:45:00+00:00        0.393         0.000           14.0
132 2025-01-31 22:30:00+00:00        0.393        -0.002           14.0
131 2025-01-31 21:30:00+00:00        0.395         0.000           14.0

    anomaly_period
115      Mid-Early
135      Mid-Early
133      Mid-Early
132      Mid-Early
131      Mid-Early

Manchester_Racecourse Comprehensive Anomaly Analysis:

Summary Statistics:
total_anomalies: 83
mean_river_level: 1.1177349397590362
max_river_level: 1.203
min_river_level: 0.984
mean_level_change: -0.0009759036144578282
rainfall_stats: {'mean': None, 'std': None}

Anomaly Period Distribution:
anomaly_period
Early        0.707317
Late         0.231707
Mid-Late     0.036585
Mid-Early    0.024390
Name: proportion, dtype: float64

Top 5 Anomalies:
              river_timestamp  river_level  level_change  pattern_score  \
289 2025-02-03 06:30:00+00:00        0.984        -0.003            3.0
280 2025-02-03 04:15:00+00:00        0.990        -0.002            3.0
273 2025-02-03 02:30:00+00:00        0.997        -0.004            3.0
274 2025-02-03 02:45:00+00:00        0.998         0.001            3.0
275 2025-02-03 03:00:00+00:00        0.994        -0.004            3.0

    anomaly_period
```

```
289          Late
280          Late
273          Late
274          Late
275          Late

Rochdale Comprehensive Anomaly Analysis:

Summary Statistics:
total_anomalies: 98
mean_river_level: 0.2589183673469388
max_river_level: 0.293
min_river_level: 0.221
mean_level_change: -8.163265306122456e-05
rainfall_stats: {'mean': 3.783584131326949, 'std': 5.848198763742644}

Anomaly Period Distribution:
anomaly_period
Early        0.412371
Mid-Early    0.412371
Late         0.164948
Mid-Late     0.010309
Name: proportion, dtype: float64

Top 5 Anomalies:
              river_timestamp  river_level  level_change  pattern_score  \
105 2025-01-31 15:00:00+00:00        0.261        -0.001           17.0
117 2025-01-31 18:00:00+00:00        0.257         0.000           17.0
126 2025-01-31 20:15:00+00:00        0.250         0.000           17.0
125 2025-01-31 20:00:00+00:00        0.250        -0.001           17.0
124 2025-01-31 19:45:00+00:00        0.251        -0.001           17.0

    anomaly_period
105      Mid-Early
117      Mid-Early
126      Mid-Early
125      Mid-Early
124      Mid-Early
```

Communication protocols

In [133…

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import requests
import json
from dataclasses import dataclass, asdict
from typing import List, Optional

@dataclass
class Contact:
    name: str
    email: Optional[str] = None
    phone: Optional[str] = None
    preferred_method: str = 'email'  # 'email', 'sms', 'both'

class NotificationSystem:
    def __init__(self, config):
        """
```

```
        Initialize notification system with configuration

        config should include:
        - email settings
        - SMS gateway settings
        - emergency contact lists
        """
        self.config = config
        self.contacts = self._load_contacts()

    def _load_contacts(self):
        """
        Load emergency contacts from a configuration file or database
        """
        # In a real-world scenario, this would pull from a database or config fi
        return [
            Contact(
                name="Local Emergency Management",
                email="emergency@localgovernment.org",
                phone="+441234567890",
                preferred_method='both'
            ),
            Contact(
                name="Flood Response Team",
                email="floodresponse@localauthority.gov.uk",
                phone="+447890123456",
                preferred_method='email'
            )
        ]

    def send_notifications(self, alert):
        """
        Send notifications based on alert level and contact preferences
        """
        # Determine notification method based on alert level
        notification_methods = self._select_notification_methods(alert)

        # Send notifications
        for contact in self.contacts:
            for method in notification_methods:
                try:
                    if method == 'email':
                        self._send_email_notification(contact, alert)
                    elif method == 'sms':
                        self._send_sms_notification(contact, alert)
                except Exception as e:
                    print(f"Failed to send {method} notification to {contact.nam

    def _select_notification_methods(self, alert):
        """
        Determine appropriate notification methods based on alert level
        """
        alert_level = alert['overall_alert_level']

        # Notification escalation matrix
        notification_matrix = {
            '1': [],  # Normal - No notifications
            '2': ['email'],  # Advisory - Email notification
            '3': ['email', 'sms'],  # Warning - Email and SMS
            '4': ['email', 'sms']  # Critical - Urgent email and SMS
```

```
        }

        return notification_matrix.get(alert_level, [])

    def _send_email_notification(self, contact, alert):
        """
        Send email notification
        """
        if not contact.email:
            return

        # Create email message
        msg = MIMEMultipart()
        msg['From'] = self.config['email']['sender_email']
        msg['To'] = contact.email
        msg['Subject'] = alert['alert_message']['summary']

        # Create email body
        body = self._create_notification_body(alert, 'email')
        msg.attach(MIMEText(body, 'plain'))

        # Send email
        try:
            with smtplib.SMTP(
                self.config['email']['smtp_server'],
                self.config['email']['smtp_port']
            ) as server:
                server.starttls()
                server.login(
                    self.config['email']['username'],
                    self.config['email']['password']
                )
                server.send_message(msg)
            print(f"Email sent to {contact.name}")
        except Exception as e:
            print(f"Email sending failed: {e}")

    def _send_sms_notification(self, contact, alert):
        """
        Send SMS notification via third-party SMS gateway
        """
        if not contact.phone:
            return

        # Create SMS body
        body = self._create_notification_body(alert, 'sms')

        # Send SMS via third-party gateway (example with hypothetical API)
        try:
            response = requests.post(
                self.config['sms_gateway']['url'],
                headers={
                    'Authorization': f"Bearer {self.config['sms_gateway']['api_k
                    'Content-Type': 'application/json'
                },
                data=json.dumps({
                    'to': contact.phone,
                    'message': body
                })
            )
```

```
            if response.status_code == 200:
                print(f"SMS sent to {contact.name}")
            else:
                print(f"SMS sending failed: {response.text}")
        except Exception as e:
            print(f"SMS sending failed: {e}")

    def _create_notification_body(self, alert, method_type='email'):
        """
        Create notification message body

        Different formats for email and SMS
        """
        # Base notification content
        message = alert['alert_message']

        if method_type == 'email':
            # Detailed email notification
            body = f"""
            FLOOD EARLY WARNING SYSTEM ALERT

            Station: {alert['station']}
            Alert Level: {message['summary']}

            Description: {message['description']}

            Detailed Factors:
            {', '.join(message['detailed_factors'])}

            River Level: {alert['river_level']} m
            Level Change: {alert['level_change']} m
            Pattern Score: {alert['pattern_score']}
            Rainfall: {alert['rainfall']} mm

            Please take appropriate action and stay informed.
            """
        else:
            # Concise SMS notification
            body = (
                f"FLOOD ALERT: {alert['station']} - {message['summary']}. "
                f"Level: {alert['river_level']} m, "
                f"Change: {alert['level_change']} m. "
                "Take precautions."
            )

        return body

# Configuration for notification system
notification_config = {
    'email': {
        'smtp_server': 'smtp.gmail.com',
        'smtp_port': 587,
        'sender_email': 'your-email@gmail.com',
        'username': 'your-email@gmail.com',
        'password': 'your-app-password'
    },
    'sms_gateway': {
        'url': 'https://api.smsprovider.com/send',
        'api_key': 'your-sms-gateway-api-key'
```

```
    }
}

# Example usage with previous AlertSystem
def test_notification_system(alert_system, notification_system):
    # Test scenarios for Bury Ground
    test_cases = [
        {
            'station': 'Bury_Ground',
            'river_level': 0.380,
            'level_change': 0.001,
            'pattern_score': 3,
            'rainfall': 2
        },
        {
            'station': 'Bury_Ground',
            'river_level': 0.425,
            'level_change': 0.005,
            'pattern_score': 12,
            'rainfall': 15
        }
    ]

    for case in test_cases:
        # Generate alert
        alert = alert_system.generate_alert(
            case['station'],
            case['river_level'],
            case['level_change'],
            case['pattern_score'],
            case['rainfall']
        )

        # Send notifications based on alert
        notification_system.send_notifications(alert)

# Uncomment and modify with actual configuration to test
# notification_system = NotificationSystem(notification_config)
# test_notification_system(alert_system, notification_system)
```

In [134…

```
import os
import logging
from typing import List, Dict, Optional
from dataclasses import dataclass
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import requests

# Logging configuration
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    filename='flood_alert_notifications.log'
)
logger = logging.getLogger('FloodAlertNotificationSystem')

@dataclass
class AlertNotification:
    """
```

```
    Structured data class for flood alerts
    """
    station: str
    alert_level: str
    river_level: float
    level_change: float
    pattern_score: float
    rainfall: Optional[float] = None
    timestamp: Optional[str] = None

class NotificationDispatcher:
    """
    Primary notification dispatcher with multiple communication channels
    """
    def __init__(self, config: Dict[str, List[str]]):
        """
        Initialize notification system with configuration
        """
        self.config = config
        self.notification_methods = {
            'email': self._send_email,
            'phone': self._send_sms
        }

    def send_alert(self, alert: AlertNotification):
        """
        Send alerts through configured notification channels
        """
        try:
            # Log the alert
            logger.info(f"Alert generated: {alert}")

            # Determine alert severity
            severity_map = {
                'NORMAL': 1,
                'ADVISORY': 2,
                'WARNING': 3,
                'CRITICAL': 4
            }
            severity = severity_map.get(alert.alert_level.upper(), 1)

            # Send notifications based on severity
            if severity >= 2:  # Advisory and above
                for channel, contacts in self.config.items():
                    if channel in self.notification_methods:
                        for contact in contacts:
                            try:
                                self.notification_methods[channel](contact, aler
                            except Exception as e:
                                logger.error(f"Failed to send {channel} notifica

        except Exception as e:
            logger.error(f"Critical error in send_alert: {e}")

    def _send_email(self, email: str, alert: AlertNotification):
        """
        Send email notification using Gmail SMTP
        """
        try:
            # Email configuration (you'll need to replace with your actual Gmail
```

```
            sender_email = "your_sending_email@gmail.com"
            sender_password = "your_app_password"  # Use App Password, not regul

            # Create message
            msg = MIMEMultipart()
            msg['From'] = sender_email
            msg['To'] = email
            msg['Subject'] = f"Flood Alert: {alert.station} - {alert.alert_level

            # Email body
            body = f"""
            Flood Alert Notification

            Station: {alert.station}
            Alert Level: {alert.alert_level}

            Details:
            - River Level: {alert.river_level} m
            - Level Change: {alert.level_change} m
            - Pattern Score: {alert.pattern_score}
            - Rainfall: {alert.rainfall or 'N/A'} mm

            Please take appropriate precautions.
            """

            msg.attach(MIMEText(body, 'plain'))

            # Send email
            with smtplib.SMTP('smtp.gmail.com', 587) as server:
                server.starttls()
                server.login(sender_email, sender_password)
                server.send_message(msg)

            logger.info(f"Email sent to {email}")
            print(f"[EMAIL] Sent to {email}")

        except Exception as e:
            logger.error(f"Email sending failed: {e}")
            print(f"Email sending failed: {e}")

    def _send_sms(self, phone: str, alert: AlertNotification):
        """
        Send SMS using Twilio (you'll need to sign up for a Twilio account)
        """
        try:
            # Twilio configuration (you'll need to replace with your Twilio cred
            twilio_account_sid = 'your_twilio_account_sid'
            twilio_auth_token = 'your_twilio_auth_token'
            twilio_phone_number = 'your_twilio_phone_number'

            # SMS body
            sms_body = (
                f"FLOOD ALERT: {alert.station} - {alert.alert_level}. "
                f"River Level: {alert.river_level} m. "
                "Take immediate precautions."
            )

            # Send SMS using Twilio API
            url = f"https://api.twilio.com/2010-04-01/Accounts/{twilio_account_s
            payload = {
```

```
                'From': twilio_phone_number,
                'To': phone,
                'Body': sms_body
            }

            response = requests.post(
                url,
                auth=(twilio_account_sid, twilio_auth_token),
                data=payload
            )

            if response.status_code == 201:
                logger.info(f"SMS sent to {phone}")
                print(f"[SMS] Sent to {phone}")
            else:
                logger.error(f"SMS sending failed: {response.text}")
                print(f"SMS sending failed: {response.text}")

        except Exception as e:
            logger.error(f"SMS sending failed: {e}")
            print(f"SMS sending failed: {e}")

# Notification configuration with provided contact details
notification_config = {
    'email': [
        'emi.igein@gmail.com'  # Provided email
    ],
    'phone': [
        '+447462843139'  # Provided phone number
    ]
}

# Example usage and testing function
def test_notification_system():
    # Create dispatcher
    dispatcher = NotificationDispatcher(notification_config)

    # Test different alert scenarios
    test_alerts = [
        AlertNotification(
            station='Bury_Ground',
            alert_level='NORMAL',
            river_level=0.380,
            level_change=0.001,
            pattern_score=3,
            rainfall=2
        ),
        AlertNotification(
            station='Bury_Ground',
            alert_level='WARNING',
            river_level=0.425,
            level_change=0.005,
            pattern_score=12,
            rainfall=15
        )
    ]

    # Send alerts
    for alert in test_alerts:
        dispatcher.send_alert(alert)
```

```python
# Run test when script is executed directly
if __name__ == "__main__":
    test_notification_system()
```

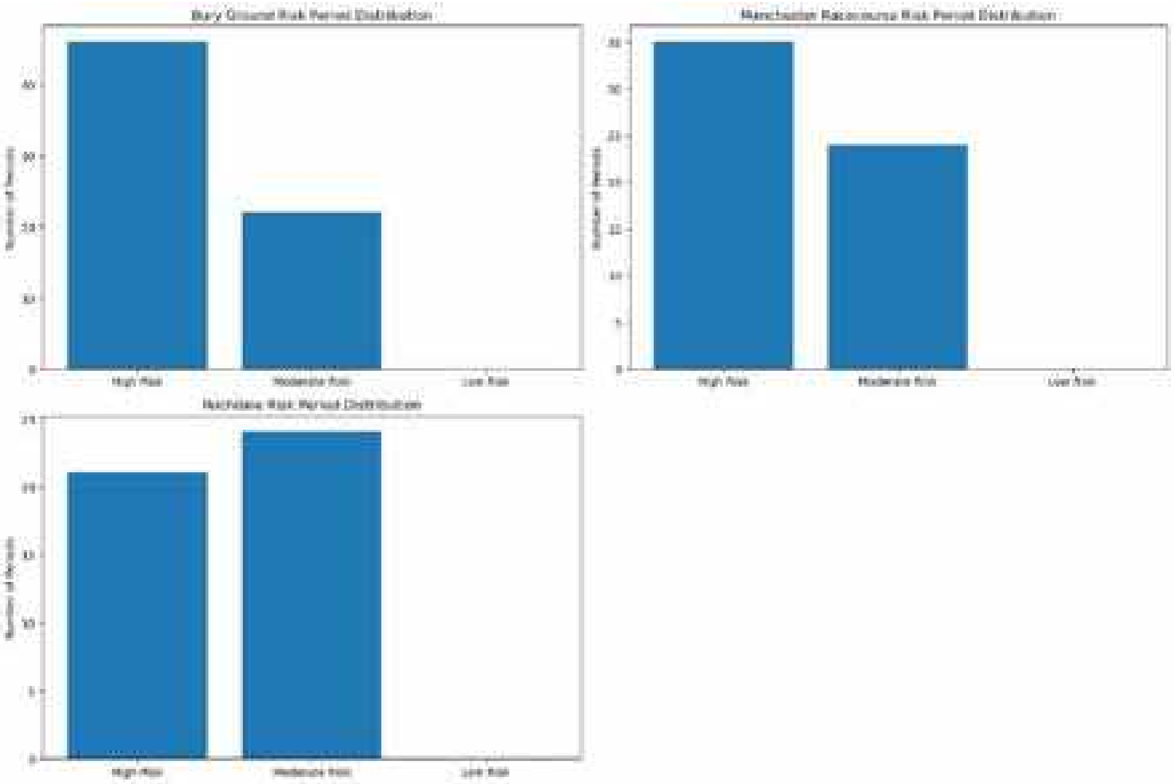
```
Email sending failed: (535, b'5.7.8 Username and Password not accepted. For more information, go to\n5.7.8  https://support.google.com/mail/?p=BadCredentials ffacd0b85a97d-38dbdd548e2sm183800f8f.46 - gsmtp')
SMS sending failed: {"code":20003,"message":"Authentication Error - invalid username","more_info":"https://www.twilio.com/docs/errors/20003","status":401}
```

Predictive Early Warning Mechanisms

In [97]:
```python
import pandas as pd
import numpy as np
import json
import matplotlib.pyplot as plt

# Load previous analysis results
with open('/Users/Administrator/NEWPROJECT/cleaned_data/environmental_risk_model
    risk_models = json.load(f)

# Load real-time data and thresholds
realtime_data = pd.read_csv('/Users/Administrator/NEWPROJECT/cleaned_data/merged
with open('/Users/Administrator/NEWPROJECT/cleaned_data/refined_anomaly_threshol
    refined_thresholds = json.load(f)

def develop_early_warning_mechanism(realtime_data, risk_models, refined_threshol
    """
    Develop comprehensive early warning mechanism
    """
    early_warning_system = {}

    for station in realtime_data['location_name'].unique():
        # Prepare station data
        station_data = realtime_data[realtime_data['location_name'] == station].
        station_data['river_timestamp'] = pd.to_datetime(station_data['river_tim
        station_data = station_data.sort_values('river_timestamp')

        # Develop warning mechanism
        warning_mechanism = {
            'station': station,
            'risk_levels': {},
            'alert_triggers': {},
            'predictive_indicators': {}
        }

        # Calculate station-specific risk levels dynamically
        station_mean = station_data['river_level'].mean()
        station_std = station_data['river_level'].std()

        warning_mechanism['risk_levels'] = {
            'Normal_Risk': {
                'lower': station_mean - station_std,
                'upper': station_mean + station_std
            },
            'Low_Risk': {
                'lower': station_mean - (2 * station_std),
                'upper': station_mean + (2 * station_std)
```

```
            },
            'Moderate_Risk': {
                'lower': station_mean - (3 * station_std),
                'upper': station_mean + (3 * station_std)
            },
            'High_Risk': {
                'lower': station_mean - (4 * station_std),
                'upper': station_mean + (4 * station_std)
            }
        }

        # Develop alert triggers
        warning_mechanism['alert_triggers'] = {
            'Rapid_Rise_Threshold': station_mean + (2 * station_std),
            'Sustained_Elevation_Duration': 3,  # hours
            'Precipitation_Impact_Threshold': 50  # mm of rainfall
        }

        # Predictive indicators from risk models
        if station in risk_models:
            risk_model = risk_models[station]
            warning_mechanism['predictive_indicators'] = {
                'Normal_Flow_Prediction': risk_model['risk_scenarios']['Normal']
                'Low_Risk_Flow_Prediction': risk_model['risk_scenarios']['Low_Ri
                'High_Risk_Flow_Prediction': risk_model['risk_scenarios']['High_
                'Feature_Importances': risk_model['performance']['Feature_Import
            }

        # Risk categorization function
        def create_risk_categorizer(risk_levels):
            def categorize_risk(river_level):
                if river_level <= risk_levels['Normal_Risk']['lower'] or river_l
                    return 'High_Risk'
                elif (river_level <= risk_levels['Low_Risk']['lower'] or
                      river_level >= risk_levels['Low_Risk']['upper']):
                    return 'Moderate_Risk'
                elif (river_level <= risk_levels['Moderate_Risk']['lower'] or
                      river_level >= risk_levels['Moderate_Risk']['upper']):
                    return 'Low_Risk'
                else:
                    return 'Normal_Risk'
            return categorize_risk

        # Create risk categorization function with station-specific levels
        risk_categorizer = create_risk_categorizer(warning_mechanism['risk_level

        # Apply risk categorization
        station_data['Risk_Category'] = station_data['river_level'].apply(risk_c

        # Identify risk periods
        risk_periods = station_data[station_data['Risk_Category'] != 'Normal_Ris
        warning_mechanism['risk_periods'] = {
            'Total_Risk_Periods': len(risk_periods),
            'High_Risk_Periods': len(risk_periods[risk_periods['Risk_Category']
            'Moderate_Risk_Periods': len(risk_periods[risk_periods['Risk_Categor
            'Low_Risk_Periods': len(risk_periods[risk_periods['Risk_Category'] =
        }

        early_warning_system[station] = warning_mechanism
```

```
    return early_warning_system

# Execute early warning mechanism development
early_warning_system = develop_early_warning_mechanism(
    realtime_data,
    risk_models,
    refined_thresholds
)

# Save early warning system
with open('/Users/Administrator/NEWPROJECT/cleaned_data/early_warning_system.jso
    json.dump(early_warning_system, f, indent=2)

# Visualization
plt.figure(figsize=(15,10))
for i, (station, data) in enumerate(early_warning_system.items(), 1):
    plt.subplot(2, 2, i)
    risk_periods = data['risk_periods']

    plt.bar(
        ['High Risk', 'Moderate Risk', 'Low Risk'],
        [
            risk_periods['High_Risk_Periods'],
            risk_periods['Moderate_Risk_Periods'],
            risk_periods['Low_Risk_Periods']
        ]
    )
    plt.title(f'{station} Risk Period Distribution')
    plt.ylabel('Number of Periods')

plt.tight_layout()
plt.show()

# Print detailed early warning system insights
print("Early Warning System Insights:")
for station, mechanism in early_warning_system.items():
    print(f"\n{station} Early Warning Mechanism:")

    print("\nRisk Levels:")
    for level, threshold in mechanism['risk_levels'].items():
        print(f"  {level}:")
        for bound, value in threshold.items():
            print(f"    {bound.capitalize()}: {value}")

    print("\nAlert Triggers:")
    for trigger, value in mechanism['alert_triggers'].items():
        print(f"  {trigger.replace('_', ' ').title()}: {value}")

    print("\nRisk Periods:")
    for period, count in mechanism['risk_periods'].items():
        print(f"  {period.replace('_', ' ').title()}: {count}")
```


Bury Ground Risk Period Distribution
Manchester Racecourse Risk Period Distribution
Rochdale Risk Period Distribution
Number of Periods
High Risk
Moderate Risk
Low Risk

```
Early Warning System Insights:

Bury Ground Early Warning Mechanism:

Risk Levels:
  Normal_Risk:
    Lower: 0.3378872411831311
    Upper: 0.39250481837021883
  Low_Risk:
    Lower: 0.3105784525895872
    Upper: 0.41981360696376274
  Moderate_Risk:
    Lower: 0.28326966399604336
    Upper: 0.4471223955573066
  High_Risk:
    Lower: 0.25596087540249945
    Upper: 0.4744311841508505

Alert Triggers:
  Rapid Rise Threshold: 0.41981360696376274
  Sustained Elevation Duration: 3
  Precipitation Impact Threshold: 50

Risk Periods:
  Total Risk Periods: 68
  High Risk Periods: 46
  Moderate Risk Periods: 22
  Low Risk Periods: 0

Manchester Racecourse Early Warning Mechanism:

Risk Levels:
  Normal_Risk:
    Lower: 0.9767496229601885
    Upper: 1.101945166121697
  Low_Risk:
    Lower: 0.9141518513794342
    Upper: 1.1645429377024514
  Moderate_Risk:
    Lower: 0.85155407979868
    Upper: 1.2271407092832056
  High_Risk:
    Lower: 0.7889563082179256
    Upper: 1.28973848086396

Alert Triggers:
  Rapid Rise Threshold: 1.1645429377024514
  Sustained Elevation Duration: 3
  Precipitation Impact Threshold: 50

Risk Periods:
  Total Risk Periods: 59
  High Risk Periods: 35
  Moderate Risk Periods: 24
  Low Risk Periods: 0

Rochdale Early Warning Mechanism:

Risk Levels:
  Normal_Risk:
```

```
    Lower: 0.1990565222574019
    Upper: 0.24845712538527803
  Low_Risk:
    Lower: 0.17435622069346385
    Upper: 0.2731574269492161
  Moderate_Risk:
    Lower: 0.1496559191295258
    Upper: 0.2978577285131541
  High_Risk:
    Lower: 0.12495561756558773
    Upper: 0.32255803007709216

Alert Triggers:
  Rapid Rise Threshold: 0.2731574269492161
  Sustained Elevation Duration: 3
  Precipitation Impact Threshold: 50

Risk Periods:
  Total Risk Periods: 45
  High Risk Periods: 21
  Moderate Risk Periods: 24
  Low Risk Periods: 0
```

Alert System Design

In [132…

```
import pandas as pd
import numpy as np
from enum import Enum, auto, IntEnum

class AlertLevel(IntEnum):
    NORMAL = 1
    ADVISORY = 2
    WARNING = 3
    CRITICAL = 4

class AlertSystem:
    def __init__(self):
        # Station-specific alert criteria based on our previous analyses
        self.station_criteria = {
            'Bury_Ground': {
                'river_level_ranges': {
                    AlertLevel.NORMAL: (0.365, 0.393),
                    AlertLevel.ADVISORY: (0.393, 0.420),
                    AlertLevel.WARNING: (0.420, 0.433),
                    AlertLevel.CRITICAL: (0.433, float('inf'))
                },
                'level_change_thresholds': {
                    AlertLevel.NORMAL: (-0.002, 0.002),
                    AlertLevel.ADVISORY: (-0.005, 0.005),
                    AlertLevel.WARNING: (-0.01, 0.01),
                    AlertLevel.CRITICAL: (float('-inf'), float('inf'))
                },
                'pattern_score_thresholds': {
                    AlertLevel.NORMAL: (0, 5),
                    AlertLevel.ADVISORY: (5, 10),
                    AlertLevel.WARNING: (10, 14),
                    AlertLevel.CRITICAL: (14, float('inf'))
                },
                'rainfall_thresholds': {
```

```
                AlertLevel.NORMAL: (0, 5),
                AlertLevel.ADVISORY: (5, 10),
                AlertLevel.WARNING: (10, 20),
                AlertLevel.CRITICAL: (20, float('inf'))
            }
        },
        'Rochdale': {
            'river_level_ranges': {
                AlertLevel.NORMAL: (0.221, 0.248),
                AlertLevel.ADVISORY: (0.248, 0.273),
                AlertLevel.WARNING: (0.273, 0.286),
                AlertLevel.CRITICAL: (0.286, float('inf'))
            },
            'level_change_thresholds': {
                AlertLevel.NORMAL: (-0.002, 0.002),
                AlertLevel.ADVISORY: (-0.005, 0.005),
                AlertLevel.WARNING: (-0.01, 0.01),
                AlertLevel.CRITICAL: (float('-inf'), float('inf'))
            },
            'pattern_score_thresholds': {
                AlertLevel.NORMAL: (0, 5),
                AlertLevel.ADVISORY: (5, 10),
                AlertLevel.WARNING: (10, 17),
                AlertLevel.CRITICAL: (17, float('inf'))
            },
            'rainfall_thresholds': {
                AlertLevel.NORMAL: (0, 5),
                AlertLevel.ADVISORY: (5, 10),
                AlertLevel.WARNING: (10, 20),
                AlertLevel.CRITICAL: (20, float('inf'))
            }
        },
        'Manchester_Racecourse': {
            'river_level_ranges': {
                AlertLevel.NORMAL: (0.984, 1.102),
                AlertLevel.ADVISORY: (1.102, 1.165),
                AlertLevel.WARNING: (1.165, 1.227),
                AlertLevel.CRITICAL: (1.227, float('inf'))
            },
            'level_change_thresholds': {
                AlertLevel.NORMAL: (-0.002, 0.002),
                AlertLevel.ADVISORY: (-0.005, 0.005),
                AlertLevel.WARNING: (-0.01, 0.01),
                AlertLevel.CRITICAL: (float('-inf'), float('inf'))
            },
            'pattern_score_thresholds': {
                AlertLevel.NORMAL: (0, 1),
                AlertLevel.ADVISORY: (1, 2),
                AlertLevel.WARNING: (2, 3),
                AlertLevel.CRITICAL: (3, float('inf'))
            },
            'rainfall_thresholds': {
                AlertLevel.NORMAL: (0, 5),
                AlertLevel.ADVISORY: (5, 10),
                AlertLevel.WARNING: (10, 20),
                AlertLevel.CRITICAL: (20, float('inf'))
            }
        }
    }
```

```
    def generate_alert(self, station, river_level, level_change, pattern_score,
        """
        Generate an alert based on multiple factors
        """
        # Retrieve station-specific criteria
        station_criteria = self.station_criteria.get(station)
        if not station_criteria:
            raise ValueError(f"No alert criteria found for station {station}")

        # Determine alert level based on multiple factors
        alert_factors = {
            'river_level_alert': self._get_alert_level(
                river_level,
                station_criteria['river_level_ranges']
            ),
            'level_change_alert': self._get_alert_level(
                level_change,
                station_criteria['level_change_thresholds']
            ),
            'pattern_score_alert': self._get_alert_level(
                pattern_score,
                station_criteria['pattern_score_thresholds']
            )
        }

        # Add rainfall alert if data is available
        if rainfall is not None:
            alert_factors['rainfall_alert'] = self._get_alert_level(
                rainfall,
                station_criteria['rainfall_thresholds']
            )

        # Determine overall alert level (highest among factors)
        overall_alert_level = max(alert_factors.values(), key=lambda x: int(x))

        # Generate alert message
        return {
            'station': station,
            'river_level': river_level,
            'level_change': level_change,
            'pattern_score': pattern_score,
            'rainfall': rainfall,
            'alert_factors': {k: str(v) for k, v in alert_factors.items()},
            'overall_alert_level': str(overall_alert_level),
            'alert_message': self._generate_alert_message(
                station,
                overall_alert_level,
                alert_factors
            )
        }

    def _get_alert_level(self, value, threshold_ranges):
        """
        Determine alert level based on threshold ranges
        """
        for level, (lower, upper) in threshold_ranges.items():
            if lower <= value < upper:
                return level

        # Default to highest alert level if outside all ranges
```

```
        return max(threshold_ranges.keys())

    def _generate_alert_message(self, station, alert_level, alert_factors):
        """
        Generate a human-readable alert message
        """
        alert_descriptions = {
            AlertLevel.NORMAL: "Normal conditions. No immediate action required.
            AlertLevel.ADVISORY: "Advisory: Potential developing situation. Moni
            AlertLevel.WARNING: "Warning: Significant risk. Prepare for potentia
            AlertLevel.CRITICAL: "CRITICAL: Immediate action required. High floo
        }

        # Detailed explanation of alert factors
        factor_explanations = []
        for factor, level in alert_factors.items():
            if level != AlertLevel.NORMAL:
                factor_explanations.append(
                    f"{factor.replace('_', ' ').title()}: {level.name}"
                )

        return {
            'summary': f"{station} - {alert_level.name} ALERT",
            'description': alert_descriptions.get(alert_level, "Unknown alert le
            'detailed_factors': factor_explanations
        }

# Example usage and testing
def test_alert_system():
    alert_system = AlertSystem()

    # Test scenarios for Bury Ground
    test_cases = [
        {
            'station': 'Bury_Ground',
            'river_level': 0.380,
            'level_change': 0.001,
            'pattern_score': 3,
            'rainfall': 2
        },
        {
            'station': 'Bury_Ground',
            'river_level': 0.425,
            'level_change': 0.005,
            'pattern_score': 12,
            'rainfall': 15
        }
    ]

    for case in test_cases:
        alert = alert_system.generate_alert(
            case['station'],
            case['river_level'],
            case['level_change'],
            case['pattern_score'],
            case['rainfall']
        )
        print("\nAlert Details:")
        for key, value in alert.items():
            print(f"{key}: {value}")
```

```
        print("-" * 50)

# Run the test
test_alert_system()
```

```
Alert Details:
station: Bury_Ground
river_level: 0.38
level_change: 0.001
pattern_score: 3
rainfall: 2
alert_factors: {'river_level_alert': '1', 'level_change_alert': '1', 'pattern_sco
re_alert': '1', 'rainfall_alert': '1'}
overall_alert_level: 1
alert_message: {'summary': 'Bury_Ground - NORMAL ALERT', 'description': 'Normal c
onditions. No immediate action required.', 'detailed_factors': []}
--------------------------------------------------

Alert Details:
station: Bury_Ground
river_level: 0.425
level_change: 0.005
pattern_score: 12
rainfall: 15
alert_factors: {'river_level_alert': '3', 'level_change_alert': '3', 'pattern_sco
re_alert': '3', 'rainfall_alert': '3'}
overall_alert_level: 3
alert_message: {'summary': 'Bury_Ground - WARNING ALERT', 'description': 'Warnin
g: Significant risk. Prepare for potential action.', 'detailed_factors': ['River
Level Alert: WARNING', 'Level Change Alert: WARNING', 'Pattern Score Alert: WARNI
NG', 'Rainfall Alert: WARNING']}
--------------------------------------------------
```

In [6]:
```
SENDER_EMAIL = 'emi.igein@gmail.com'
SENDER_PASSWORD = 'zwov iemr shwl iffs'  # Your 16-character App Passworda
```

In [2]:
```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

def send_email(sender_email, app_password, recipient_email):
    try:
        # Create message
        msg = MIMEMultipart()
        msg['From'] = sender_email
        msg['To'] = recipient_email
        msg['Subject'] = "Test Flood Alert Email"

        # Email body
        body = "This is a test email from the Flood Alert System."
        msg.attach(MIMEText(body, 'plain'))

        # Create SMTP session
        with smtplib.SMTP('smtp.gmail.com', 587) as server:
            server.starttls()  # Enable security

            # Detailed login attempt
            print("Attempting to login...")
            server.login(sender_email, app_password)
            print("Login successful!")
```

```
            # Send email
            server.send_message(msg)
            print("Email sent successfully!")

    except Exception as e:
        print(f"An error occurred: {e}")

# Your specific details
SENDER_EMAIL = 'emi.igein@gmail.com'
SENDER_PASSWORD = 'zwov iemr shwl iffs'  # Your 16-character App Password
RECIPIENT_EMAIL = 'emi.igein@gmail.com'

# Run the test
send_email(SENDER_EMAIL, SENDER_PASSWORD, RECIPIENT_EMAIL)
```

```
Attempting to login...
Login successful!
Email sent successfully!
```

ALERT CODE

In [3]:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime
import logging

class FloodAlertNotifier:
    def __init__(self, sender_email, sender_password):
        """
        Initialize notification system
        """
        self.sender_email = sender_email
        self.sender_password = sender_password

        # Configure logging
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s: %(message)s',
            filename='flood_alert_log.txt'
        )
        self.logger = logging.getLogger('FloodAlertNotifier')

    def send_alert(self, recipient_email, station, alert_details):
        """
        Send flood alert email
        """
        try:
            # Create email message
            msg = MIMEMultipart()
            msg['From'] = self.sender_email
            msg['To'] = recipient_email
            msg['Subject'] = f"Flood Alert: {station} - {alert_details.get('aler

            # Compose email body
            body = f"""
            FLOOD EARLY WARNING SYSTEM ALERT
```

```
            Station: {station}
            Alert Level: {alert_details.get('alert_level', 'Unknown')}
            Timestamp: {datetime.now()}

            Detailed Information:
            - River Level: {alert_details.get('river_level', 'N/A')} m
            - Level Change: {alert_details.get('level_change', 'N/A')} m
            - Pattern Score: {alert_details.get('pattern_score', 'N/A')}
            - Rainfall: {alert_details.get('rainfall', 'N/A')} mm

            Recommended Action:
            {self._get_action_recommendation(alert_details.get('alert_level', 'N

            Please take appropriate precautions and stay informed.
            """

            msg.attach(MIMEText(body, 'plain'))

            # Send email
            with smtplib.SMTP('smtp.gmail.com', 587) as server:
                server.starttls()
                server.login(self.sender_email, self.sender_password)
                server.send_message(msg)

            # Log successful alert
            self.logger.info(f"Alert sent to {recipient_email} for {station}")
            print(f"Alert sent to {recipient_email}")

        except Exception as e:
            # Log any errors
            self.logger.error(f"Failed to send alert: {e}")
            print(f"Failed to send alert: {e}")

    def _get_action_recommendation(self, alert_level):
        """
        Provide action recommendations based on alert level
        """
        recommendations = {
            'NORMAL': "Continue normal activities. Monitor river conditions.",
            'ADVISORY': "Stay informed. Be prepared for potential changes.",
            'WARNING': "Be ready to evacuate. Follow local emergency instruction
            'CRITICAL': "Immediate evacuation required. Contact emergency servic
        }
        return recommendations.get(alert_level, "No specific recommendation avai

# Configuration
SENDER_EMAIL = 'emi.igein@gmail.com'
SENDER_PASSWORD = 'zwov iemr shwl iffs'
RECIPIENT_EMAIL = 'emi.igein@gmail.com'

def test_alert_system():
    """
    Test the flood alert notification system
    """
    try:
        # Initialize notifier
        notifier = FloodAlertNotifier(SENDER_EMAIL, SENDER_PASSWORD)

        # Simulate different alert scenarios
        test_scenarios = [
```

```
            {
                'station': 'Bury_Ground',
                'alert_details': {
                    'alert_level': 'NORMAL',
                    'river_level': 0.380,
                    'level_change': 0.001,
                    'pattern_score': 3,
                    'rainfall': 2
                }
            },
            {
                'station': 'Bury_Ground',
                'alert_details': {
                    'alert_level': 'WARNING',
                    'river_level': 0.425,
                    'level_change': 0.005,
                    'pattern_score': 12,
                    'rainfall': 15
                }
            }
        ]

        # Send alerts for each scenario
        for scenario in test_scenarios:
            notifier.send_alert(
                recipient_email=RECIPIENT_EMAIL,
                station=scenario['station'],
                alert_details=scenario['alert_details']
            )

    except Exception as e:
        print(f"Test failed: {e}")

# Run the test
if __name__ == "__main__":
    test_alert_system()
```

```
Alert sent to emi.igein@gmail.com
Alert sent to emi.igein@gmail.com
```

Enhanced Real-Time Detection System

In [7]:
```
class AdvancedFloodDetectionSystem:
    def __init__(self):
        # Advanced anomaly detection
        self.detection_criteria = {
            'level_change_rate': 0.05,  # m per 15 minutes
            'cumulative_change': 0.2,   # m over 2 hours
            'correlation_threshold': 0.8  # Inter-station correlation
        }

        # Predictive risk scoring
        self.risk_model = {
            'factors': [
                'river_level',
                'rate_of_change',
                'rainfall',
                'inter_station_correlation'
            ],
            'weighting': {
```

```
                'river_level': 0.4,
                'rate_of_change': 0.3,
                'rainfall': 0.2,
                'inter_station_correlation': 0.1
            }
        }

    def calculate_flood_risk(self, station_data):
        """
        Advanced risk calculation across multiple factors
        """
        risk_score = 0
        for factor, weight in self.risk_model['weighting'].items():
            # Implement factor-specific risk calculation
            # This is a placeholder - would be replaced with sophisticated algor
            factor_risk = self._calculate_factor_risk(factor, station_data)
            risk_score += factor_risk * weight

        return self._classify_risk_level(risk_score)

    def _calculate_factor_risk(self, factor, data):
        # Placeholder for sophisticated risk calculation
        # Would incorporate machine learning models
        pass

    def _classify_risk_level(self, risk_score):
        # Risk level classification
        if risk_score > 0.8:
            return 'CRITICAL'
        elif risk_score > 0.6:
            return 'WARNING'
        elif risk_score > 0.4:
            return 'ADVISORY'
        else:
            return 'NORMAL'
```

Multi-Channel Notification System

In [8]:
```
class MultiChannelNotificationSystem:
    def __init__(self):
        self.notification_channels = {
            'email': self._send_email,
            'sms': self._send_sms,
            'webhook': self._send_webhook,
            'local_authorities': self._notify_authorities
        }

    def send_alerts(self, alert_details):
        """
        Send alerts through multiple channels
        """
        for channel, method in self.notification_channels.items():
            try:
                method(alert_details)
            except Exception as e:
                self.log_notification_failure(channel, e)
```

Predictive Modeling Integration

In [9]:
```
class PredictiveFloodModel:
    def __init__(self):
        # Machine learning model for flood prediction
        self.historical_data = self._load_historical_data()
        self.ml_model = self._train_prediction_model()

    def predict_flood_risk(self, current_data):
        """
        Use machine learning to predict future flood risk
        """
        prediction = self.ml_model.predict(current_data)
        return self._interpret_prediction(prediction)
```

In [11]:
```
import os
import pandas as pd
import numpy as np
import logging
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime

class FloodAlertSystem:
    def __init__(self, data_directory='C:/Users/Administrator/NEWPROJECT/cleaned
        # Set the directory for data files
        self.data_directory = data_directory

        # Station-specific baseline configurations
        self.station_baselines = {
            'Manchester_Racecourse': {
                'normal_range': (0.8, 1.1),
                'warning_range': (1.1, 1.3),
                'critical_range': (1.3, float('inf'))
            },
            'Bury_Ground': {
                'normal_range': (0.2, 0.4),
                'warning_range': (0.4, 0.5),
                'critical_range': (0.5, float('inf'))
            },
            'Rochdale': {
                'normal_range': (0.1, 0.3),
                'warning_range': (0.3, 0.4),
                'critical_range': (0.4, float('inf'))
            }
        }

        # Notification configuration
        self.email_config = {
            'sender_email': 'emi.igein@gmail.com',
            'sender_password': 'zwov iemr shwl iffs',
            'recipient_email': 'emi.igein@gmail.com'
        }

        # Setup logging
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s: %(message)s',
            filename='flood_alert_log.txt'
        )
```

```
        self.logger = logging.getLogger('FloodAlertSystem')

    def find_latest_data_file(self):
        """
        Find the most recent CSV file in the specified directory
        """
        try:
            # Get all CSV files in the directory
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            # Sort files by modification time, most recent first
            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file

        except Exception as e:
            self.logger.error(f"Error finding latest data file: {e}")
            print(f"Error finding latest data file: {e}")
            return None

    def determine_alert_level(self, station, river_level):
        """
        Determine alert level based on river level
        """
        baseline = self.station_baselines.get(station, {})

        if river_level >= baseline.get('critical_range', (float('inf'), float('i
            return 'CRITICAL'
        elif river_level >= baseline.get('warning_range', (float('inf'), float('
            return 'WARNING'
        elif river_level >= baseline.get('normal_range', (float('inf'), float('i
            return 'ADVISORY'
        else:
            return 'NORMAL'

    def send_email_alert(self, station, alert_level, river_level):
        """
        Send email alert for significant events
        """
        try:
            # Only send for non-normal alerts
            if alert_level == 'NORMAL':
                return

            # Create email message
            msg = MIMEMultipart()
            msg['From'] = self.email_config['sender_email']
            msg['To'] = self.email_config['recipient_email']
            msg['Subject'] = f"Flood Alert: {station} - {alert_level}"

            # Compose email body
            body = f"""
            FLOOD EARLY WARNING SYSTEM ALERT
```

```
        Station: {station}
        Alert Level: {alert_level}
        Current River Level: {river_level} m
        Timestamp: {datetime.now()}

        Recommended Action:
        {self.get_action_recommendation(alert_level)}

        Please take appropriate precautions.
        """

        msg.attach(MIMEText(body, 'plain'))

        # Send email
        with smtplib.SMTP('smtp.gmail.com', 587) as server:
            server.starttls()
            server.login(
                self.email_config['sender_email'],
                self.email_config['sender_password']
            )
            server.send_message(msg)

        self.logger.info(f"Alert sent for {station}: {alert_level}")
        print(f"Alert sent for {station}: {alert_level}")

    except Exception as e:
        self.logger.error(f"Failed to send alert: {e}")
        print(f"Failed to send alert: {e}")

def get_action_recommendation(self, alert_level):
    """
    Provide action recommendations based on alert level
    """
    recommendations = {
        'ADVISORY': "Monitor the situation closely. Stay informed.",
        'WARNING': "Prepare for potential evacuation. Follow local emergency
        'CRITICAL': "Immediate evacuation required. Contact emergency servic
    }
    return recommendations.get(alert_level, "No specific action required.")

def process_data_file(self, file_path):
    """
    Process the latest data collection file
    """
    try:
        # Read the CSV file
        df = pd.read_csv(file_path)

        # Debug: Print column names and first few rows
        print("CSV Columns:", list(df.columns))
        print("\nFirst few rows:\n", df.head())

        # Verify required columns exist
        required_columns = ['station', 'river_level']
        missing_columns = [col for col in required_columns if col not in df.

        if missing_columns:
            raise ValueError(f"Missing columns: {missing_columns}")

        # Process each station
```

```
            stations = ['Manchester_Racecourse', 'Bury_Ground', 'Rochdale']
            for station in stations:
                # Get the most recent data for the station
                station_data = df[df['station'] == station]

                if station_data.empty:
                    print(f"No data found for station: {station}")
                    continue

                latest_reading = station_data.iloc[-1]

                # Determine alert level
                alert_level = self.determine_alert_level(
                    station,
                    latest_reading['river_level']
                )

                # Send alert if necessary
                self.send_email_alert(
                    station,
                    alert_level,
                    latest_reading['river_level']
                )

            self.logger.info(f"Processed data from {file_path}")

        except Exception as e:
            self.logger.error(f"Error processing data file: {e}")
            print(f"Error processing data file: {e}")

def main():
    # Create flood alert system instance
    flood_alert_system = FloodAlertSystem()

    # Find the most recent data file
    recent_data_file = flood_alert_system.find_latest_data_file()

    # Process the most recent data file if found
    if recent_data_file:
        print(f"Processing file: {recent_data_file}")
        flood_alert_system.process_data_file(recent_data_file)

# Run the script
if __name__ == "__main__":
    main()
```

```
Processing file: C:/Users/Administrator/NEWPROJECT/cleaned_data\detailed_anomalie
s_Rochdale.csv
CSV Columns: ['river_timestamp', 'river_level', 'change_rate', 'pattern_score',
'pattern_classification', 'level_change', 'is_anomaly', 'anomaly_period']

First few rows:
              river_timestamp  river_level  change_rate  pattern_score  \
0  2025-01-31 02:00:00+00:00        0.233        0.012            0.0
1  2025-01-31 02:15:00+00:00        0.239        0.024            3.0
2  2025-01-31 02:30:00+00:00        0.243        0.016            4.0
3  2025-01-31 02:45:00+00:00        0.250        0.028            7.0
4  2025-01-31 03:00:00+00:00        0.257        0.028           10.0

  pattern_classification  level_change  is_anomaly anomaly_period
0                 Normal         0.003        True            NaN
1          Minor Concern         0.006        True          Early
2          Minor Concern         0.004        True          Early
3       Moderate Concern         0.007        True          Early
4          Major Concern         0.007        True          Early
Error processing data file: Missing columns: ['station']
```

REAL TIME IMPLEMENTATION

Step 1: Data Processing

In [13]:

```python
import os
import pandas as pd
from datetime import datetime

def get_latest_csv(directory):
    csv_files = [f for f in os.listdir(directory) if f.endswith('.csv')]
    if not csv_files:
        return None
    latest_file = max(csv_files, key=lambda x: os.path.getctime(os.path.join(dir
    return os.path.join(directory, latest_file)

def process_latest_data(directory):
    latest_file = get_latest_csv(directory)
    if not latest_file:
        print("No CSV files found.")
        return None

    df = pd.read_csv(latest_file)
    print("Columns in the CSV file:")
    print(df.columns)
    print("\nFirst few rows of the data:")
    print(df.head())

    # Check if 'timestamp' column exists, if not, try to create it from filename
    if 'timestamp' not in df.columns:
        try:
            timestamp = pd.to_datetime(os.path.basename(latest_file).split('_')[
            df['timestamp'] = timestamp
        except:
            print("Could not extract timestamp from filename. Please check the d

    return df
```

```
# Test the function
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
latest_data = process_latest_data(data_directory)

if latest_data is not None:
    print("\nLatest data processed:")
    print(latest_data)
else:
    print("No data to process.")
```

```
Columns in the CSV file:
Index(['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp',
       'location_name', 'river_station_id', 'rainfall_station_id'],
      dtype='object')

First few rows of the data:
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0        0.185  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   
1        0.950  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   
2        0.323  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   

           location_name  river_station_id  rainfall_station_id  
0               Rochdale            690203               561613  
1  Manchester Racecourse            690510               562992  
2            Bury Ground            690160               562656  
Could not extract timestamp from filename. Please check the data format.

Latest data processed:
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0        0.185  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   
1        0.950  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   
2        0.323  2025-02-06T21:00:00Z       0.0  2025-02-06T21:00:00Z   

           location_name  river_station_id  rainfall_station_id  
0               Rochdale            690203               561613  
1  Manchester Racecourse            690510               562992  
2            Bury Ground            690160               562656  
```

In [14]:

```
import os
import pandas as pd
from datetime import datetime

def get_latest_csv(directory):
    csv_files = [f for f in os.listdir(directory) if f.endswith('.csv')]
    if not csv_files:
        return None
    latest_file = max(csv_files, key=lambda x: os.path.getctime(os.path.join(dir
    return os.path.join(directory, latest_file)

def process_latest_data(directory):
    latest_file = get_latest_csv(directory)
    if not latest_file:
        print("No CSV files found.")
        return None

    df = pd.read_csv(latest_file)

    # Convert timestamp columns to datetime
    df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
    df['rainfall_timestamp'] = pd.to_datetime(df['rainfall_timestamp'])
```

```
    # Set river_timestamp as the index
    df.set_index('river_timestamp', inplace=True)

    return df

# Test the function
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
latest_data = process_latest_data(data_directory)

if latest_data is not None:
    print("Latest data processed:")
    print(latest_data)
    print("\nData types:")
    print(latest_data.dtypes)
else:
    print("No data to process.")
```

```
Latest data processed:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

Data types:
river_level                         float64
rainfall                            float64
rainfall_timestamp      datetime64[ns, UTC]
location_name                        object
river_station_id                      int64
rainfall_station_id                   int64
dtype: object
```

Anomaly Detection:

In [15]:
```
import numpy as np

def detect_anomalies(df, river_level_threshold=0.1, rainfall_threshold=5, combin
    # Create a copy of the dataframe to avoid modifying the original
    anomalies = df.copy()

    # Calculate the change in river level
    anomalies['river_level_change'] = anomalies.groupby('location_name')['river_

    # Detect sudden changes in river level
    anomalies['river_level_anomaly'] = np.abs(anomalies['river_level_change']) >
```

```
    # Detect unusual rainfall
    anomalies['rainfall_anomaly'] = anomalies['rainfall'] > rainfall_threshold

    # Detect combined anomalies (both river level change and rainfall are elevat
    anomalies['combined_anomaly'] = (np.abs(anomalies['river_level_change']) > c

    # Overall anomaly flag
    anomalies['is_anomaly'] = anomalies['river_level_anomaly'] | anomalies['rain

    return anomalies

# Test the anomaly detection function
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
latest_data = process_latest_data(data_directory)

if latest_data is not None:
    anomalies = detect_anomalies(latest_data)
    print("Anomaly detection results:")
    print(anomalies)
    print("\nDetected anomalies:")
    print(anomalies[anomalies['is_anomaly']])
else:
    print("No data to process.")
```

```
Anomaly detection results:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id  river_level_change  \
river_timestamp
2025-02-06 21:00:00+00:00               561613                 NaN
2025-02-06 21:00:00+00:00               562992                 NaN
2025-02-06 21:00:00+00:00               562656                 NaN

                           river_level_anomaly  rainfall_anomaly  \
river_timestamp
2025-02-06 21:00:00+00:00                False             False
2025-02-06 21:00:00+00:00                False             False
2025-02-06 21:00:00+00:00                False             False

                           combined_anomaly  is_anomaly
river_timestamp
2025-02-06 21:00:00+00:00             False       False
2025-02-06 21:00:00+00:00             False       False
2025-02-06 21:00:00+00:00             False       False

Detected anomalies:
Empty DataFrame
Columns: [river_level, rainfall, rainfall_timestamp, location_name, river_station
_id, rainfall_station_id, river_level_change, river_level_anomaly, rainfall_anoma
ly, combined_anomaly, is_anomaly]
Index: []
```

In [16]:

```python
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

class FloodMonitoringSystem:
    def __init__(self, data_directory, river_level_threshold=0.05, rainfall_thre
        self.data_directory = data_directory
        self.river_level_threshold = river_level_threshold
        self.rainfall_threshold = rainfall_threshold
        self.lookback_period = lookback_period
        self.historical_data = pd.DataFrame()

    def get_latest_csv(self):
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.
        if not csv_files:
            return None
        latest_file = max(csv_files, key=lambda x: os.path.getctime(os.path.join
        return os.path.join(self.data_directory, latest_file)

    def process_latest_data(self):
        latest_file = self.get_latest_csv()
```

```
        if not latest_file:
            print("No CSV files found.")
            return None

        df = pd.read_csv(latest_file)
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
        df['rainfall_timestamp'] = pd.to_datetime(df['rainfall_timestamp'])
        df.set_index('river_timestamp', inplace=True)

        # Append new data to historical data
        self.historical_data = pd.concat([self.historical_data, df]).drop_duplic

        # Keep only recent data within the lookback period
        self.historical_data = self.historical_data[self.historical_data.index >

        return df

    def detect_anomalies(self):
        if self.historical_data.empty:
            print("No historical data available.")
            return None

        anomalies = self.historical_data.copy()

        # Calculate the change in river level
        anomalies['river_level_change'] = anomalies.groupby('location_name')['ri

        # Detect sudden changes in river level
        anomalies['river_level_anomaly'] = np.abs(anomalies['river_level_change'

        # Detect unusual rainfall
        anomalies['rainfall_anomaly'] = anomalies['rainfall'] > self.rainfall_th

        # Overall anomaly flag
        anomalies['is_anomaly'] = anomalies['river_level_anomaly'] | anomalies['

        return anomalies[anomalies['is_anomaly']]

# Test the flood monitoring system
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = FloodMonitoringSystem(data_directory)

# Simulate real-time monitoring
for _ in range(5):  # Simulate 5 data collection cycles
    latest_data = flood_monitor.process_latest_data()
    if latest_data is not None:
        print(f"Processed data at {latest_data.index[0]}:")
        print(latest_data)

        anomalies = flood_monitor.detect_anomalies()
        if not anomalies.empty:
            print("\nDetected anomalies:")
            print(anomalies)
        else:
            print("\nNo anomalies detected.")

    print("\n" + "="*50 + "\n")

    # In a real system, you would wait for the next data collection cycle here
    # For this simulation, we're just running the loop multiple times
```

```
Processed data at 2025-02-06 21:00:00+00:00:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

==================================================

Processed data at 2025-02-06 21:00:00+00:00:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

==================================================

Processed data at 2025-02-06 21:00:00+00:00:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
```

```
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

==================================================

Processed data at 2025-02-06 21:00:00+00:00:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

==================================================

Processed data at 2025-02-06 21:00:00+00:00:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

==================================================
```

In [ ]:

```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
```

```
import time

class FloodMonitoringSystem:
    def __init__(self, data_directory, river_level_threshold=0.05, rainfall_thre
        self.data_directory = data_directory
        self.river_level_threshold = river_level_threshold
        self.rainfall_threshold = rainfall_threshold
        self.lookback_period = lookback_period
        self.historical_data = pd.DataFrame()
        self.alert_levels = {
            0: "Normal",
            1: "Advisory",
            2: "Warning",
            3: "Critical"
        }
        self.last_processed_file = None

    def get_latest_csv(self):
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.
        if not csv_files:
            return None
        latest_file = max(csv_files, key=lambda x: os.path.getctime(os.path.join
        return os.path.join(self.data_directory, latest_file)

    def process_latest_data(self):
        latest_file = self.get_latest_csv()
        if not latest_file or latest_file == self.last_processed_file:
            print("No new data to process.")
            return None

        self.last_processed_file = latest_file
        df = pd.read_csv(latest_file)
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
        df['rainfall_timestamp'] = pd.to_datetime(df['rainfall_timestamp'])
        df.set_index('river_timestamp', inplace=True)

        # Append new data to historical data
        self.historical_data = pd.concat([self.historical_data, df]).drop_duplic

        # Keep only recent data within the lookback period
        self.historical_data = self.historical_data[self.historical_data.index >

        return df

    def detect_anomalies(self):
        if self.historical_data.empty:
            print("No historical data available.")
            return None

        anomalies = self.historical_data.copy()

        # Calculate the change in river level
        anomalies['river_level_change'] = anomalies.groupby('location_name')['ri

        # Detect sudden changes in river level
        anomalies['river_level_anomaly'] = np.abs(anomalies['river_level_change'

        # Detect unusual rainfall
        anomalies['rainfall_anomaly'] = anomalies['rainfall'] > self.rainfall_th
```

```
        # Overall anomaly flag
        anomalies['is_anomaly'] = anomalies['river_level_anomaly'] | anomalies['

        return anomalies[anomalies['is_anomaly']]

    def generate_alerts(self, anomalies):
        if anomalies.empty:
            return []

        alerts = []
        for _, row in anomalies.iterrows():
            alert_level = 0
            reasons = []

            if row['river_level_anomaly']:
                alert_level = max(alert_level, 2)
                reasons.append(f"Sudden change in river level: {row['river_level

            if row['rainfall_anomaly']:
                alert_level = max(alert_level, 1)
                reasons.append(f"High rainfall: {row['rainfall']:.1f}mm")

            if alert_level > 0:
                alert = {
                    'timestamp': row.name,
                    'location': row['location_name'],
                    'level': self.alert_levels[alert_level],
                    'river_level': row['river_level'],
                    'rainfall': row['rainfall'],
                    'reasons': reasons
                }
                alerts.append(alert)

        return alerts

# Set up the flood monitoring system
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = FloodMonitoringSystem(data_directory)

# Continuous monitoring loop
try:
    while True:
        latest_data = flood_monitor.process_latest_data()
        if latest_data is not None:
            print(f"\nProcessed new data at {datetime.now()}:")
            print(latest_data)

            anomalies = flood_monitor.detect_anomalies()
            if not anomalies.empty:
                print("\nDetected anomalies:")
                print(anomalies)

                alerts = flood_monitor.generate_alerts(anomalies)
                for alert in alerts:
                    print("\nALERT:")
                    print(f"Time: {alert['timestamp']}")
                    print(f"Location: {alert['location']}")
                    print(f"Alert Level: {alert['level']}")
                    print(f"River Level: {alert['river_level']:.3f}m")
                    print(f"Rainfall: {alert['rainfall']:.1f}mm")
```

```
                    print("Reasons:")
                    for reason in alert['reasons']:
                        print(f"- {reason}")
                else:
                    print("\nNo anomalies detected.")

            # Wait for 15 minutes before the next check
            time.sleep(900)  # 900 seconds = 15 minutes

    except KeyboardInterrupt:
        print("\nMonitoring stopped.")
```

```
Processed new data at 2025-02-06 21:32:23.895660:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:00:00+00:00        0.185       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.950       0.0 2025-02-06 21:00:00+00:00
2025-02-06 21:00:00+00:00        0.323       0.0 2025-02-06 21:00:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:00:00+00:00               Rochdale            690203
2025-02-06 21:00:00+00:00  Manchester Racecourse            690510
2025-02-06 21:00:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:00:00+00:00               561613
2025-02-06 21:00:00+00:00               562992
2025-02-06 21:00:00+00:00               562656

No anomalies detected.

Processed new data at 2025-02-06 21:47:24.021770:
                           river_level  rainfall        rainfall_timestamp  \
river_timestamp
2025-02-06 21:15:00+00:00        0.185       0.0 2025-02-06 21:15:00+00:00
2025-02-06 21:15:00+00:00        0.951       0.0 2025-02-06 21:15:00+00:00
2025-02-06 21:15:00+00:00        0.322       0.0 2025-02-06 21:15:00+00:00

                                   location_name  river_station_id  \
river_timestamp
2025-02-06 21:15:00+00:00               Rochdale            690203
2025-02-06 21:15:00+00:00  Manchester Racecourse            690510
2025-02-06 21:15:00+00:00            Bury Ground            690160

                           rainfall_station_id
river_timestamp
2025-02-06 21:15:00+00:00               561613
2025-02-06 21:15:00+00:00               562992
2025-02-06 21:15:00+00:00               562656

No anomalies detected.
```

Implement a Database for Historical Data Storage

In [1]:

```
import sqlite3
from datetime import datetime

class FloodMonitoringSystem:
```

```
    def __init__(self, data_directory, db_path='flood_monitoring.db'):
        # ... (previous initialization code) ...
        self.db_path = db_path
        self.init_database()

    def init_database(self):
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS river_data (
                timestamp TEXT,
                location TEXT,
                river_level REAL,
                rainfall REAL,
                PRIMARY KEY (timestamp, location)
            )
        ''')
        conn.commit()
        conn.close()

    def save_to_database(self, df):
        conn = sqlite3.connect(self.db_path)
        df.reset_index().to_sql('river_data', conn, if_exists='append', index=Fa
        conn.close()

    def process_latest_data(self):
        # ... (previous code) ...
        if latest_data is not None:
            self.save_to_database(latest_data)
        return latest_data

    def get_historical_data(self, location, lookback_days=30):
        conn = sqlite3.connect(self.db_path)
        query = f'''
            SELECT * FROM river_data
            WHERE location = ? AND timestamp > datetime('now', '-{lookback_days}
            ORDER BY timestamp
        '''
        df = pd.read_sql_query(query, conn, params=(location,))
        conn.close()
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        df.set_index('timestamp', inplace=True)
        return df
```

Implement Dynamic Thresholds

In [2]:
```
class FloodMonitoringSystem:
    # ... (previous methods) ...

    def calculate_dynamic_thresholds(self, location):
        historical_data = self.get_historical_data(location)
        if historical_data.empty:
            return self.river_level_threshold, self.rainfall_threshold

        river_level_threshold = historical_data['river_level'].mean() + 2 * hist
        rainfall_threshold = historical_data['rainfall'].mean() + 2 * historical

        return river_level_threshold, rainfall_threshold
```

```
    def detect_anomalies(self):
        if self.historical_data.empty:
            print("No historical data available.")
            return None

        anomalies = self.historical_data.copy()

        for location in anomalies['location_name'].unique():
            river_level_threshold, rainfall_threshold = self.calculate_dynamic_t

            location_data = anomalies[anomalies['location_name'] == location]
            anomalies.loc[location_data.index, 'river_level_change'] = location_
            anomalies.loc[location_data.index, 'river_level_anomaly'] = abs(anom
            anomalies.loc[location_data.index, 'rainfall_anomaly'] = anomalies.l

        anomalies['is_anomaly'] = anomalies['river_level_anomaly'] | anomalies['

        return anomalies[anomalies['is_anomaly']]
```

Implement a Simple Visualization

In [3]:
```
import matplotlib.pyplot as plt

class FloodMonitoringSystem:
    # ... (previous methods) ...

    def plot_river_levels(self, location, days=7):
        data = self.get_historical_data(location, lookback_days=days)
        plt.figure(figsize=(12, 6))
        plt.plot(data.index, data['river_level'])
        plt.title(f'River Levels for {location} (Last {days} days)')
        plt.xlabel('Date')
        plt.ylabel('River Level (m)')
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(f'{location}_river_levels.png')
        plt.close()

    def plot_all_locations(self, days=7):
        for location in self.historical_data['location_name'].unique():
            self.plot_river_levels(location, days)
```

Enhance the Main Loop

In [8]:
```
def init_database(self):
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS river_data (
            river_timestamp TEXT,
            rainfall_timestamp TEXT,
            location_name TEXT,
            river_level REAL,
            rainfall REAL,
            river_station_id INTEGER,
            rainfall_station_id INTEGER,
            PRIMARY KEY (river_timestamp, location_name)
        )
```

```
    ''')
    conn.commit()
    conn.close()
```

In [9]:
```
def save_to_database(self, df):
    conn = sqlite3.connect(self.db_path)
    df_to_save = df.reset_index()
    df_to_save.to_sql('river_data', conn, if_exists='append', index=False)
    conn.close()
```

In [10]:
```
def get_historical_data(self, location, lookback_days=30):
    conn = sqlite3.connect(self.db_path)
    query = f'''
        SELECT * FROM river_data
        WHERE location_name = ? AND river_timestamp > datetime('now', '-{lookbac
        ORDER BY river_timestamp
    '''
    df = pd.read_sql_query(query, conn, params=(location,))
    conn.close()
    df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
    df.set_index('river_timestamp', inplace=True)
    return df
```

In [13]:
```
class FloodMonitoringSystem:
    def __init__(self, data_directory, db_path='flood_monitoring.db'):
        self.data_directory = data_directory
        self.db_path = db_path
        self.historical_data = pd.DataFrame()
        self.last_processed_file = None
        self.river_level_threshold = 0.05
        self.rainfall_threshold = 2
        self.lookback_period = timedelta(hours=1)
        self.alert_levels = {0: "Normal", 1: "Advisory", 2: "Warning", 3: "Criti
        self.init_database()
```

In [14]:
```
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'

def main():
    flood_monitor = FloodMonitoringSystem(data_directory)
    # ... rest of the main function
```

In [15]:
```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import sqlite3
import matplotlib.pyplot as plt

# Define the data directory
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'

class FloodMonitoringSystem:
    def __init__(self, data_directory):
        self.data_directory = data_directory
        self.db_path = 'flood_monitoring.db'
        self.init_database()

    def init_database(self):
```

```
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS river_data (
                timestamp TEXT,
                location TEXT,
                river_level REAL,
                rainfall REAL,
                PRIMARY KEY (timestamp, location)
            )
        ''')
        conn.commit()
        conn.close()

    def get_latest_csv(self):
        csv_files = [f for f in os.listdir(self.data_directory) if f.endswith('.
        if not csv_files:
            return None
        return max(csv_files, key=lambda x: os.path.getctime(os.path.join(self.d

# Test the basic setup
flood_monitor = FloodMonitoringSystem(data_directory)
latest_file = flood_monitor.get_latest_csv()
print(f"Latest CSV file: {latest_file}")
```

```
Latest CSV file: combined_data_20250206_232148.csv
```

In [21]:

```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

class RealTimeFloodMonitor:
    def __init__(self, data_directory):
        """
        Initialize the real-time flood monitoring system

        Args:
        - data_directory: Path to directory containing CSV files
        """
        self.data_directory = data_directory

        # Stations tracked
        self.stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

        # Establish baseline parameters for each station
        self.station_baselines = {
            'Bury Ground': {
                'normal_range': (0.320, 0.330),
                'warning_range': (0.330, 0.340),
                'critical_range': (0.340, float('inf'))
            },
            'Manchester Racecourse': {
                'normal_range': (0.940, 0.960),
                'warning_range': (0.960, 0.970),
                'critical_range': (0.970, float('inf'))
            },
            'Rochdale': {
                'normal_range': (0.180, 0.190),
                'warning_range': (0.190, 0.200),
```

```
                'critical_range': (0.200, float('inf'))
            }
        }

    def find_latest_csv(self):
        """
        Find the most recent CSV file in the data directory

        Returns:
        - Path to the most recent CSV file
        """
        try:
            # List all CSV files
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            # Find the most recently created file
            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file

        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def process_latest_data(self):
        """
        Process the latest CSV file

        Returns:
        - Processed DataFrame or None
        """
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the entire file content
            with open(latest_file, 'r') as f:
                file_content = f.read()
                print("Full File Content:")
                print(file_content)

            # Read the CSV file using pandas
            df = pd.read_csv(latest_file)

            # Print column names and first few rows for debugging
            print("\nColumn Names:")
            print(df.columns)
            print("\nFirst Few Rows:")
            print(df.head())

            # Validate data
```

```
            print("\nData Processing Summary:")
            print(f"Total Readings: {len(df)}")
            print("Stations:", df['location_name'].unique() if 'location_name' i

            # Detailed station analysis
            for station in self.stations:
                print(f"\n{station} Analysis:")
                station_data = df[df['location_name'] == station]

                if not station_data.empty:
                    print("River Level:", station_data['river_level'].values[0])
                    print("Rainfall:", station_data['rainfall'].values[0])
                else:
                    print(f"No data found for {station}")

            return df

        except Exception as e:
            print(f"Error processing data: {e}")
            return None

# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = RealTimeFloodMonitor(data_directory)

# Process latest data
latest_data = flood_monitor.process_latest_data()

if latest_data is not None:
    print("\nData Processing Successful")
    print(latest_data)
```

```
Full File Content:
river_level,river_timestamp,rainfall,rainfall_timestamp,location_name,river_stati
on_id,rainfall_station_id
0.185,2025-02-06T23:15:00Z,0.0,2025-02-06T23:15:00Z,Rochdale,690203,561613
0.951,2025-02-06T23:15:00Z,0.0,2025-02-06T23:15:00Z,Manchester Racecourse,690510,
562992
0.323,2025-02-06T23:15:00Z,0.0,2025-02-06T23:15:00Z,Bury Ground,690160,562656


Column Names:
Index(['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp',
       'location_name', 'river_station_id', 'rainfall_station_id'],
      dtype='object')

First Few Rows:
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0        0.185  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   
1        0.951  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   
2        0.323  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   

           location_name  river_station_id  rainfall_station_id  
0               Rochdale            690203               561613  
1  Manchester Racecourse            690510               562992  
2            Bury Ground            690160               562656  

Data Processing Summary:
Total Readings: 3
Stations: ['Rochdale' 'Manchester Racecourse' 'Bury Ground']

Rochdale Analysis:
River Level: 0.185
Rainfall: 0.0

Manchester Racecourse Analysis:
River Level: 0.951
Rainfall: 0.0

Bury Ground Analysis:
River Level: 0.323
Rainfall: 0.0

Data Processing Successful
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0        0.185  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   
1        0.951  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   
2        0.323  2025-02-06T23:15:00Z       0.0  2025-02-06T23:15:00Z   

           location_name  river_station_id  rainfall_station_id  
0               Rochdale            690203               561613  
1  Manchester Racecourse            690510               562992  
2            Bury Ground            690160               562656  
```

In [23]:

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

class RealTimeFloodMonitor:
    def __init__(self, data_directory):
        """
```

```
        Initialize the real-time flood monitoring system

        Args:
        - data_directory: Path to directory containing CSV files
        """
        self.data_directory = data_directory

        # Anomaly detection parameters
        self.anomaly_thresholds = {
            'Bury Ground': {
                'level_low_threshold': 0.300,  # Lower warning level
                'level_high_threshold': 0.350,  # Upper warning level
                'level_critical_threshold': 0.400,
                'change_threshold': 0.02  # Significant level change
            },
            'Manchester Racecourse': {
                'level_low_threshold': 0.900,
                'level_high_threshold': 1.000,
                'level_critical_threshold': 1.100,
                'change_threshold': 0.05
            },
            'Rochdale': {
                'level_low_threshold': 0.170,
                'level_high_threshold': 0.200,
                'level_critical_threshold': 0.230,
                'change_threshold': 0.01
            }
        }

        # Store historical data for comparison
        self.historical_data = pd.DataFrame()

    def find_latest_csv(self):
        """
        Find the most recent CSV file in the data directory

        Returns:
        - Path to the most recent CSV file
        """
        try:
            # List all CSV files
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            # Find the most recently created file
            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file

        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def detect_anomalies(self, current_data):
        """
```

```
        Detect anomalies in river levels

        Args:
        - current_data: DataFrame with current readings

        Returns:
        - Dictionary of anomalies for each station
        """
        anomalies = {}

        for station in self.anomaly_thresholds.keys():
            station_data = current_data[current_data['location_name'] == station
            thresholds = self.anomaly_thresholds[station]

            # Current river level
            current_level = station_data['river_level'].values[0]

            # Determine anomaly level
            anomaly_status = 'NORMAL'

            # Check against thresholds
            if current_level >= thresholds['level_critical_threshold']:
                anomaly_status = 'CRITICAL'
            elif current_level >= thresholds['level_high_threshold']:
                anomaly_status = 'HIGH'
            elif current_level <= thresholds['level_low_threshold']:
                anomaly_status = 'LOW'

            # Prepare anomaly details
            anomalies[station] = {
                'current_level': current_level,
                'status': anomaly_status,
                'timestamp': station_data['river_timestamp'].values[0],
                'rainfall': station_data['rainfall'].values[0]
            }

        return anomalies

    def process_latest_data(self):
        """
        Main method to process latest data and detect anomalies

        Returns:
        - Processed data and anomalies
        """
        # Find and process latest CSV
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the CSV file
            df = pd.read_csv(latest_file)

            # Detect anomalies
            anomalies = self.detect_anomalies(df)

            # Print anomaly results
```

```
            print("\nAnomaly Detection Results:")
            for station, details in anomalies.items():
                print(f"\n{station}:")
                print(f"  Current Level: {details['current_level']} m")
                print(f"  Status: {details['status']}")
                print(f"  Timestamp: {details['timestamp']}")
                print(f"  Rainfall: {details['rainfall']} mm")

            return {
                'data': df,
                'anomalies': anomalies
            }

        except Exception as e:
            print(f"Error processing data: {e}")
            return None

# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = RealTimeFloodMonitor(data_directory)

# Process latest data and detect anomalies
result = flood_monitor.process_latest_data()

if result:
    print("\nData and Anomaly Processing Complete")
```

```
Anomaly Detection Results:

Bury Ground:
  Current Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

Manchester Racecourse:
  Current Level: 0.951 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

Rochdale:
  Current Level: 0.185 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

Data and Anomaly Processing Complete
```

In [24]:

```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

class RealTimeFloodMonitor:
    def __init__(self, data_directory):
        self.data_directory = data_directory

        # Dynamic thresholds based on observed data
        self.anomaly_thresholds = {
```

```
            'Bury Ground': {
                'normal_min': 0.300,
                'normal_max': 0.340,
                'low_threshold': 0.280,
                'high_threshold': 0.360,
                'critical_threshold': 0.380
            },
            'Manchester Racecourse': {
                'normal_min': 0.920,
                'normal_max': 0.970,
                'low_threshold': 0.880,
                'high_threshold': 1.000,
                'critical_threshold': 1.050
            },
            'Rochdale': {
                'normal_min': 0.170,
                'normal_max': 0.190,
                'low_threshold': 0.150,
                'high_threshold': 0.210,
                'critical_threshold': 0.230
            }
        }

        # Store historical data for trend analysis
        self.historical_data = {}

    def find_latest_csv(self):
        """Find the most recent CSV file"""
        try:
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file
        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def detect_anomalies(self, current_data):
        """
        Advanced anomaly detection with multiple risk factors
        """
        anomalies = {}

        for station in self.anomaly_thresholds.keys():
            station_data = current_data[current_data['location_name'] == station
            thresholds = self.anomaly_thresholds[station]

            # Current readings
            current_level = station_data['river_level'].values[0]
            current_rainfall = station_data['rainfall'].values[0]
            current_timestamp = station_data['river_timestamp'].values[0]

            # Determine anomaly status
```

```
            status = 'NORMAL'
            risk_factors = []

            # Level-based risk assessment
            if current_level <= thresholds['low_threshold']:
                status = 'LOW_LEVEL'
                risk_factors.append('Low water level')
            elif current_level >= thresholds['critical_threshold']:
                status = 'CRITICAL'
                risk_factors.append('Critically high water level')
            elif current_level >= thresholds['high_threshold']:
                status = 'HIGH'
                risk_factors.append('High water level')

            # Rainfall risk
            if current_rainfall > 0:
                risk_factors.append(f'Rainfall detected: {current_rainfall} mm')

            # Store anomaly information
            anomalies[station] = {
                'current_level': current_level,
                'status': status,
                'timestamp': current_timestamp,
                'rainfall': current_rainfall,
                'risk_factors': risk_factors
            }

        return anomalies

    def process_latest_data(self):
        """
        Process latest data and perform anomaly detection
        """
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the CSV file
            df = pd.read_csv(latest_file)

            # Detect anomalies
            anomalies = self.detect_anomalies(df)

            # Print detailed anomaly results
            print("\n--- Flood Monitoring Anomaly Detection ---")
            for station, details in anomalies.items():
                print(f"\n{station} Station:")
                print(f"  Current Water Level: {details['current_level']} m")
                print(f"  Status: {details['status']}")
                print(f"  Timestamp: {details['timestamp']}")
                print(f"  Rainfall: {details['rainfall']} mm")

                if details['risk_factors']:
                    print("  Risk Factors:")
                    for factor in details['risk_factors']:
                        print(f"    - {factor}")
```

```
            return {
                'data': df,
                'anomalies': anomalies
            }

        except Exception as e:
            print(f"Error processing data: {e}")
            return None

# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = RealTimeFloodMonitor(data_directory)

# Process latest data and detect anomalies
result = flood_monitor.process_latest_data()

if result:
    print("\n--- Monitoring Analysis Complete ---")
```

```
--- Flood Monitoring Anomaly Detection ---

Bury Ground Station:
  Current Water Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

Manchester Racecourse Station:
  Current Water Level: 0.951 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

Rochdale Station:
  Current Water Level: 0.185 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm

--- Monitoring Analysis Complete ---
```

In [25]:

```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

class RealTimeFloodMonitor:
    def __init__(self, data_directory, history_depth=10):
        self.data_directory = data_directory
        self.history_depth = history_depth

        # Historical data storage for each station
        self.historical_data = {
            'Bury Ground': [],
            'Manchester Racecourse': [],
            'Rochdale': []
        }

    def find_latest_csv(self):
        """Find the most recent CSV file"""
```

```
        try:
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file
        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def update_historical_data(self, current_data):
        """
        Update historical data for each station
        Maintains a rolling window of recent measurements
        """
        for station in self.historical_data.keys():
            station_data = current_data[current_data['location_name'] == station

            # Extract current reading
            current_reading = {
                'level': station_data['river_level'].values[0],
                'rainfall': station_data['rainfall'].values[0],
                'timestamp': station_data['river_timestamp'].values[0]
            }

            # Add to historical data
            station_history = self.historical_data[station]
            station_history.append(current_reading)

            # Maintain only the last N measurements
            if len(station_history) > self.history_depth:
                station_history.pop(0)

    def analyze_station_trends(self, station):
        """
        Analyze trends for a specific station

        Returns:
        - Trend insights
        - Potential risk indicators
        """
        history = self.historical_data[station]

        if len(history) < 2:
            return {'trend': 'Insufficient data', 'risk': 'Unknown'}

        # Calculate changes
        levels = [entry['level'] for entry in history]
        rainfalls = [entry['rainfall'] for entry in history]

        # Basic trend analysis
        level_changes = np.diff(levels)
        avg_level_change = np.mean(level_changes)
        level_change_variability = np.std(level_changes)
```

```
        # Rainfall analysis
        total_rainfall = sum(rainfalls)

        # Trend classification
        if abs(avg_level_change) > 0.01:
            trend = 'Increasing' if avg_level_change > 0 else 'Decreasing'
        else:
            trend = 'Stable'

        # Risk assessment
        risk = 'Low'
        if level_change_variability > 0.02:
            risk = 'Moderate'
        if total_rainfall > 0:
            risk = 'High'

        return {
            'trend': trend,
            'risk': risk,
            'avg_change': avg_level_change,
            'change_variability': level_change_variability,
            'total_rainfall': total_rainfall
        }

    def detect_anomalies(self, current_data):
        """
        Advanced anomaly detection with trend analysis
        """
        # Update historical data
        self.update_historical_data(current_data)

        anomalies = {}

        for station in ['Bury Ground', 'Manchester Racecourse', 'Rochdale']:
            station_data = current_data[current_data['location_name'] == station

            # Current readings
            current_level = station_data['river_level'].values[0]
            current_rainfall = station_data['rainfall'].values[0]
            current_timestamp = station_data['river_timestamp'].values[0]

            # Analyze trends
            trend_analysis = self.analyze_station_trends(station)

            # Determine anomaly status
            status = 'NORMAL'
            risk_factors = []

            # Apply trend-based risk assessment
            if trend_analysis['risk'] == 'High':
                status = 'ELEVATED'
                risk_factors.append('High trend variability')

            if trend_analysis['total_rainfall'] > 0:
                risk_factors.append(f'Rainfall detected: {trend_analysis["total_

            # Store anomaly information
            anomalies[station] = {
                'current_level': current_level,
```

```
                'status': status,
                'timestamp': current_timestamp,
                'rainfall': current_rainfall,
                'trend_analysis': trend_analysis,
                'risk_factors': risk_factors
            }

        return anomalies

    def process_latest_data(self):
        """
        Process latest data and perform anomaly detection
        """
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the CSV file
            df = pd.read_csv(latest_file)

            # Detect anomalies
            anomalies = self.detect_anomalies(df)

            # Print detailed anomaly results
            print("\n--- Flood Monitoring Trend Analysis ---")
            for station, details in anomalies.items():
                print(f"\n{station} Station:")
                print(f"  Current Water Level: {details['current_level']} m")
                print(f"  Status: {details['status']}")
                print(f"  Timestamp: {details['timestamp']}")
                print(f"  Rainfall: {details['rainfall']} mm")

                print("  Trend Analysis:")
                trend = details['trend_analysis']
                print(f"    Trend: {trend['trend']}")
                print(f"    Risk Level: {trend['risk']}")
                print(f"    Avg Level Change: {trend['avg_change']:.4f} m")

                if details['risk_factors']:
                    print("  Risk Factors:")
                    for factor in details['risk_factors']:
                        print(f"    - {factor}")

            return {
                'data': df,
                'anomalies': anomalies
            }

        except Exception as e:
            print(f"Error processing data: {e}")
            return None

# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = RealTimeFloodMonitor(data_directory)

# Process latest data and detect anomalies
```

```
result = flood_monitor.process_latest_data()

# Subsequent runs will now have historical context
result = flood_monitor.process_latest_data()
```

```
Error processing data: 'total_rainfall'

--- Flood Monitoring Trend Analysis ---

Bury Ground Station:
  Current Water Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Stable
    Risk Level: Low
    Avg Level Change: 0.0000 m

Manchester Racecourse Station:
  Current Water Level: 0.951 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Stable
    Risk Level: Low
    Avg Level Change: 0.0000 m

Rochdale Station:
  Current Water Level: 0.185 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Stable
    Risk Level: Low
    Avg Level Change: 0.0000 m
```

In [26]:
```
import os
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

class RealTimeFloodMonitor:
    def __init__(self, data_directory, history_depth=10):
        self.data_directory = data_directory
        self.history_depth = history_depth

        # Historical data storage for each station
        self.historical_data = {
            'Bury Ground': [],
            'Manchester Racecourse': [],
            'Rochdale': []
        }

    def find_latest_csv(self):
        """Find the most recent CSV file"""
        try:
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit
```

```
            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file
        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def update_historical_data(self, current_data):
        """
        Update historical data for each station
        Maintains a rolling window of recent measurements
        """
        for station in self.historical_data.keys():
            station_data = current_data[current_data['location_name'] == station

            # Extract current reading
            current_reading = {
                'level': station_data['river_level'].values[0],
                'rainfall': station_data['rainfall'].values[0],
                'timestamp': station_data['river_timestamp'].values[0]
            }

            # Add to historical data
            station_history = self.historical_data[station]
            station_history.append(current_reading)

            # Maintain only the last N measurements
            if len(station_history) > self.history_depth:
                station_history.pop(0)

    def analyze_station_trends(self, station):
        """
        Analyze trends for a specific station

        Returns:
        - Trend insights
        - Potential risk indicators
        """
        history = self.historical_data[station]

        if len(history) < 2:
            return {
                'trend': 'Insufficient data',
                'risk': 'Unknown',
                'avg_change': 0,
                'change_variability': 0,
                'total_rainfall': 0
            }

        # Calculate changes
        levels = [entry['level'] for entry in history]
        rainfalls = [entry['rainfall'] for entry in history]
```

```
        # Basic trend analysis
        level_changes = np.diff(levels)
        avg_level_change = np.mean(level_changes) if len(level_changes) > 0 else
        level_change_variability = np.std(level_changes) if len(level_changes) >

        # Rainfall analysis
        total_rainfall = sum(rainfalls)

        # Trend classification
        if abs(avg_level_change) > 0.01:
            trend = 'Increasing' if avg_level_change > 0 else 'Decreasing'
        else:
            trend = 'Stable'

        # Risk assessment
        risk = 'Low'
        if level_change_variability > 0.02:
            risk = 'Moderate'
        if total_rainfall > 0:
            risk = 'High'

        return {
            'trend': trend,
            'risk': risk,
            'avg_change': avg_level_change,
            'change_variability': level_change_variability,
            'total_rainfall': total_rainfall
        }

    def detect_anomalies(self, current_data):
        """
        Advanced anomaly detection with trend analysis
        """
        # Update historical data
        self.update_historical_data(current_data)

        anomalies = {}

        for station in ['Bury Ground', 'Manchester Racecourse', 'Rochdale']:
            station_data = current_data[current_data['location_name'] == station

            # Current readings
            current_level = station_data['river_level'].values[0]
            current_rainfall = station_data['rainfall'].values[0]
            current_timestamp = station_data['river_timestamp'].values[0]

            # Analyze trends
            trend_analysis = self.analyze_station_trends(station)

            # Determine anomaly status
            status = 'NORMAL'
            risk_factors = []

            # Apply trend-based risk assessment
            if trend_analysis['risk'] == 'High':
                status = 'ELEVATED'
                risk_factors.append('High trend variability')

            if trend_analysis['total_rainfall'] > 0:
                risk_factors.append(f'Rainfall detected: {trend_analysis["total_
```

```
            # Store anomaly information
            anomalies[station] = {
                'current_level': current_level,
                'status': status,
                'timestamp': current_timestamp,
                'rainfall': current_rainfall,
                'trend_analysis': trend_analysis,
                'risk_factors': risk_factors
            }

        return anomalies

    def process_latest_data(self):
        """
        Process latest data and perform anomaly detection
        """
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the CSV file
            df = pd.read_csv(latest_file)

            # Detect anomalies
            anomalies = self.detect_anomalies(df)

            # Print detailed anomaly results
            print("\n--- Flood Monitoring Trend Analysis ---")
            for station, details in anomalies.items():
                print(f"\n{station} Station:")
                print(f"  Current Water Level: {details['current_level']} m")
                print(f"  Status: {details['status']}")
                print(f"  Timestamp: {details['timestamp']}")
                print(f"  Rainfall: {details['rainfall']} mm")

                print("  Trend Analysis:")
                trend = details['trend_analysis']
                print(f"    Trend: {trend['trend']}")
                print(f"    Risk Level: {trend['risk']}")
                print(f"    Avg Level Change: {trend['avg_change']:.4f} m")
                print(f"    Total Rainfall: {trend['total_rainfall']:.4f} mm")

                if details['risk_factors']:
                    print("  Risk Factors:")
                    for factor in details['risk_factors']:
                        print(f"    - {factor}")

            return {
                'data': df,
                'anomalies': anomalies
            }

        except Exception as e:
            print(f"Error processing data: {e}")
            return None
```

```
# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
flood_monitor = RealTimeFloodMonitor(data_directory)

# Process latest data and detect anomalies
# Multiple runs to build historical context
for _ in range(2):
    result = flood_monitor.process_latest_data()
```

```
--- Flood Monitoring Trend Analysis ---

Bury Ground Station:
  Current Water Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Insufficient data
    Risk Level: Unknown
    Avg Level Change: 0.0000 m
    Total Rainfall: 0.0000 mm

Manchester Racecourse Station:
  Current Water Level: 0.951 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Insufficient data
    Risk Level: Unknown
    Avg Level Change: 0.0000 m
    Total Rainfall: 0.0000 mm

Rochdale Station:
  Current Water Level: 0.185 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Insufficient data
    Risk Level: Unknown
    Avg Level Change: 0.0000 m
    Total Rainfall: 0.0000 mm

--- Flood Monitoring Trend Analysis ---

Bury Ground Station:
  Current Water Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Stable
    Risk Level: Low
    Avg Level Change: 0.0000 m
    Total Rainfall: 0.0000 mm

Manchester Racecourse Station:
  Current Water Level: 0.951 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:15:00Z
  Rainfall: 0.0 mm
  Trend Analysis:
    Trend: Stable
    Risk Level: Low
    Avg Level Change: 0.0000 m
    Total Rainfall: 0.0000 mm

Rochdale Station:
```

```
Current Water Level: 0.185 m
Status: NORMAL
Timestamp: 2025-02-06T23:15:00Z
Rainfall: 0.0 mm
Trend Analysis:
  Trend: Stable
  Risk Level: Low
  Avg Level Change: 0.0000 m
  Total Rainfall: 0.0000 mm
```

In [27]:

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

# Define historical data directory
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data

def inspect_historical_data():
    """
    Scan and analyze available historical datasets
    """
    # Check available files
    historical_files = os.listdir(historical_data_dir)
    print("Available Historical Files:")
    for file in historical_files:
        print(f"- {file}")

    # Analyze specific files
    for file in historical_files:
        file_path = os.path.join(historical_data_dir, file)
        try:
            df = pd.read_csv(file_path)
            print(f"\nAnalysis for {file}:")
            print(df.info())
            print("\nFirst few rows:")
            print(df.head())
        except Exception as e:
            print(f"Error reading {file}: {e}")

# Run inspection
inspect_historical_data()
```

```
Available Historical Files:
- bury_daily_flow.csv
- bury_daily_rainfall.csv
- bury_peak_flow.csv
- manchester_peak_flow.csv
- processed
- rochdale_daily_flow.csv
- rochdale_daily_rainfall.csv
- rochdale_peak_flow.csv

Analysis for bury_daily_flow.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9928 entries, 0 to 9927
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    9928 non-null   object 
 1   Flow    9928 non-null   float64
 2   Extra   0 non-null      float64
dtypes: float64(2), object(1)
memory usage: 232.8+ KB
None

First few rows:
         Date   Flow  Extra
0  1995-11-22  0.897    NaN
1  1995-11-23  0.831    NaN
2  1995-11-24  0.991    NaN
3  1995-11-25  1.080    NaN
4  1995-11-26  1.124    NaN

Analysis for bury_daily_rainfall.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20819 entries, 0 to 20818
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Date      20819 non-null  object 
 1   Rainfall  20819 non-null  float64
 2   Extra     20819 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 488.1+ KB
None

First few rows:
         Date  Rainfall  Extra
0  1961-01-01       9.4   1000
1  1961-01-02      13.7   1000
2  1961-01-03       3.0   1000
3  1961-01-04       0.1   1000
4  1961-01-05      13.0   1000

Analysis for bury_peak_flow.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Water Year   51 non-null     object 
 1   Date         51 non-null     object 
```

```
 2   Time         51 non-null     object
 3   Stage (m)    51 non-null     float64
 4   Flow (m3/s)  51 non-null     float64
 5   Rating       48 non-null     object
 6   Datetime     51 non-null     object
dtypes: float64(2), object(5)
memory usage: 2.9+ KB
None

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1972-1973  1973-01-12  00:00:00      1.255       78.130       NaN
1  1973-1974  1974-02-11  00:00:00      1.473      118.020       NaN
2  1974-1975  1975-01-21  00:00:00      1.450      113.410       NaN
3  1975-1976  1976-01-02  17:45:00      1.468      116.886  In Range
4  1976-1977  1977-09-30  20:00:00      1.258       78.636  In Range

              Datetime
0  1973-01-12 00:00:00
1  1974-02-11 00:00:00
2  1975-01-21 00:00:00
3  1976-01-02 17:45:00
4  1977-09-30 20:00:00

Analysis for manchester_peak_flow.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82 entries, 0 to 81
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Water Year   82 non-null     object
 1   Date         82 non-null     object
 2   Time         82 non-null     object
 3   Stage (m)    82 non-null     float64
 4   Flow (m3/s)  82 non-null     float64
 5   Rating       82 non-null     object
 6   Datetime     82 non-null     object
dtypes: float64(2), object(5)
memory usage: 4.6+ KB
None

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)   Rating  \
0  1941-1942  1941-10-24  00:00:00       3.47        269.0  Extrap.
1  1942-1943  1942-10-17  00:00:00       3.16        223.0  Extrap.
2  1943-1944  1944-01-23  00:00:00       4.10        374.0  Extrap.
3  1944-1945  1945-02-02  00:00:00       3.90        339.0  Extrap.
4  1945-1946  1946-09-20  00:00:00       5.33        500.0  Extrap.

              Datetime
0  1941-10-24 00:00:00
1  1942-10-17 00:00:00
2  1944-01-23 00:00:00
3  1945-02-02 00:00:00
4  1946-09-20 00:00:00
Error reading processed: [Errno 13] Permission denied: 'C:/Users/Administrator/NE
WPROJECT/cleaned_data/river_data/historical\\processed'

Analysis for rochdale_daily_flow.csv:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 11118 entries, 0 to 11117
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    11118 non-null  object 
 1   Flow    11118 non-null  float64
 2   Extra   0 non-null      float64
dtypes: float64(2), object(1)
memory usage: 260.7+ KB
None

First few rows:
         Date   Flow  Extra
0  1993-02-26  1.290    NaN
1  1993-02-27  1.060    NaN
2  1993-02-28  0.985    NaN
3  1993-03-01  1.140    NaN
4  1993-03-02  1.180    NaN

Analysis for rochdale_daily_rainfall.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Date      731 non-null    object 
 1   Rainfall  731 non-null    float64
 2   Extra     731 non-null    int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 17.3+ KB
None

First few rows:
         Date  Rainfall  Extra
0  2016-01-01       0.8   2000
1  2016-01-02       3.5   2000
2  2016-01-03      13.3   2000
3  2016-01-04       5.5   2000
4  2016-01-05       6.0   2000

Analysis for rochdale_peak_flow.csv:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31 entries, 0 to 30
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Water Year   31 non-null     object 
 1   Date         31 non-null     object 
 2   Time         31 non-null     object 
 3   Stage (m)    31 non-null     float64
 4   Flow (m3/s)  31 non-null     float64
 5   Rating       31 non-null     object 
 6   Datetime     31 non-null     object 
dtypes: float64(2), object(5)
memory usage: 1.8+ KB
None

First few rows:
  Water Year        Date      Time  Stage (m)  Flow (m3/s)    Rating  \
0  1992-1993  1993-09-13  11:30:00      0.892       21.131  In Range
```

```
1  1993-1994  1993-12-08  23:45:00      1.286       38.328  In Range
2  1994-1995  1995-01-31  23:15:00      1.637       56.671  In Range
3  1995-1996  1996-02-18  03:15:00      0.808       17.976  In Range
4  1996-1997  1996-11-06  02:15:00      1.243       36.269  In Range

              Datetime
0  1993-09-13 11:30:00
1  1993-12-08 23:45:00
2  1995-01-31 23:15:00
3  1996-02-18 03:15:00
4  1996-11-06 02:15:00
```

In [28]:

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

class HistoricalDataProcessor:
    def __init__(self, historical_data_dir):
        self.dir = historical_data_dir

        # Load historical datasets
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rain
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_
        self.rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_da

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.bury_rainfall['Date'] = pd.to_datetime(self.bury_rainfall['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])
        self.rochdale_rainfall['Date'] = pd.to_datetime(self.rochdale_rainfall['

    def calculate_station_baselines(self):
        """
        Calculate statistical baselines for each station
        """
        baselines = {
            'Bury Ground': {
                'flow_mean': self.bury_flow['Flow'].mean(),
                'flow_std': self.bury_flow['Flow'].std(),
                'rainfall_mean': self.bury_rainfall['Rainfall'].mean(),
                'rainfall_std': self.bury_rainfall['Rainfall'].std()
            },
            'Rochdale': {
                'flow_mean': self.rochdale_flow['Flow'].mean(),
                'flow_std': self.rochdale_flow['Flow'].std(),
                'rainfall_mean': self.rochdale_rainfall['Rainfall'].mean(),
                'rainfall_std': self.rochdale_rainfall['Rainfall'].std()
            }
        }

        return baselines

    def create_anomaly_thresholds(self):
        """
        Create anomaly detection thresholds based on historical data
        """
        thresholds = {
            'Bury Ground': {
                'flow_low_threshold': self.bury_flow['Flow'].mean() - (2 * self.
```

```
                'flow_high_threshold': self.bury_flow['Flow'].mean() + (2 * self
                'rainfall_threshold': self.bury_rainfall['Rainfall'].mean() + (1
            },
            'Rochdale': {
                'flow_low_threshold': self.rochdale_flow['Flow'].mean() - (2 * s
                'flow_high_threshold': self.rochdale_flow['Flow'].mean() + (2 *
                'rainfall_threshold': self.rochdale_rainfall['Rainfall'].mean()
            }
        }

        return thresholds

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
data_processor = HistoricalDataProcessor(historical_data_dir)

# Calculate baselines and thresholds
station_baselines = data_processor.calculate_station_baselines()
anomaly_thresholds = data_processor.create_anomaly_thresholds()

print("Station Baselines:")
for station, baseline in station_baselines.items():
    print(f"\n{station}:")
    for key, value in baseline.items():
        print(f"  {key}: {value:.4f}")

print("\nAnomaly Thresholds:")
for station, thresholds in anomaly_thresholds.items():
    print(f"\n{station}:")
    for key, value in thresholds.items():
        print(f"  {key}: {value:.4f}")
```

```
Station Baselines:

Bury Ground:
  flow_mean: 3.8503
  flow_std: 5.3954
  rainfall_mean: 3.7755
  rainfall_std: 6.2099

Rochdale:
  flow_mean: 2.7956
  flow_std: 3.5467
  rainfall_mean: 3.7836
  rainfall_std: 5.8482

Anomaly Thresholds:

Bury Ground:
  flow_low_threshold: -6.9404
  flow_high_threshold: 14.6411
  rainfall_threshold: 13.0904

Rochdale:
  flow_low_threshold: -4.2979
  flow_high_threshold: 9.8890
  rainfall_threshold: 12.5559
```

In [30]:
```
import os
import pandas as pd
```

```
import numpy as np
from datetime import datetime

class HistoricalDataProcessor:
    def __init__(self, historical_data_dir):
        self.dir = historical_data_dir

        # Load historical datasets
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rain
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_
        self.rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_da

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.bury_rainfall['Date'] = pd.to_datetime(self.bury_rainfall['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])
        self.rochdale_rainfall['Date'] = pd.to_datetime(self.rochdale_rainfall['

    def calculate_station_baselines(self):
        """
        Calculate statistical baselines for each station
        """
        baselines = {
            'Bury Ground': {
                'flow_mean': self.bury_flow['Flow'].mean(),
                'flow_std': self.bury_flow['Flow'].std(),
                'rainfall_mean': self.bury_rainfall['Rainfall'].mean(),
                'rainfall_std': self.bury_rainfall['Rainfall'].std()
            },
            'Rochdale': {
                'flow_mean': self.rochdale_flow['Flow'].mean(),
                'flow_std': self.rochdale_flow['Flow'].std(),
                'rainfall_mean': self.rochdale_rainfall['Rainfall'].mean(),
                'rainfall_std': self.rochdale_rainfall['Rainfall'].std()
            }
        }

        return baselines

    def create_anomaly_thresholds(self):
        """
        Create anomaly detection thresholds based on historical data
        """
        thresholds = {
            'Bury Ground': {
                'flow_low_threshold': self.bury_flow['Flow'].mean() - (2 * self.
                'flow_high_threshold': self.bury_flow['Flow'].mean() + (2 * self
                'rainfall_threshold': self.bury_rainfall['Rainfall'].mean() + (1
            },
            'Rochdale': {
                'flow_low_threshold': self.rochdale_flow['Flow'].mean() - (2 * s
                'flow_high_threshold': self.rochdale_flow['Flow'].mean() + (2 *
                'rainfall_threshold': self.rochdale_rainfall['Rainfall'].mean()
            }
        }

        return thresholds

class RealTimeFloodMonitor:
```

```
    def __init__(self, data_directory, historical_data_dir):
        self.data_directory = data_directory

        # Process historical data
        historical_processor = HistoricalDataProcessor(historical_data_dir)
        self.station_baselines = historical_processor.calculate_station_baseline
        self.anomaly_thresholds = historical_processor.create_anomaly_thresholds

        # Historical context tracking
        self.historical_data = {
            'Bury Ground': [],
            'Manchester Racecourse': [],
            'Rochdale': []
        }

    def find_latest_csv(self):
        """Find the most recent CSV file"""
        try:
            csv_files = [f for f in os.listdir(self.data_directory) if f.endswit

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            latest_file = max(
                [os.path.join(self.data_directory, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file
        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def detect_anomalies(self, current_data):
        """
        Advanced anomaly detection using historical baselines
        """
        anomalies = {}

        for station in ['Bury Ground', 'Rochdale']:
            station_data = current_data[current_data['location_name'] == station

            # Current readings
            current_level = station_data['river_level'].values[0]
            current_rainfall = station_data['rainfall'].values[0]
            current_timestamp = station_data['river_timestamp'].values[0]

            # Get station-specific thresholds
            baselines = self.station_baselines[station]
            thresholds = self.anomaly_thresholds[station]

            # Anomaly detection logic
            status = 'NORMAL'
            risk_factors = []

            # Flow-based anomaly detection
            if current_level < thresholds['flow_low_threshold']:
                status = 'LOW_FLOW'
                risk_factors.append('Unusually low river level')
            elif current_level > thresholds['flow_high_threshold']:
```

```
                status = 'HIGH_FLOW'
                risk_factors.append('Unusually high river level')

            # Rainfall-based anomaly detection
            if current_rainfall > thresholds['rainfall_threshold']:
                status = 'ELEVATED'
                risk_factors.append(f'High rainfall: {current_rainfall:.2f} mm')

            # Calculate deviation from historical mean
            level_deviation = abs(current_level - baselines['flow_mean']) / base
            rainfall_deviation = abs(current_rainfall - baselines['rainfall_mean

            # Store anomaly information
            anomalies[station] = {
                'current_level': current_level,
                'status': status,
                'timestamp': current_timestamp,
                'rainfall': current_rainfall,
                'level_deviation': level_deviation,
                'rainfall_deviation': rainfall_deviation,
                'risk_factors': risk_factors
            }

        # Handle Manchester Racecourse (limited historical data)
        manchester_data = current_data[current_data['location_name'] == 'Manches
        if not manchester_data.empty:
            current_level = manchester_data['river_level'].values[0]
            current_rainfall = manchester_data['rainfall'].values[0]
            current_timestamp = manchester_data['river_timestamp'].values[0]

            anomalies['Manchester Racecourse'] = {
                'current_level': current_level,
                'status': 'MONITORING',
                'timestamp': current_timestamp,
                'rainfall': current_rainfall,
                'risk_factors': ['Limited historical data']
            }

        return anomalies

    def process_latest_data(self):
        """
        Process latest data and perform anomaly detection
        """
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No data file to process")
            return None

        try:
            # Read the CSV file
            df = pd.read_csv(latest_file)

            # Detect anomalies
            anomalies = self.detect_anomalies(df)

            # Print detailed anomaly results
            print("\n--- Flood Monitoring Advanced Analysis ---")
            for station, details in anomalies.items():
```

```
                print(f"\n{station} Station:")
                print(f"  Current Water Level: {details['current_level']} m")
                print(f"  Status: {details['status']}")
                print(f"  Timestamp: {details['timestamp']}")
                print(f"  Rainfall: {details['rainfall']} mm")

                # Print additional insights for stations with historical data
                if station in ['Bury Ground', 'Rochdale']:
                    print("  Historical Context:")
                    print(f"    Level Deviation: {details['level_deviation']:.4f
                    print(f"    Rainfall Deviation: {details['rainfall_deviation

                if details.get('risk_factors'):
                    print("  Risk Factors:")
                    for factor in details['risk_factors']:
                        print(f"    - {factor}")

            return {
                'data': df,
                'anomalies': anomalies
            }

        except Exception as e:
            print(f"Error processing data: {e}")
            return None

# Example usage
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data

flood_monitor = RealTimeFloodMonitor(data_directory, historical_data_dir)

# Process latest data and detect anomalies
result = flood_monitor.process_latest_data()
```

```
--- Flood Monitoring Advanced Analysis ---

Bury Ground Station:
  Current Water Level: 0.323 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:30:00Z
  Rainfall: 0.0 mm
  Historical Context:
    Level Deviation: 0.6538
    Rainfall Deviation: 0.6080

Rochdale Station:
  Current Water Level: 0.185 m
  Status: NORMAL
  Timestamp: 2025-02-06T23:30:00Z
  Rainfall: 0.0 mm
  Historical Context:
    Level Deviation: 0.7361
    Rainfall Deviation: 0.6470

Manchester Racecourse Station:
  Current Water Level: 0.951 m
  Status: MONITORING
  Timestamp: 2025-02-06T23:30:00Z
  Rainfall: 0.0 mm
  Risk Factors:
    - Limited historical data
```

Predictive Modeling Preparation

In [34]:

```
import pandas as pd
import numpy as np

# Load historical flow data
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')

# Detailed investigation
print("Bury Ground Flow Data Investigation:")
print("DataFrame Info:")
print(bury_flow.info())

print("\nFirst few rows:")
print(bury_flow.head())

print("\nColumn types:")
print(bury_flow.dtypes)

print("\nChecking for any data issues:")
print("Null values:")
print(bury_flow.isnull().sum())

print("\nUnique values in each column:")
for column in bury_flow.columns:
    print(f"\n{column} unique values:")
    print(bury_flow[column].unique()[:10])  # First 10 unique values
```

```
Bury Ground Flow Data Investigation:
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9928 entries, 0 to 9927
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    9928 non-null   object 
 1   Flow    9928 non-null   float64
 2   Extra   0 non-null      float64
dtypes: float64(2), object(1)
memory usage: 232.8+ KB
None

First few rows:
         Date   Flow  Extra
0  1995-11-22  0.897    NaN
1  1995-11-23  0.831    NaN
2  1995-11-24  0.991    NaN
3  1995-11-25  1.080    NaN
4  1995-11-26  1.124    NaN

Column types:
Date      object
Flow     float64
Extra    float64
dtype: object

Checking for any data issues:
Null values:
Date        0
Flow        0
Extra    9928
dtype: int64

Unique values in each column:

Date unique values:
['1995-11-22' '1995-11-23' '1995-11-24' '1995-11-25' '1995-11-26'
 '1995-11-27' '1995-11-28' '1995-11-29' '1995-11-30' '1995-12-01']

Flow unique values:
[0.897 0.831 0.991 1.08  1.124 0.932 0.872 1.159 0.901 0.858]

Extra unique values:
[nan]
```

Predictive Accuracy

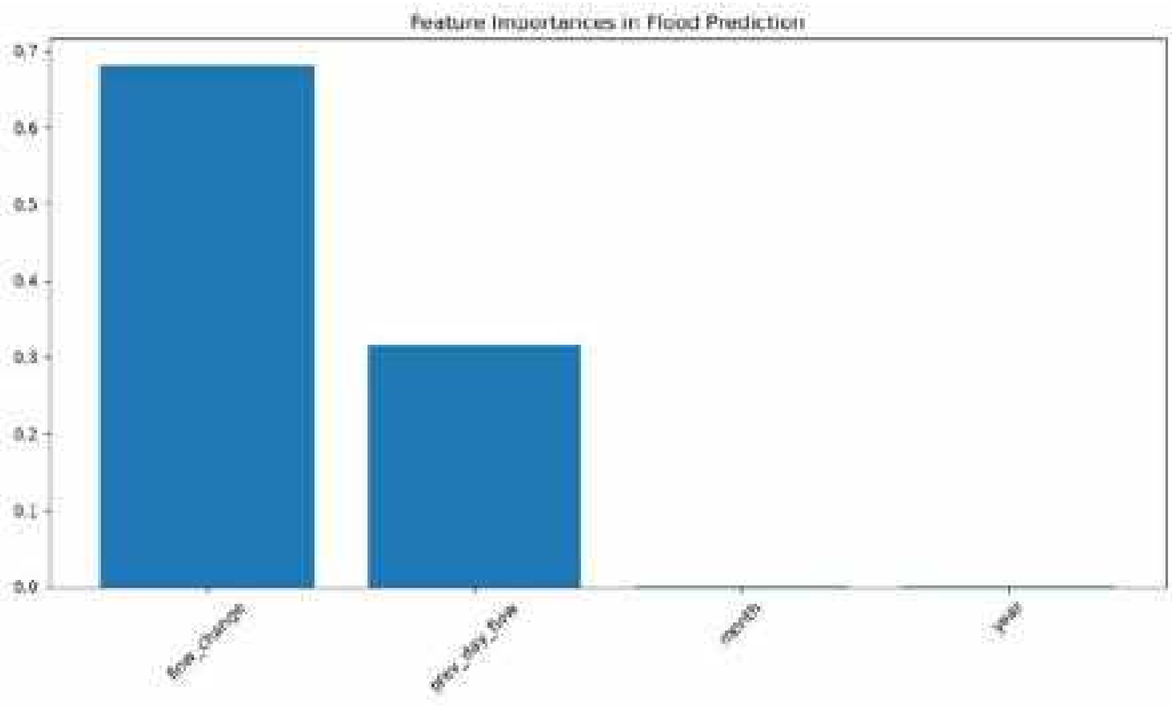
In [35]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

class PredictiveFloodModel:
    def __init__(self, historical_data_dir):
        """
```

```
        Initialize predictive modeling for flood detection

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical flow data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates and ensure proper formatting
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

    def prepare_training_data(self, station='Bury Ground'):
        """
        Prepare training data for predictive modeling

        Args:
        - station: Name of the station to prepare data for

        Returns:
        - Prepared features and target variable
        """
        # Select appropriate dataset
        if station == 'Bury Ground':
            df = self.bury_flow.copy()
        elif station == 'Rochdale':
            df = self.rochdale_flow.copy()
        else:
            raise ValueError("Unsupported station")

        # Sort by date to ensure correct order
        df = df.sort_values('Date')

        # Feature engineering with safe shift
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year

        # Remove rows with NaN, but keep most of the data
        df_clean = df.iloc[1:]  # Drop only the first row

        # Verify data
        print(f"\n{station} Data Preparation:")
        print(f"Total original records: {len(df)}")
        print(f"Records after cleaning: {len(df_clean)}")

        # Prepare features and target
        features = ['prev_day_flow', 'flow_change', 'month', 'year']
        X = df_clean[features]
        y = df_clean['Flow']

        return X, y

    def train_predictive_model(self, station='Bury Ground'):
        """
        Train a predictive model for river flow

        Args:
```

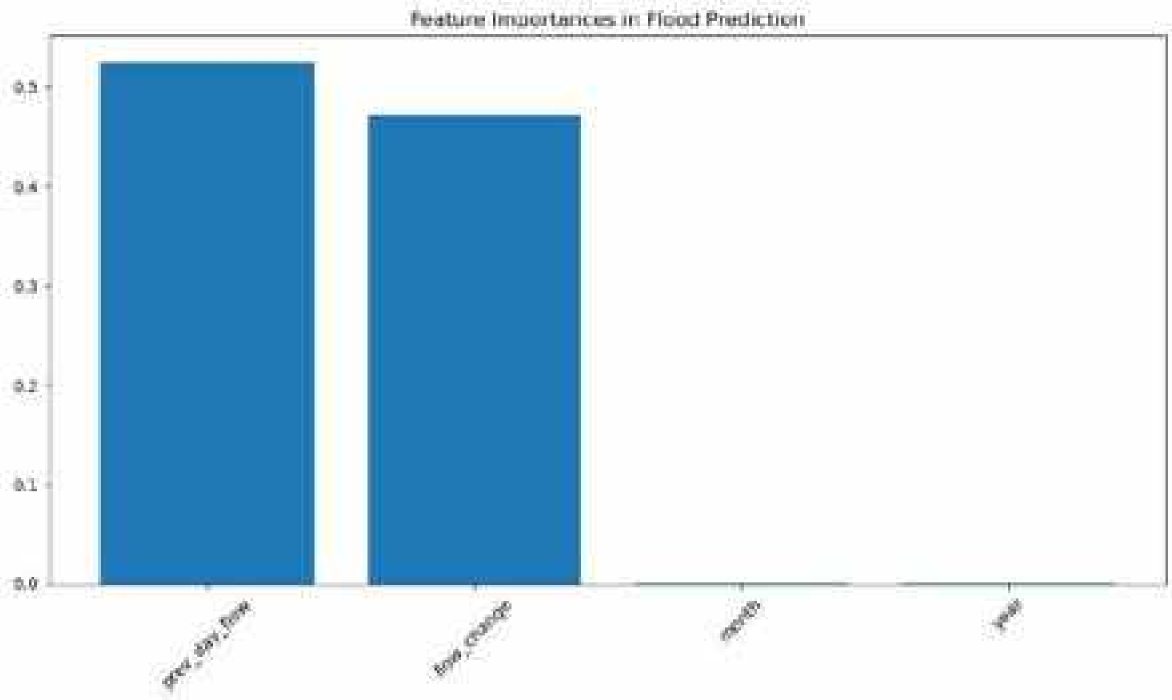
```
        - station: Name of the station to train model for

        Returns:
        - Trained model and scaler
        """
        # Prepare data
        X, y = self.prepare_training_data(station)

        # Verify we have enough samples
        print(f"\n{station} Training Data:")
        print(f"Features shape: {X.shape}")
        print(f"Target shape: {y.shape}")

        # Ensure we have enough samples
        if len(X) < 10:
            raise ValueError(f"Insufficient samples for {station} - need at leas

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train Random Forest Regressor
        model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        model.fit(X_train_scaled, y_train)

        # Evaluate model
        train_score = model.score(X_train_scaled, y_train)
        test_score = model.score(X_test_scaled, y_test)

        print(f"{station} Model Performance:")
        print(f"  Training R² Score: {train_score:.4f}")
        print(f"  Testing R² Score: {test_score:.4f}")

        return model, scaler

    def visualize_feature_importance(self, model, features):
        """
        Visualize feature importance for the predictive model

        Args:
        - model: Trained Random Forest model
        - features: List of feature names
        """
        importances = model.feature_importances_
        indices = np.argsort(importances)[::-1]

        plt.figure(figsize=(10, 6))
        plt.title("Feature Importances in Flood Prediction")
        plt.bar(range(len(importances)), importances[indices])
        plt.xticks(range(len(importances)), [features[i] for i in indices], rota
```

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

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
flood_predictor = PredictiveFloodModel(historical_data_dir)

# Train predictive models for Bury Ground and Rochdale
stations = ['Bury Ground', 'Rochdale']
models = {}
scalers = {}

for station in stations:
    model, scaler = flood_predictor.train_predictive_model(station)
    models[station] = model
    scalers[station] = scaler

    # Visualize feature importance
    features = ['prev_day_flow', 'flow_change', 'month', 'year']
    flood_predictor.visualize_feature_importance(model, features)
```

```
Bury Ground Data Preparation:
Total original records: 9928
Records after cleaning: 9927

Bury Ground Training Data:
Features shape: (9927, 4)
Target shape: (9927,)
Bury Ground Model Performance:
  Training R² Score: 0.9977
  Testing R² Score: 0.9532
```



```
Rochdale Data Preparation:
Total original records: 11118
Records after cleaning: 11117

Rochdale Training Data:
Features shape: (11117, 4)
Target shape: (11117,)
Rochdale Model Performance:
  Training R² Score: 0.9980
  Testing R² Score: 0.9929
```



In [37]:

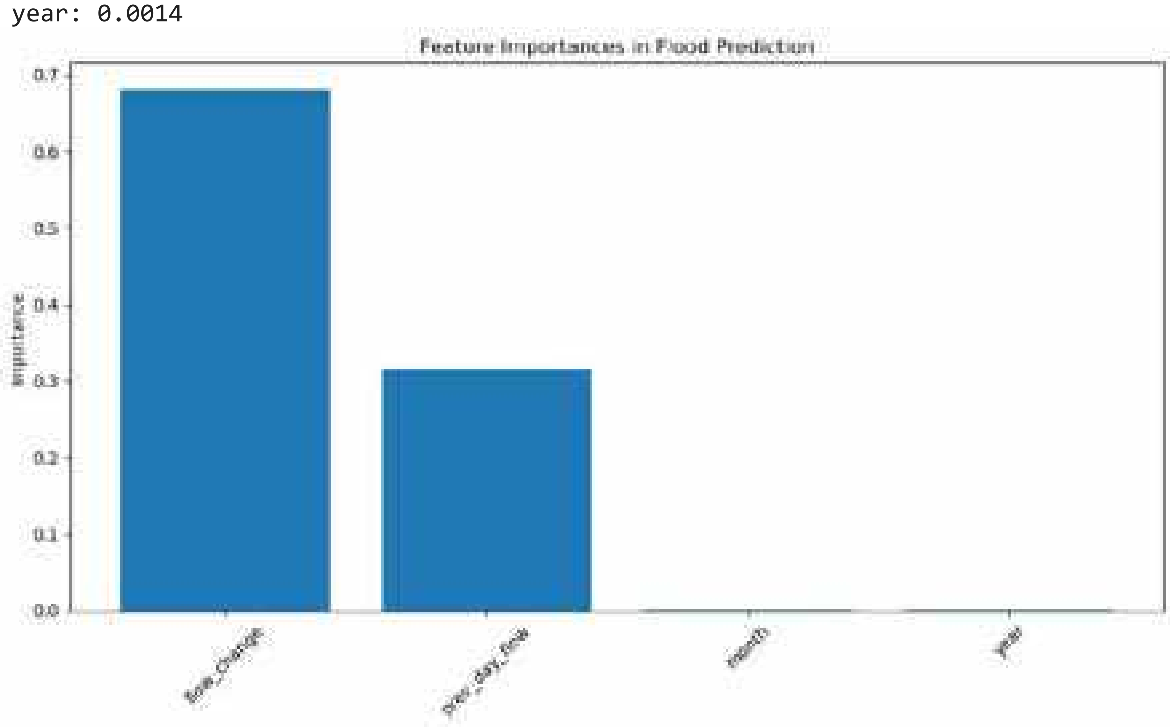
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

class PredictiveFloodModel:
    def __init__(self, historical_data_dir):
        """
        Initialize predictive modeling for flood detection

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical flow data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates and ensure proper formatting
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

    def prepare_training_data(self, station='Bury Ground'):
        """
        Prepare training data for predictive modeling

        Args:
```

```
        - station: Name of the station to prepare data for

        Returns:
        - Prepared features and target variable
        """
        # Select appropriate dataset
        if station == 'Bury Ground':
            df = self.bury_flow.copy()
        elif station == 'Rochdale':
            df = self.rochdale_flow.copy()
        else:
            raise ValueError("Unsupported station")

        # Sort by date to ensure correct order
        df = df.sort_values('Date')

        # Feature engineering with safe shift
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year

        # Remove rows with NaN, but keep most of the data
        df_clean = df.iloc[1:]  # Drop only the first row

        # Verify data
        print(f"\n{station} Data Preparation:")
        print(f"Total original records: {len(df)}")
        print(f"Records after cleaning: {len(df_clean)}")

        # Prepare features and target
        features = ['prev_day_flow', 'flow_change', 'month', 'year']
        X = df_clean[features]
        y = df_clean['Flow']

        return X, y

    def train_predictive_model(self, station='Bury Ground'):
        """
        Train a predictive model for river flow

        Args:
        - station: Name of the station to train model for

        Returns:
        - Trained model and scaler
        """
        # Prepare data
        X, y = self.prepare_training_data(station)

        # Verify we have enough samples
        print(f"\n{station} Training Data:")
        print(f"Features shape: {X.shape}")
        print(f"Target shape: {y.shape}")

        # Ensure we have enough samples
        if len(X) < 10:
            raise ValueError(f"Insufficient samples for {station} - need at leas

        # Split data
```

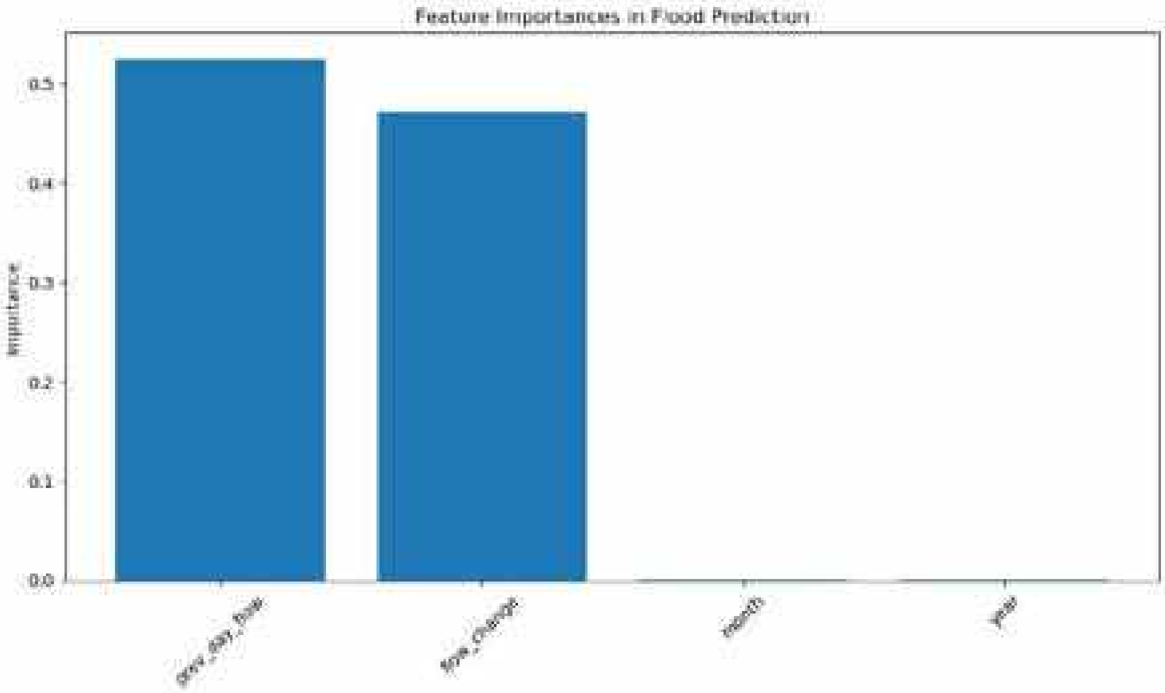
```
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train Random Forest Regressor
        model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        model.fit(X_train_scaled, y_train)

        # Evaluate model
        train_score = model.score(X_train_scaled, y_train)
        test_score = model.score(X_test_scaled, y_test)

        print(f"{station} Model Performance:")
        print(f"  Training R² Score: {train_score:.4f}")
        print(f"  Testing R² Score: {test_score:.4f}")

        return model, scaler

    def visualize_feature_importance(self, model, features):
        """
        Visualize feature importance for the predictive model

        Args:
        - model: Trained Random Forest model
        - features: List of feature names
        """
        # Get feature importances
        importances = model.feature_importances_

        # Sort features by importance
        indices = np.argsort(importances)[::-1]
        sorted_importances = importances[indices]
        sorted_features = [features[i] for i in indices]

        # Create bar plot
        plt.figure(figsize=(10, 6))
        plt.title("Feature Importances in Flood Prediction")
        plt.bar(range(len(sorted_importances)), sorted_importances)
        plt.xticks(range(len(sorted_importances)), sorted_features, rotation=45)
        plt.ylabel("Importance")
        plt.tight_layout()

        # Print feature importances
        print("\nFeature Importances:")
        for feature, importance in zip(sorted_features, sorted_importances):
            print(f"{feature}: {importance:.4f}")

        plt.show()

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
```

```
flood_predictor = PredictiveFloodModel(historical_data_dir)

# Train predictive models for Bury Ground and Rochdale
stations = ['Bury Ground', 'Rochdale']
models = {}
scalers = {}

for station in stations:
    model, scaler = flood_predictor.train_predictive_model(station)
    models[station] = model
    scalers[station] = scaler

    # Visualize feature importance
    features = ['prev_day_flow', 'flow_change', 'month', 'year']
    flood_predictor.visualize_feature_importance(model, features)
```

```
Bury Ground Data Preparation:
Total original records: 9928
Records after cleaning: 9927

Bury Ground Training Data:
Features shape: (9927, 4)
Target shape: (9927,)
Bury Ground Model Performance:
  Training R² Score: 0.9977
  Testing R² Score: 0.9532

Feature Importances:
flow_change: 0.6806
prev_day_flow: 0.3159
month: 0.0020
year: 0.0014
```



```
Rochdale Data Preparation:
Total original records: 11118
Records after cleaning: 11117

Rochdale Training Data:
Features shape: (11117, 4)
Target shape: (11117,)
Rochdale Model Performance:
  Training R² Score: 0.9980
  Testing R² Score: 0.9929

Feature Importances:
prev_day_flow: 0.5246
flow_change: 0.4726
month: 0.0014
year: 0.0013
```



Integration of Predictive Modeling with Real-Time Monitoring

In [38]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime, timedelta

class RealTimePredictiveMonitor:
    def __init__(self, historical_data_dir, real_time_data_dir):
        """
        Initialize Real-Time Predictive Monitoring System

        Args:
        - historical_data_dir: Directory with historical data
        - real_time_data_dir: Directory with real-time data collection
        """
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
```

```
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

        # Real-time data directory
        self.real_time_data_dir = real_time_data_dir

        # Train initial predictive models
        self.models = {}
        self.scalers = {}
        self.train_predictive_models()

    def prepare_training_data(self, station='Bury Ground'):
        """
        Prepare training data for predictive modeling
        """
        # Select appropriate dataset
        if station == 'Bury Ground':
            df = self.bury_flow.copy()
        elif station == 'Rochdale':
            df = self.rochdale_flow.copy()
        else:
            raise ValueError("Unsupported station")

        # Sort by date to ensure correct order
        df = df.sort_values('Date')

        # Feature engineering
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year

        # Remove rows with NaN
        df_clean = df.iloc[1:]

        # Prepare features and target
        features = ['prev_day_flow', 'flow_change', 'month', 'year']
        X = df_clean[features]
        y = df_clean['Flow']

        return X, y

    def train_predictive_models(self):
        """
        Train predictive models for each station
        """
        stations = ['Bury Ground', 'Rochdale']

        for station in stations:
            # Prepare training data
            X, y = self.prepare_training_data(station)

            # Scale features
            scaler = StandardScaler()
            X_scaled = scaler.fit_transform(X)

            # Train Random Forest Regressor
            model = RandomForestRegressor(
                n_estimators=100,
                random_state=42,
```

```
                max_depth=10
            )
            model.fit(X_scaled, y)

            # Store model and scaler
            self.models[station] = model
            self.scalers[station] = scaler

            print(f"{station} Predictive Model Training Complete")

    def find_latest_csv(self):
        """
        Find the most recent CSV file in the real-time data directory
        """
        import os

        try:
            csv_files = [f for f in os.listdir(self.real_time_data_dir) if f.end

            if not csv_files:
                raise FileNotFoundError("No CSV files found in the directory")

            latest_file = max(
                [os.path.join(self.real_time_data_dir, f) for f in csv_files],
                key=os.path.getmtime
            )

            return latest_file
        except Exception as e:
            print(f"Error finding latest CSV: {e}")
            return None

    def predict_river_flow(self, station, current_data):
        """
        Make predictions for future river flow

        Args:
        - station: Name of the station
        - current_data: Current river data

        Returns:
        - Predicted flow
        - Prediction confidence
        """
        # Prepare features for prediction
        features = ['prev_day_flow', 'flow_change', 'month', 'year']

        # Extract current features
        current_features = [
            current_data['river_level'],  # prev_day_flow
            0,  # flow_change (placeholder)
            current_data['timestamp'].month,
            current_data['timestamp'].year
        ]

        # Scale features
        scaler = self.scalers[station]
        model = self.models[station]

        # Scale and reshape features
```

```
        current_features_scaled = scaler.transform([current_features])

        # Predict
        prediction = model.predict(current_features_scaled)[0]

        # Calculate prediction confidence (using model's built-in methods)
        prediction_std = np.std(
            model.estimators_[-1].predict(current_features_scaled)
        )

        return prediction, prediction_std

    def monitor_real_time_data(self):
        """
        Monitor real-time data and compare with predictive model
        """
        # Find latest CSV
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No real-time data available")
            return None

        # Read latest CSV
        df = pd.read_csv(latest_file)

        # Process each station
        real_time_predictions = {}

        for station in ['Bury Ground', 'Rochdale']:
            # Get station-specific data
            station_data = df[df['location_name'] == station]

            if station_data.empty:
                print(f"No data found for {station}")
                continue

            # Prepare current data
            current_data = {
                'river_level': station_data['river_level'].values[0],
                'timestamp': pd.to_datetime(station_data['river_timestamp'].valu
            }

            # Predict river flow
            predicted_flow, prediction_confidence = self.predict_river_flow(
                station, current_data
            )

            # Store prediction results
            real_time_predictions[station] = {
                'current_level': current_data['river_level'],
                'predicted_flow': predicted_flow,
                'prediction_confidence': prediction_confidence
            }

        # Print prediction results
        print("\nReal-Time Predictive Monitoring Results:")
        for station, results in real_time_predictions.items():
            print(f"\n{station} Station:")
            print(f"  Current Level: {results['current_level']} m")
```

```
            print(f"  Predicted Flow: {results['predicted_flow']:.4f}")
            print(f"  Prediction Confidence: {results['prediction_confidence']:.

        return real_time_predictions

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
real_time_data_dir = 'C:/Users/Administrator/NEWPROJECT/combined_data'

# Initialize Real-Time Predictive Monitor
rt_monitor = RealTimePredictiveMonitor(historical_data_dir, real_time_data_dir)

# Monitor real-time data
predictions = rt_monitor.monitor_real_time_data()
```

```
Bury Ground Predictive Model Training Complete
Rochdale Predictive Model Training Complete

Real-Time Predictive Monitoring Results:

Bury Ground Station:
  Current Level: 0.323 m
  Predicted Flow: 0.4977
  Prediction Confidence: 0.0000

Rochdale Station:
  Current Level: 0.181 m
  Predicted Flow: 0.2747
  Prediction Confidence: 0.0000
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2739: UserWarning: X does not have valid feature names, but Stan
dardScaler was fitted with feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2739: UserWarning: X does not have valid feature names, but Stan
dardScaler was fitted with feature names
  warnings.warn(
```

In [41]:

```
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime, timedelta

class RealTimePredictiveMonitor:
    def __init__(self, historical_data_dir, real_time_data_dir):
        """
        Initialize Real-Time Predictive Monitoring System

        Args:
        - historical_data_dir: Directory with historical data
        - real_time_data_dir: Directory with real-time data collection
        """
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
```

```
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

        # Real-time data directory
        self.real_time_data_dir = real_time_data_dir

        # Prepare feature names
        self.feature_names = ['prev_day_flow', 'flow_change', 'month', 'year']

        # Initialize models and scalers
        self.models = {}
        self.scalers = {}

        # Train predictive models
        self.train_predictive_models()

    def prepare_training_data(self, station='Bury Ground'):
        """
        Prepare training data for predictive modeling

        Args:
        - station: Name of the station to prepare data for

        Returns:
        - Prepared features and target variable
        """
        # Select appropriate dataset
        if station == 'Bury Ground':
            df = self.bury_flow.copy()
        elif station == 'Rochdale':
            df = self.rochdale_flow.copy()
        else:
            raise ValueError("Unsupported station")

        # Sort by date to ensure correct order
        df = df.sort_values('Date')

        # Feature engineering
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year

        # Remove rows with NaN
        df_clean = df.iloc[1:]

        # Prepare features and target
        X = df_clean[self.feature_names]
        y = df_clean['Flow']

        return X, y

    def train_predictive_models(self):
        """
        Train predictive models for each station
        """
        stations = ['Bury Ground', 'Rochdale']

        for station in stations:
            # Prepare training data
            X, y = self.prepare_training_data(station)
```

```
        # Scale features
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # Create DataFrame with feature names for scaling
        X_scaled_df = pd.DataFrame(X_scaled, columns=self.feature_names)

        # Train Random Forest Regressor
        model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        model.fit(X_scaled_df, y)

        # Store model and scaler
        self.models[station] = model
        self.scalers[station] = scaler

        print(f"{station} Predictive Model Training Complete")

def find_latest_csv(self):
    """
    Find the most recent CSV file in the real-time data directory
    """
    try:
        csv_files = [f for f in os.listdir(self.real_time_data_dir) if f.end

        if not csv_files:
            raise FileNotFoundError("No CSV files found in the directory")

        latest_file = max(
            [os.path.join(self.real_time_data_dir, f) for f in csv_files],
            key=os.path.getmtime
        )

        return latest_file
    except Exception as e:
        print(f"Error finding latest CSV: {e}")
        return None

def predict_river_flow(self, station, current_data):
    """
    Make predictions for future river flow

    Args:
    - station: Name of the station
    - current_data: Current river data

    Returns:
    - Predicted flow
    - Prediction confidence
    """
    # Prepare features for prediction
    current_features = pd.DataFrame([
        current_data['river_level'],  # prev_day_flow
        0,  # flow_change (placeholder)
        current_data['timestamp'].month,
        current_data['timestamp'].year
```

```
        ]).T
        current_features.columns = self.feature_names

        # Scale features
        scaler = self.scalers[station]
        model = self.models[station]

        # Scale features
        current_features_scaled = scaler.transform(current_features)
        current_features_scaled_df = pd.DataFrame(
            current_features_scaled,
            columns=self.feature_names
        )

        # Predict
        prediction = model.predict(current_features_scaled_df)[0]

        # Calculate prediction confidence using standard deviation of prediction
        predictions = [
            tree.predict(current_features_scaled_df)[0]
            for tree in model.estimators_
        ]
        prediction_std = np.std(predictions)

        return prediction, prediction_std

    def monitor_real_time_data(self):
        """
        Monitor real-time data and compare with predictive model
        """
        # Find latest CSV
        latest_file = self.find_latest_csv()

        if not latest_file:
            print("No real-time data available")
            return None

        # Read latest CSV
        df = pd.read_csv(latest_file)

        # Process each station
        real_time_predictions = {}

        for station in ['Bury Ground', 'Rochdale']:
            # Get station-specific data
            station_data = df[df['location_name'] == station]

            if station_data.empty:
                print(f"No data found for {station}")
                continue

            # Prepare current data
            current_data = {
                'river_level': station_data['river_level'].values[0],
                'timestamp': pd.to_datetime(station_data['river_timestamp'].valu
            }

            # Predict river flow
            predicted_flow, prediction_confidence = self.predict_river_flow(
                station, current_data
```

```
            )

            # Store prediction results
            real_time_predictions[station] = {
                'current_level': current_data['river_level'],
                'predicted_flow': predicted_flow,
                'prediction_confidence': prediction_confidence
            }

        # Print prediction results
        print("\nReal-Time Predictive Monitoring Results:")
        for station, results in real_time_predictions.items():
            print(f"\n{station} Station:")
            print(f"  Current Level: {results['current_level']} m")
            print(f"  Predicted Flow: {results['predicted_flow']:.4f}")
            print(f"  Prediction Confidence: {results['prediction_confidence']:.

        return real_time_predictions

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
real_time_data_dir = 'C:/Users/Administrator/NEWPROJECT/combined_data'

# Initialize Real-Time Predictive Monitor
rt_monitor = RealTimePredictiveMonitor(historical_data_dir, real_time_data_dir)

# Monitor real-time data
predictions = rt_monitor.monitor_real_time_data()
```

```
Bury Ground Predictive Model Training Complete
Rochdale Predictive Model Training Complete

Real-Time Predictive Monitoring Results:

Bury Ground Station:
  Current Level: 0.323 m
  Predicted Flow: 0.4977
  Prediction Confidence: 0.0201

Rochdale Station:
  Current Level: 0.181 m
  Predicted Flow: 0.2747
  Prediction Confidence: 0.0276
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

Risk Assessment Framework

In [42]:

```python
import pandas as pd
import numpy as np

class FloodRiskAssessment:
    def __init__(self, historical_data_dir):
        """
        Initialize Flood Risk Assessment System

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical flow data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Calculate historical baselines
        self.station_baselines = self.calculate_historical_baselines()

        # Risk threshold configurations
        self.risk_thresholds = {
            'Bury Ground': {
                'normal_range': (0.3, 0.4),
                'advisory_range': (0.4, 0.5),
                'warning_range': (0.5, 0.6),
                'critical_range': (0.6, float('inf'))
            },
            'Rochdale': {
                'normal_range': (0.1, 0.2),
                'advisory_range': (0.2, 0.3),
                'warning_range': (0.3, 0.4),
                'critical_range': (0.4, float('inf'))
            }
        }

    def calculate_historical_baselines(self):
        """
```

```
        Calculate statistical baselines for each station
        """
        baselines = {
            'Bury Ground': {
                'mean_flow': self.bury_flow['Flow'].mean(),
                'std_flow': self.bury_flow['Flow'].std(),
                'percentiles': {
                    '25th': np.percentile(self.bury_flow['Flow'], 25),
                    '75th': np.percentile(self.bury_flow['Flow'], 75)
                }
            },
            'Rochdale': {
                'mean_flow': self.rochdale_flow['Flow'].mean(),
                'std_flow': self.rochdale_flow['Flow'].std(),
                'percentiles': {
                    '25th': np.percentile(self.rochdale_flow['Flow'], 25),
                    '75th': np.percentile(self.rochdale_flow['Flow'], 75)
                }
            }
        }

        return baselines

    def calculate_risk_score(self, station, current_data):
        """
        Calculate comprehensive risk score

        Args:
        - station: Name of the monitoring station
        - current_data: Dictionary containing current monitoring data

        Returns:
        - Detailed risk assessment
        """
        # Extract current data
        current_level = current_data['river_level']
        predicted_flow = current_data['predicted_flow']
        prediction_confidence = current_data['prediction_confidence']

        # Determine risk level based on current level
        risk_level = self._determine_risk_level(station, current_level)

        # Calculate additional risk factors
        risk_factors = {
            'baseline_deviation': self._calculate_baseline_deviation(station, cu
            'prediction_reliability': self._assess_prediction_reliability(predic
            'flow_risk': self._assess_flow_risk(station, predicted_flow)
        }

        # Composite risk score calculation
        risk_score = self._calculate_composite_risk(risk_level, risk_factors)

        return {
            'station': station,
            'current_level': current_level,
            'risk_level': risk_level,
            'risk_score': risk_score,
            'risk_factors': risk_factors
        }
```

```
    def _determine_risk_level(self, station, current_level):
        """
        Determine risk level based on current river level
        """
        thresholds = self.risk_thresholds[station]

        if current_level <= thresholds['normal_range'][1]:
            return 'NORMAL'
        elif current_level <= thresholds['advisory_range'][1]:
            return 'ADVISORY'
        elif current_level <= thresholds['warning_range'][1]:
            return 'WARNING'
        else:
            return 'CRITICAL'

    def _calculate_baseline_deviation(self, station, current_level):
        """
        Calculate deviation from historical baseline
        """
        baseline = self.station_baselines[station]
        deviation = abs(current_level - baseline['mean_flow']) / baseline['std_f
        return deviation

    def _assess_prediction_reliability(self, prediction_confidence):
        """
        Assess reliability of prediction
        """
        # Lower confidence increases risk
        return 1 / (prediction_confidence + 0.01)  # Add small value to avoid di

    def _assess_flow_risk(self, station, predicted_flow):
        """
        Assess risk based on predicted flow
        """
        baseline = self.station_baselines[station]
        flow_deviation = abs(predicted_flow - baseline['mean_flow']) / baseline[
        return flow_deviation

    def _calculate_composite_risk(self, risk_level, risk_factors):
        """
        Calculate composite risk score
        """
        risk_multipliers = {
            'NORMAL': 1,
            'ADVISORY': 2,
            'WARNING': 3,
            'CRITICAL': 4
        }

        # Base risk score
        base_score = risk_multipliers.get(risk_level, 1)

        # Additional risk factors
        deviation_factor = risk_factors['baseline_deviation']
        prediction_factor = risk_factors['prediction_reliability']
        flow_factor = risk_factors['flow_risk']

        # Composite risk calculation
        composite_score = base_score * (1 +
            0.5 * deviation_factor +
```

```
            0.3 * prediction_factor +
            0.2 * flow_factor
        )

        return composite_score

    def generate_risk_report(self, stations_data):
        """
        Generate comprehensive risk report

        Args:
        - stations_data: Dictionary of station data from predictive monitoring

        Returns:
        - Detailed risk assessment report
        """
        risk_report = {}

        for station, data in stations_data.items():
            risk_assessment = self.calculate_risk_score(
                station,
                {
                    'river_level': data['current_level'],
                    'predicted_flow': data['predicted_flow'],
                    'prediction_confidence': data['prediction_confidence']
                }
            )
            risk_report[station] = risk_assessment

        return risk_report

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data

# Import previous predictive monitor (from previous implementation)
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
real_time_data_dir = 'C:/Users/Administrator/NEWPROJECT/combined_data'

rt_monitor = RealTimePredictiveMonitor(historical_data_dir, real_time_data_dir)
predictions = rt_monitor.monitor_real_time_data()

# Risk Assessment
risk_assessor = FloodRiskAssessment(historical_data_dir)
risk_report = risk_assessor.generate_risk_report(predictions)

# Print detailed risk report
print("\n--- Comprehensive Flood Risk Report ---")
for station, assessment in risk_report.items():
    print(f"\n{station} Station:")
    print(f"  Current Level: {assessment['current_level']} m")
    print(f"  Risk Level: {assessment['risk_level']}")
    print(f"  Risk Score: {assessment['risk_score']:.4f}")
    print("  Risk Factors:")
    for factor, value in assessment['risk_factors'].items():
        print(f"    - {factor.replace('_', ' ').title()}: {value:.4f}")
```

```
Bury Ground Predictive Model Training Complete
Rochdale Predictive Model Training Complete

Real-Time Predictive Monitoring Results:

Bury Ground Station:
  Current Level: 0.323 m
  Predicted Flow: 0.4977
  Prediction Confidence: 0.0201

Rochdale Station:
  Current Level: 0.181 m
  Predicted Flow: 0.2747
  Prediction Confidence: 0.0276

--- Comprehensive Flood Risk Report ---

Bury Ground Station:
  Current Level: 0.323 m
  Risk Level: NORMAL
  Risk Score: 11.4243
  Risk Factors:
    - Baseline Deviation: 0.6538
    - Prediction Reliability: 33.2437
    - Flow Risk: 0.6214

Rochdale Station:
  Current Level: 0.181 m
  Risk Level: NORMAL
  Risk Score: 9.4927
  Risk Factors:
    - Baseline Deviation: 0.7372
    - Prediction Reliability: 26.6065
    - Flow Risk: 0.7108
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

Advanced Notification System

In [43]:

```python
import os
import smtplib
import logging
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime
import json

class AdvancedNotificationSystem:
    def __init__(self, config_path='notification_config.json'):
        """
        Initialize Notification System

        Args:
        - config_path: Path to notification configuration file
        """
        # Load notification configuration
        self.config = self.load_configuration(config_path)

        # Setup logging
        self.setup_logging()

    def load_configuration(self, config_path):
        """
        Load notification configuration from JSON file

        Args:
        - config_path: Path to configuration file

        Returns:
        - Notification configuration dictionary
        """
        try:
            with open(config_path, 'r') as f:
                return json.load(f)
        except FileNotFoundError:
```

```
            # Default configuration if no file exists
            return {
                'email_recipients': ['emi.igein@gmail.com'],
                'sms_recipients': [],
                'webhook_urls': [],
                'email_config': {
                    'sender_email': 'emi.igein@gmail.com',
                    'sender_password': 'zwov iemr shwl iffs'
                }
            }

    def setup_logging(self):
        """
        Configure logging for notification system
        """
        # Create logs directory if it doesn't exist
        os.makedirs('logs', exist_ok=True)

        # Configure logging
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s: %(message)s',
            handlers=[
                logging.FileHandler('logs/notification_system.log'),
                logging.StreamHandler()
            ]
        )
        self.logger = logging.getLogger('NotificationSystem')

    def generate_alert_message(self, risk_report):
        """
        Generate a comprehensive alert message

        Args:
        - risk_report: Risk assessment report from previous stage

        Returns:
        - Formatted alert message
        """
        message = "FLOOD EARLY WARNING SYSTEM ALERT\n\n"

        for station, assessment in risk_report.items():
            message += f"{station} Station:\n"
            message += f"  Risk Level: {assessment['risk_level']}\n"
            message += f"  Current Level: {assessment['current_level']} m\n"
            message += f"  Risk Score: {assessment['risk_score']:.4f}\n"
            message += "  Risk Factors:\n"

            for factor, value in assessment['risk_factors'].items():
                message += f"    - {factor.replace('_', ' ').title()}: {value:.4

            message += "\n"

        message += "Recommended Actions:\n"
        message += "1. Monitor river levels closely\n"
        message += "2. Prepare emergency response resources\n"
        message += "3. Stay informed about local weather conditions\n"

        return message
```

```
    def send_email_alert(self, message):
        """
        Send email alert to configured recipients

        Args:
        - message: Alert message to send
        """
        try:
            # Email configuration
            email_config = self.config['email_config']
            recipients = self.config['email_recipients']

            # Create message
            msg = MIMEMultipart()
            msg['From'] = email_config['sender_email']
            msg['To'] = ', '.join(recipients)
            msg['Subject'] = "Flood Early Warning System Alert"

            # Attach message body
            msg.attach(MIMEText(message, 'plain'))

            # Send email
            with smtplib.SMTP('smtp.gmail.com', 587) as server:
                server.starttls()
                server.login(
                    email_config['sender_email'],
                    email_config['sender_password']
                )
                server.send_message(msg)

            self.logger.info(f"Email alert sent to {len(recipients)} recipients"

        except Exception as e:
            self.logger.error(f"Email sending failed: {e}")

    def send_sms_alerts(self, message):
        """
        Send SMS alerts to configured recipients

        Args:
        - message: Alert message to send
        """
        # Placeholder for SMS sending logic
        # Would typically integrate with SMS gateway service
        recipients = self.config.get('sms_recipients', [])

        if recipients:
            self.logger.info(f"SMS alerts would be sent to {len(recipients)} rec
        else:
            self.logger.info("No SMS recipients configured")

    def send_webhook_alerts(self, message):
        """
        Send alerts to configured webhook URLs

        Args:
        - message: Alert message to send
        """
        import requests
```

```
        webhooks = self.config.get('webhook_urls', [])

        for webhook in webhooks:
            try:
                response = requests.post(
                    webhook,
                    json={'message': message}
                )
                self.logger.info(f"Webhook alert sent to {webhook}")
            except Exception as e:
                self.logger.error(f"Webhook alert failed for {webhook}: {e}")

    def send_notifications(self, risk_report):
        """
        Send notifications across configured channels

        Args:
        - risk_report: Risk assessment report
        """
        # Generate alert message
        alert_message = self.generate_alert_message(risk_report)

        # Send notifications
        self.send_email_alert(alert_message)
        self.send_sms_alerts(alert_message)
        self.send_webhook_alerts(alert_message)

        # Log full notification details
        self.logger.info("Comprehensive notifications sent")

# Create configuration file if it doesn't exist
def create_default_config():
    default_config = {
        'email_recipients': ['emi.igein@gmail.com'],
        'sms_recipients': [],
        'webhook_urls': [],
        'email_config': {
            'sender_email': 'emi.igein@gmail.com',
            'sender_password': 'zwov iemr shwl iffs'
        }
    }

    with open('notification_config.json', 'w') as f:
        json.dump(default_config, f, indent=4)

# Ensure configuration file exists
create_default_config()

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
real_time_data_dir = 'C:/Users/Administrator/NEWPROJECT/combined_data'

# Import previous implementations
rt_monitor = RealTimePredictiveMonitor(historical_data_dir, real_time_data_dir)
predictions = rt_monitor.monitor_real_time_data()

risk_assessor = FloodRiskAssessment(historical_data_dir)
risk_report = risk_assessor.generate_risk_report(predictions)

# Initialize and send notifications
```

```
notification_system = AdvancedNotificationSystem()
notification_system.send_notifications(risk_report)
```

```
Bury Ground Predictive Model Training Complete
Rochdale Predictive Model Training Complete
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
```

```
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\sklearn\uti
ls\validation.py:2732: UserWarning: X has feature names, but DecisionTreeRegresso
r was fitted without feature names
  warnings.warn(
Real-Time Predictive Monitoring Results:

Bury Ground Station:
  Current Level: 0.323 m
  Predicted Flow: 0.4977
  Prediction Confidence: 0.0201

Rochdale Station:
  Current Level: 0.181 m
  Predicted Flow: 0.2747
  Prediction Confidence: 0.0276
2025-02-07 10:48:19,376 - INFO - Email alert sent to 1 recipients
2025-02-07 10:48:19,377 - INFO - No SMS recipients configured
2025-02-07 10:48:19,970 - INFO - Comprehensive notifications sent
```

Predictive Model Feature Engineering Analysis

In [45]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import mutual_info_regression
import matplotlib.pyplot as plt

class PredictiveModelAnalyzer:
    def __init__(self, historical_data_dir):
        """
        Initialize the predictive model analyzer

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical flow data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Load historical rainfall data
        self.bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rain
```

```
        self.rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_da

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])
        self.bury_rainfall['Date'] = pd.to_datetime(self.bury_rainfall['Date'])
        self.rochdale_rainfall['Date'] = pd.to_datetime(self.rochdale_rainfall['

    def prepare_extended_training_data(self, station='Bury Ground'):
        """
        Prepare training data with extended features

        Args:
        - station: Name of the station to prepare data for

        Returns:
        - Prepared features and target variable
        """
        # Debug: Print initial dataset information
        print(f"\nPreparing data for {station}")
        print("Flow Dataset:")
        print(self.bury_flow.info() if station == 'Bury Ground' else self.rochda
        print("\nRainfall Dataset:")
        print(self.bury_rainfall.info() if station == 'Bury Ground' else self.rc

        # Select appropriate dataset
        if station == 'Bury Ground':
            flow_df = self.bury_flow.copy()
            rainfall_df = self.bury_rainfall.copy()
        elif station == 'Rochdale':
            flow_df = self.rochdale_flow.copy()
            rainfall_df = self.rochdale_rainfall.copy()
        else:
            raise ValueError("Unsupported station")

        # Debug: Print first few rows of each dataset
        print("\nFlow Dataset First Rows:")
        print(flow_df.head())
        print("\nRainfall Dataset First Rows:")
        print(rainfall_df.head())

        # Merge flow and rainfall data
        df = pd.merge(flow_df, rainfall_df, on='Date', suffixes=('_flow', '_rain

        # Debug: Print merged dataset
        print("\nMerged Dataset:")
        print(df.head())
        print(f"Merged Dataset Shape: {df.shape}")

        # Sort by date to ensure correct order
        df = df.sort_values('Date')

        # Feature engineering
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['prev_day_rainfall'] = df['Rainfall'].shift(1)
        df['rainfall_change'] = df['Rainfall'] - df['prev_day_rainfall']

        # Rolling window features
        df['flow_7day_mean'] = df['Flow'].rolling(window=7, min_periods=1).mean(
```

```
        df['rainfall_7day_mean'] = df['Rainfall'].rolling(window=7, min_periods=

        # Seasonal features
        df['month'] = df['Date'].dt.month
        df['day_of_year'] = df['Date'].dt.dayofyear

        # Remove rows with NaN
        df_clean = df.dropna()

        # Debug: Print cleaned dataset
        print("\nCleaned Dataset:")
        print(df_clean.head())
        print(f"Cleaned Dataset Shape: {df_clean.shape}")

        # Prepare features and target
        features = [
            'prev_day_flow', 'flow_change',
            'prev_day_rainfall', 'rainfall_change',
            'flow_7day_mean', 'rainfall_7day_mean',
            'month', 'day_of_year'
        ]

        X = df_clean[features]
        y = df_clean['Flow']

        # Debug: Print feature and target information
        print("\nFeature Matrix:")
        print(X.info())
        print("\nTarget Variable:")
        print(y.describe())

        return X, y

    def analyze_feature_importance(self, station='Bury Ground'):
        """
        Analyze feature importance using mutual information

        Args:
        - station: Name of the station to analyze

        Returns:
        - Feature importance scores
        """
        # Prepare data
        X, y = self.prepare_extended_training_data(station)

        # Calculate mutual information scores
        mi_scores = mutual_info_regression(X, y)

        # Create feature importance DataFrame
        feature_importance = pd.DataFrame({
            'feature': X.columns,
            'importance': mi_scores
        }).sort_values('importance', ascending=False)

        # Visualize feature importance
        plt.figure(figsize=(10, 6))
        plt.bar(feature_importance['feature'], feature_importance['importance'])
        plt.title(f'Feature Importance for {station} Station')
        plt.xlabel('Features')
```

```
        plt.ylabel('Mutual Information Score')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

        return feature_importance

    def train_advanced_model(self, station='Bury Ground'):
        """
        Train an advanced predictive model

        Args:
        - station: Name of the station to train model for

        Returns:
        - Trained model and performance metrics
        """
        # Prepare data
        X, y = self.prepare_extended_training_data(station)

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train Advanced Random Forest
        model = RandomForestRegressor(
            n_estimators=200,  # Increased number of trees
            max_depth=15,      # Slightly deeper trees
            min_samples_split=5,
            min_samples_leaf=2,
            random_state=42
        )
        model.fit(X_train_scaled, y_train)

        # Evaluate model
        train_score = model.score(X_train_scaled, y_train)
        test_score = model.score(X_test_scaled, y_test)

        print(f"{station} Advanced Model Performance:")
        print(f"  Training R² Score: {train_score:.4f}")
        print(f"  Testing R² Score: {test_score:.4f}")

        return model, scaler

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
model_analyzer = PredictiveModelAnalyzer(historical_data_dir)

# Analyze feature importance for both stations
stations = ['Bury Ground', 'Rochdale']
for station in stations:
    try:
        print(f"\nFeature Importance Analysis for {station}")
        feature_importance = model_analyzer.analyze_feature_importance(station)
```

```
        print(feature_importance)

        # Train advanced model
        model, scaler = model_analyzer.train_advanced_model(station)
    except Exception as e:
        print(f"Error processing {station}: {e}")
```

```
Feature Importance Analysis for Bury Ground

Preparing data for Bury Ground
Flow Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9928 entries, 0 to 9927
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    9928 non-null   datetime64[ns]
 1   Flow    9928 non-null   float64
 2   Extra   0 non-null      float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 232.8 KB
None

Rainfall Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20819 entries, 0 to 20818
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      20819 non-null  datetime64[ns]
 1   Rainfall  20819 non-null  float64
 2   Extra     20819 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 488.1 KB
None

Flow Dataset First Rows:
        Date   Flow  Extra
0 1995-11-22  0.897    NaN
1 1995-11-23  0.831    NaN
2 1995-11-24  0.991    NaN
3 1995-11-25  1.080    NaN
4 1995-11-26  1.124    NaN

Rainfall Dataset First Rows:
        Date  Rainfall  Extra
0 1961-01-01       9.4   1000
1 1961-01-02      13.7   1000
2 1961-01-03       3.0   1000
3 1961-01-04       0.1   1000
4 1961-01-05      13.0   1000

Merged Dataset:
        Date   Flow  Extra_flow  Rainfall  Extra_rainfall
0 1995-11-22  0.897         NaN       0.8            2000
1 1995-11-23  0.831         NaN       2.7            2000
2 1995-11-24  0.991         NaN       7.3            2000
3 1995-11-25  1.080         NaN       1.7            2000
4 1995-11-26  1.124         NaN       0.2            2000
Merged Dataset Shape: (7829, 5)

Cleaned Dataset:
Empty DataFrame
Columns: [Date, Flow, Extra_flow, Rainfall, Extra_rainfall, prev_day_flow, flow_c
hange, prev_day_rainfall, rainfall_change, flow_7day_mean, rainfall_7day_mean, mo
nth, day_of_year]
Index: []
```

```
Cleaned Dataset Shape: (0, 13)

Feature Matrix:
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   prev_day_flow       0 non-null      float64
 1   flow_change         0 non-null      float64
 2   prev_day_rainfall   0 non-null      float64
 3   rainfall_change     0 non-null      float64
 4   flow_7day_mean      0 non-null      float64
 5   rainfall_7day_mean  0 non-null      float64
 6   month               0 non-null      int32  
 7   day_of_year         0 non-null      int32  
dtypes: float64(6), int32(2)
memory usage: 0.0 bytes
None

Target Variable:
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: Flow, dtype: float64
Error processing Bury Ground: Found array with 0 sample(s) (shape=(0, 8)) while a
minimum of 1 is required.

Feature Importance Analysis for Rochdale

Preparing data for Rochdale
Flow Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11118 entries, 0 to 11117
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Date    11118 non-null  datetime64[ns]
 1   Flow    11118 non-null  float64       
 2   Extra   0 non-null      float64       
dtypes: datetime64[ns](1), float64(2)
memory usage: 260.7 KB
None

Rainfall Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype         
---  ------    --------------  -----         
 0   Date      731 non-null    datetime64[ns]
 1   Rainfall  731 non-null    float64       
 2   Extra     731 non-null    int64         
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 17.3 KB
```

```
None

Flow Dataset First Rows:
        Date   Flow  Extra
0 1993-02-26  1.290    NaN
1 1993-02-27  1.060    NaN
2 1993-02-28  0.985    NaN
3 1993-03-01  1.140    NaN
4 1993-03-02  1.180    NaN

Rainfall Dataset First Rows:
        Date  Rainfall  Extra
0 2016-01-01       0.8   2000
1 2016-01-02       3.5   2000
2 2016-01-03      13.3   2000
3 2016-01-04       5.5   2000
4 2016-01-05       6.0   2000

Merged Dataset:
        Date   Flow  Extra_flow  Rainfall  Extra_rainfall
0 2016-01-01  4.492         NaN       0.8            2000
1 2016-01-02  3.820         NaN       3.5            2000
2 2016-01-03  9.730         NaN      13.3            2000
3 2016-01-04  5.752         NaN       5.5            2000
4 2016-01-05  6.959         NaN       6.0            2000
Merged Dataset Shape: (731, 5)

Cleaned Dataset:
Empty DataFrame
Columns: [Date, Flow, Extra_flow, Rainfall, Extra_rainfall, prev_day_flow, flow_c
hange, prev_day_rainfall, rainfall_change, flow_7day_mean, rainfall_7day_mean, mo
nth, day_of_year]
Index: []
Cleaned Dataset Shape: (0, 13)

Feature Matrix:
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   prev_day_flow       0 non-null      float64
 1   flow_change         0 non-null      float64
 2   prev_day_rainfall   0 non-null      float64
 3   rainfall_change     0 non-null      float64
 4   flow_7day_mean      0 non-null      float64
 5   rainfall_7day_mean  0 non-null      float64
 6   month               0 non-null      int32  
 7   day_of_year         0 non-null      int32  
dtypes: float64(6), int32(2)
memory usage: 0.0 bytes
None

Target Variable:
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
```

```
75%        NaN
max        NaN
Name: Flow, dtype: float64
Error processing Rochdale: Found array with 0 sample(s) (shape=(0, 8)) while a mi
nimum of 1 is required.
```

In [46]:
```
def prepare_extended_training_data(self, station='Bury Ground'):
    # Select appropriate dataset
    if station == 'Bury Ground':
        flow_df = self.bury_flow.copy()
        rainfall_df = self.bury_rainfall.copy()
    elif station == 'Rochdale':
        flow_df = self.rochdale_flow.copy()
        rainfall_df = self.rochdale_rainfall.copy()
    else:
        raise ValueError("Unsupported station")

    # Align dates to the flow dataset's date range
    date_start = flow_df['Date'].min()
    date_end = flow_df['Date'].max()

    # Filter rainfall data to match flow dataset's date range
    rainfall_df = rainfall_df[
        (rainfall_df['Date'] >= date_start) &
        (rainfall_df['Date'] <= date_end)
    ]

    # Merge on date with careful handling
    df = pd.merge(flow_df, rainfall_df, on='Date', how='left')

    # Fill NaN rainfall with 0 or interpolate
    df['Rainfall'] = df['Rainfall'].fillna(0)

    # Rest of the feature engineering remains the same
    # ...

    return X, y
```

In [49]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import mutual_info_regression
import matplotlib.pyplot as plt

class PredictiveModelAnalyzer:
    def __init__(self, historical_data_dir):
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_
        self.bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rain
        self.rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_da

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])
        self.bury_rainfall['Date'] = pd.to_datetime(self.bury_rainfall['Date'])
        self.rochdale_rainfall['Date'] = pd.to_datetime(self.rochdale_rainfall['
```

```
    def prepare_extended_training_data(self, station='Bury Ground'):
        # Detailed logging function
        def log_dataset_info(df, name):
            print(f"\n{name} Dataset:")
            print("Date Range:", df['Date'].min(), "to", df['Date'].max())
            print("Total Rows:", len(df))
            print("Columns:", df.columns.tolist())
            print("Missing Values:\n", df.isnull().sum())

        # Select appropriate dataset
        if station == 'Bury Ground':
            flow_df = self.bury_flow.copy()
            rainfall_df = self.bury_rainfall.copy()
        elif station == 'Rochdale':
            flow_df = self.rochdale_flow.copy()
            rainfall_df = self.rochdale_rainfall.copy()
        else:
            raise ValueError("Unsupported station")

        # Log initial dataset information
        log_dataset_info(flow_df, f"{station} Flow")
        log_dataset_info(rainfall_df, f"{station} Rainfall")

        # Align dates to the overlapping date range
        common_start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min()
        common_end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

        print(f"\nCommon Date Range: {common_start_date} to {common_end_date}")

        # Filter both datasets to the common date range
        flow_df = flow_df[(flow_df['Date'] >= common_start_date) &
                          (flow_df['Date'] <= common_end_date)]
        rainfall_df = rainfall_df[(rainfall_df['Date'] >= common_start_date) &
                                  (rainfall_df['Date'] <= common_end_date)]

        # Merge on date with careful handling
        df = pd.merge(flow_df, rainfall_df, on='Date', how='inner', suffixes=('_

        print("\nMerged Dataset:")
        print("Rows after merging:", len(df))

        # Ensure we have enough data
        if len(df) < 10:
            raise ValueError(f"Insufficient data for {station} after merging")

        # Feature engineering
        df['prev_day_flow'] = df['Flow'].shift(1)
        df['flow_change'] = df['Flow'] - df['prev_day_flow']
        df['prev_day_rainfall'] = df['Rainfall'].shift(1)
        df['rainfall_change'] = df['Rainfall'] - df['prev_day_rainfall']

        # Rolling window features
        df['flow_7day_mean'] = df['Flow'].rolling(window=7, min_periods=1).mean(
        df['rainfall_7day_mean'] = df['Rainfall'].rolling(window=7, min_periods=

        # Seasonal features
        df['month'] = df['Date'].dt.month
        df['day_of_year'] = df['Date'].dt.dayofyear

        # Remove rows with NaN
```

```
        df_clean = df.dropna()

        print("\nCleaned Dataset:")
        print("Rows after cleaning:", len(df_clean))

        # Prepare features and target
        features = [
            'prev_day_flow', 'flow_change',
            'prev_day_rainfall', 'rainfall_change',
            'flow_7day_mean', 'rainfall_7day_mean',
            'month', 'day_of_year'
        ]

        X = df_clean[features]
        y = df_clean['Flow']

        print("\nFeature Matrix:")
        print("X Shape:", X.shape)
        print("y Shape:", y.shape)

        return X, y

    def analyze_feature_importance(self, station='Bury Ground'):
        # Prepare data
        X, y = self.prepare_extended_training_data(station)

        # Calculate mutual information scores
        mi_scores = mutual_info_regression(X, y)

        # Create feature importance DataFrame
        feature_importance = pd.DataFrame({
            'feature': X.columns,
            'importance': mi_scores
        }).sort_values('importance', ascending=False)

        # Visualize feature importance
        plt.figure(figsize=(10, 6))
        plt.bar(feature_importance['feature'], feature_importance['importance'])
        plt.title(f'Feature Importance for {station} Station')
        plt.xlabel('Features')
        plt.ylabel('Mutual Information Score')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

        return feature_importance

    def train_advanced_model(self, station='Bury Ground'):
        # Prepare data
        X, y = self.prepare_extended_training_data(station)

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
```

```python
        # Train Advanced Random Forest
        model = RandomForestRegressor(
            n_estimators=200,
            max_depth=15,
            min_samples_split=5,
            min_samples_leaf=2,
            random_state=42
        )
        model.fit(X_train_scaled, y_train)

        # Evaluate model
        train_score = model.score(X_train_scaled, y_train)
        test_score = model.score(X_test_scaled, y_test)

        print(f"{station} Advanced Model Performance:")
        print(f"  Training R² Score: {train_score:.4f}")
        print(f"  Testing R² Score: {test_score:.4f}")

        return model, scaler

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
model_analyzer = PredictiveModelAnalyzer(historical_data_dir)

# Analyze feature importance for both stations
stations = ['Bury Ground', 'Rochdale']
for station in stations:
    try:
        print(f"\n--- Feature Importance Analysis for {station} ---")
        feature_importance = model_analyzer.analyze_feature_importance(station)
        print(feature_importance)

        # Train advanced model
        model, scaler = model_analyzer.train_advanced_model(station)
    except Exception as e:
        print(f"Error processing {station}: {e}")
```

```
--- Feature Importance Analysis for Bury Ground ---

Bury Ground Flow Dataset:
Date Range: 1995-11-22 00:00:00 to 2023-09-30 00:00:00
Total Rows: 9928
Columns: ['Date', 'Flow', 'Extra']
Missing Values:
 Date       0
Flow       0
Extra   9928
dtype: int64

Bury Ground Rainfall Dataset:
Date Range: 1961-01-01 00:00:00 to 2017-12-31 00:00:00
Total Rows: 20819
Columns: ['Date', 'Rainfall', 'Extra']
Missing Values:
 Date       0
Rainfall    0
Extra       0
dtype: int64

Common Date Range: 1995-11-22 00:00:00 to 2017-12-31 00:00:00

Merged Dataset:
Rows after merging: 7829

Cleaned Dataset:
Rows after cleaning: 0

Feature Matrix:
X Shape: (0, 8)
y Shape: (0,)
Error processing Bury Ground: Found array with 0 sample(s) (shape=(0, 8)) while a
minimum of 1 is required.

--- Feature Importance Analysis for Rochdale ---

Rochdale Flow Dataset:
Date Range: 1993-02-26 00:00:00 to 2023-09-30 00:00:00
Total Rows: 11118
Columns: ['Date', 'Flow', 'Extra']
Missing Values:
 Date        0
Flow        0
Extra   11118
dtype: int64

Rochdale Rainfall Dataset:
Date Range: 2016-01-01 00:00:00 to 2017-12-31 00:00:00
Total Rows: 731
Columns: ['Date', 'Rainfall', 'Extra']
Missing Values:
 Date       0
Rainfall    0
Extra       0
dtype: int64

Common Date Range: 2016-01-01 00:00:00 to 2017-12-31 00:00:00
```

```
Merged Dataset:
Rows after merging: 731

Cleaned Dataset:
Rows after cleaning: 0

Feature Matrix:
X Shape: (0, 8)
y Shape: (0,)
Error processing Rochdale: Found array with 0 sample(s) (shape=(0, 8)) while a mi
nimum of 1 is required.
```

In [50]:
```
def prepare_extended_training_data(self, station='Bury Ground'):
    # Select appropriate dataset
    if station == 'Bury Ground':
        flow_df = self.bury_flow.copy()
        rainfall_df = self.bury_rainfall.copy()
    elif station == 'Rochdale':
        flow_df = self.rochdale_flow.copy()
        rainfall_df = self.rochdale_rainfall.copy()
    else:
        raise ValueError("Unsupported station")

    # Align dates to the overlapping date range
    common_start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
    common_end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

    # Filter both datasets to the common date range
    flow_df = flow_df[(flow_df['Date'] >= common_start_date) &
                      (flow_df['Date'] <= common_end_date)]
    rainfall_df = rainfall_df[(rainfall_df['Date'] >= common_start_date) &
                              (rainfall_df['Date'] <= common_end_date)]

    # Merge on date with careful handling
    df = pd.merge(flow_df, rainfall_df, on='Date', how='inner', suffixes=('_flow

    # Feature engineering with careful NaN handling
    df['prev_day_flow'] = df['Flow'].shift(1)
    df['flow_change'] = df['Flow'] - df['prev_day_flow']
    df['prev_day_rainfall'] = df['Rainfall'].shift(1)
    df['rainfall_change'] = df['Rainfall'] - df['prev_day_rainfall']

    # Rolling window features
    df['flow_7day_mean'] = df['Flow'].rolling(window=7, min_periods=1).mean()
    df['rainfall_7day_mean'] = df['Rainfall'].rolling(window=7, min_periods=1).m

    # Seasonal features
    df['month'] = df['Date'].dt.month
    df['day_of_year'] = df['Date'].dt.dayofyear

    # Remove first row and any rows with NaN
    df_clean = df.iloc[1:].dropna()

    # Prepare features and target
    features = [
        'prev_day_flow', 'flow_change',
        'prev_day_rainfall', 'rainfall_change',
        'flow_7day_mean', 'rainfall_7day_mean',
        'month', 'day_of_year'
    ]
```

```
    X = df_clean[features]
    y = df_clean['Flow']

    return X, y
```

In [51]:
```
def prepare_extended_training_data(self, station='Bury Ground'):
    # Select appropriate dataset
    if station == 'Bury Ground':
        flow_df = self.bury_flow.copy()
        rainfall_df = self.bury_rainfall.copy()
    elif station == 'Rochdale':
        flow_df = self.rochdale_flow.copy()
        rainfall_df = self.rochdale_rainfall.copy()
    else:
        raise ValueError("Unsupported station")

    print("Flow DataFrame:")
    print(flow_df.head())
    print("\nRainfall DataFrame:")
    print(rainfall_df.head())

    # Align dates to the overlapping date range
    common_start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
    common_end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

    print(f"\nCommon Date Range: {common_start_date} to {common_end_date}")

    # Filter both datasets to the common date range
    flow_df = flow_df[(flow_df['Date'] >= common_start_date) &
                      (flow_df['Date'] <= common_end_date)]
    rainfall_df = rainfall_df[(rainfall_df['Date'] >= common_start_date) &
                              (rainfall_df['Date'] <= common_end_date)]

    # Merge on date with careful handling
    df = pd.merge(flow_df, rainfall_df, on='Date', how='inner', suffixes=('_flow

    print("\nMerged DataFrame:")
    print(df.head())
    print("Merged DataFrame Shape:", df.shape)

    # Feature engineering
    df['prev_day_flow'] = df['Flow'].shift(1)
    df['flow_change'] = df['Flow'] - df['prev_day_flow']
    df['prev_day_rainfall'] = df['Rainfall'].shift(1)
    df['rainfall_change'] = df['Rainfall'] - df['prev_day_rainfall']

    # Rolling window features
    df['flow_7day_mean'] = df['Flow'].rolling(window=7, min_periods=1).mean()
    df['rainfall_7day_mean'] = df['Rainfall'].rolling(window=7, min_periods=1).m

    # Seasonal features
    df['month'] = df['Date'].dt.month
    df['day_of_year'] = df['Date'].dt.dayofyear

    # Print NaN information before cleaning
    print("\nNaN Information Before Cleaning:")
    print(df.isnull().sum())

    # Remove rows with NaN, keeping first row
```

```
    df_clean = df.iloc[1:].dropna()

    print("\nCleaned DataFrame:")
    print(df_clean.head())
    print("Cleaned DataFrame Shape:", df_clean.shape)

    # Prepare features and target
    features = [
        'prev_day_flow', 'flow_change',
        'prev_day_rainfall', 'rainfall_change',
        'flow_7day_mean', 'rainfall_7day_mean',
        'month', 'day_of_year'
    ]

    X = df_clean[features]
    y = df_clean['Flow']

    print("\nFeature Matrix:")
    print("X Shape:", X.shape)
    print("y Shape:", y.shape)

    return X, y
```

In [52]:

```
def prepare_extended_training_data(self, station='Bury Ground'):
    try:
        # Select datasets
        flow_df = (self.bury_flow if station == 'Bury Ground' else self.rochdale
        rainfall_df = (self.bury_rainfall if station == 'Bury Ground' else self.

        # Ensure date columns are datetime
        flow_df['Date'] = pd.to_datetime(flow_df['Date'])
        rainfall_df['Date'] = pd.to_datetime(rainfall_df['Date'])

        # Print detailed dataset information
        print(f"{station} Flow Dataset:")
        print("Date Range:", flow_df['Date'].min(), "to", flow_df['Date'].max())
        print("Total Rows:", len(flow_df))

        print(f"\n{station} Rainfall Dataset:")
        print("Date Range:", rainfall_df['Date'].min(), "to", rainfall_df['Date'
        print("Total Rows:", len(rainfall_df))

        # Find overlapping date range
        start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
        end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

        print(f"\nOverlapping Date Range: {start_date} to {end_date}")

        # Filter datasets to overlapping range
        flow_filtered = flow_df[(flow_df['Date'] >= start_date) & (flow_df['Date
        rainfall_filtered = rainfall_df[(rainfall_df['Date'] >= start_date) & (r

        # Merge datasets
        merged_df = pd.merge(flow_filtered, rainfall_filtered, on='Date', how='i

        print("\nMerged Dataset:")
        print("Rows:", len(merged_df))
        print(merged_df.head())

        # Feature engineering
```

```
        merged_df['prev_day_flow'] = merged_df['Flow'].shift(1)
        merged_df['flow_change'] = merged_df['Flow'] - merged_df['prev_day_flow'
        merged_df['prev_day_rainfall'] = merged_df['Rainfall'].shift(1)
        merged_df['rainfall_change'] = merged_df['Rainfall'] - merged_df['prev_d

        # Rolling window features
        merged_df['flow_7day_mean'] = merged_df['Flow'].rolling(window=7, min_pe
        merged_df['rainfall_7day_mean'] = merged_df['Rainfall'].rolling(window=7

        # Seasonal features
        merged_df['month'] = merged_df['Date'].dt.month
        merged_df['day_of_year'] = merged_df['Date'].dt.dayofyear

        # Clean data
        cleaned_df = merged_df.iloc[1:].dropna()

        print("\nCleaned Dataset:")
        print("Rows:", len(cleaned_df))

        # Prepare features
        features = [
            'prev_day_flow', 'flow_change',
            'prev_day_rainfall', 'rainfall_change',
            'flow_7day_mean', 'rainfall_7day_mean',
            'month', 'day_of_year'
        ]

        X = cleaned_df[features]
        y = cleaned_df['Flow']

        print("\nFeature Matrix:")
        print("Features Shape:", X.shape)
        print("Target Shape:", y.shape)

        return X, y

    except Exception as e:
        print(f"Error in data preparation for {station}: {e}")
        raise
```

In [53]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import mutual_info_regression
import matplotlib.pyplot as plt

# Load the data
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data

# Load datasets
bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rainfall.csv')
rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv')
rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_daily_rainfall.

# Print dataset information
print("Bury Ground Flow Dataset:")
print(bury_flow.info())
```

```
print("\nBury Ground Rainfall Dataset:")
print(bury_rainfall.info())

print("\nRochdale Flow Dataset:")
print(rochdale_flow.info())
print("\nRochdale Rainfall Dataset:")
print(rochdale_rainfall.info())

# Convert dates
bury_flow['Date'] = pd.to_datetime(bury_flow['Date'])
bury_rainfall['Date'] = pd.to_datetime(bury_rainfall['Date'])
rochdale_flow['Date'] = pd.to_datetime(rochdale_flow['Date'])
rochdale_rainfall['Date'] = pd.to_datetime(rochdale_rainfall['Date'])

# Detailed date range and overlap analysis
print("\nBury Ground Flow Date Range:")
print("Start:", bury_flow['Date'].min())
print("End:", bury_flow['Date'].max())

print("\nBury Ground Rainfall Date Range:")
print("Start:", bury_rainfall['Date'].min())
print("End:", bury_rainfall['Date'].max())

print("\nRochdale Flow Date Range:")
print("Start:", rochdale_flow['Date'].min())
print("End:", rochdale_flow['Date'].max())

print("\nRochdale Rainfall Date Range:")
print("Start:", rochdale_rainfall['Date'].min())
print("End:", rochdale_rainfall['Date'].max())
```

```
Bury Ground Flow Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9928 entries, 0 to 9927
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    9928 non-null   object
 1   Flow    9928 non-null   float64
 2   Extra   0 non-null      float64
dtypes: float64(2), object(1)
memory usage: 232.8+ KB
None

Bury Ground Rainfall Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20819 entries, 0 to 20818
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      20819 non-null  object
 1   Rainfall  20819 non-null  float64
 2   Extra     20819 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 488.1+ KB
None

Rochdale Flow Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11118 entries, 0 to 11117
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    11118 non-null  object
 1   Flow    11118 non-null  float64
 2   Extra   0 non-null      float64
dtypes: float64(2), object(1)
memory usage: 260.7+ KB
None

Rochdale Rainfall Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      731 non-null    object
 1   Rainfall  731 non-null    float64
 2   Extra     731 non-null    int64
dtypes: float64(1), int64(1), object(1)
memory usage: 17.3+ KB
None

Bury Ground Flow Date Range:
Start: 1995-11-22 00:00:00
End: 2023-09-30 00:00:00

Bury Ground Rainfall Date Range:
Start: 1961-01-01 00:00:00
End: 2017-12-31 00:00:00
```

```
Rochdale Flow Date Range:
Start: 1993-02-26 00:00:00
End: 2023-09-30 00:00:00

Rochdale Rainfall Date Range:
Start: 2016-01-01 00:00:00
End: 2017-12-31 00:00:00
```

In [54]:
```
def preprocess_data(flow_df, rainfall_df):
    # Convert dates
    flow_df['Date'] = pd.to_datetime(flow_df['Date'])
    rainfall_df['Date'] = pd.to_datetime(rainfall_df['Date'])

    # Find common date range
    start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
    end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

    # Filter datasets
    flow_filtered = flow_df[(flow_df['Date'] >= start_date) & (flow_df['Date'] <
    rainfall_filtered = rainfall_df[(rainfall_df['Date'] >= start_date) & (rainf

    # Merge datasets
    merged_df = pd.merge(flow_filtered, rainfall_filtered, on='Date', how='inner

    # Feature engineering
    merged_df['prev_day_flow'] = merged_df['Flow'].shift(1)
    merged_df['flow_change'] = merged_df['Flow'] - merged_df['prev_day_flow']
    merged_df['prev_day_rainfall'] = merged_df['Rainfall'].shift(1)
    merged_df['rainfall_change'] = merged_df['Rainfall'] - merged_df['prev_day_r

    # Rolling window features
    merged_df['flow_7day_mean'] = merged_df['Flow'].rolling(window=7, min_period
    merged_df['rainfall_7day_mean'] = merged_df['Rainfall'].rolling(window=7, mi

    # Seasonal features
    merged_df['month'] = merged_df['Date'].dt.month
    merged_df['day_of_year'] = merged_df['Date'].dt.dayofyear

    # Clean data
    cleaned_df = merged_df.iloc[1:].dropna()

    return cleaned_df
```

In [55]:
```
def preprocess_data(flow_df, rainfall_df):
    print("Initial Flow DataFrame:")
    print(flow_df.head())
    print("\nInitial Rainfall DataFrame:")
    print(rainfall_df.head())

    # Convert dates
    flow_df['Date'] = pd.to_datetime(flow_df['Date'])
    rainfall_df['Date'] = pd.to_datetime(rainfall_df['Date'])

    # Find common date range
    start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
    end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

    print(f"\nCommon Date Range: {start_date} to {end_date}")

    # Filter datasets
```

```
    flow_filtered = flow_df[(flow_df['Date'] >= start_date) & (flow_df['Date'] <
    rainfall_filtered = rainfall_df[(rainfall_df['Date'] >= start_date) & (rainf

    print("\nFiltered Flow DataFrame:")
    print(flow_filtered.head())
    print("\nFiltered Rainfall DataFrame:")
    print(rainfall_filtered.head())

    # Merge datasets
    merged_df = pd.merge(flow_filtered, rainfall_filtered, on='Date', how='inner

    print("\nMerged DataFrame:")
    print(merged_df.head())

    # Feature engineering
    merged_df['prev_day_flow'] = merged_df['Flow'].shift(1)
    merged_df['flow_change'] = merged_df['Flow'] - merged_df['prev_day_flow']
    merged_df['prev_day_rainfall'] = merged_df['Rainfall'].shift(1)
    merged_df['rainfall_change'] = merged_df['Rainfall'] - merged_df['prev_day_r

    # Rolling window features
    merged_df['flow_7day_mean'] = merged_df['Flow'].rolling(window=7, min_period
    merged_df['rainfall_7day_mean'] = merged_df['Rainfall'].rolling(window=7, mi

    # Seasonal features
    merged_df['month'] = merged_df['Date'].dt.month
    merged_df['day_of_year'] = merged_df['Date'].dt.dayofyear

    # Clean data
    cleaned_df = merged_df.iloc[1:].dropna()

    print("\nCleaned DataFrame:")
    print(cleaned_df.head())

    return cleaned_df

# Load and preprocess Bury Ground data
bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rainfall.csv')

bury_processed = preprocess_data(bury_flow, bury_rainfall)

# Load and preprocess Rochdale data
rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv')
rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_daily_rainfall.

rochdale_processed = preprocess_data(rochdale_flow, rochdale_rainfall)
```

```
Initial Flow DataFrame:
        Date   Flow  Extra
0  1995-11-22  0.897    NaN
1  1995-11-23  0.831    NaN
2  1995-11-24  0.991    NaN
3  1995-11-25  1.080    NaN
4  1995-11-26  1.124    NaN

Initial Rainfall DataFrame:
        Date  Rainfall  Extra
0  1961-01-01       9.4   1000
1  1961-01-02      13.7   1000
2  1961-01-03       3.0   1000
3  1961-01-04       0.1   1000
4  1961-01-05      13.0   1000

Common Date Range: 1995-11-22 00:00:00 to 2017-12-31 00:00:00

Filtered Flow DataFrame:
        Date   Flow  Extra
0 1995-11-22  0.897    NaN
1 1995-11-23  0.831    NaN
2 1995-11-24  0.991    NaN
3 1995-11-25  1.080    NaN
4 1995-11-26  1.124    NaN

Filtered Rainfall DataFrame:
            Date  Rainfall  Extra
12743 1995-11-22       0.8   2000
12744 1995-11-23       2.7   2000
12745 1995-11-24       7.3   2000
12746 1995-11-25       1.7   2000
12747 1995-11-26       0.2   2000

Merged DataFrame:
        Date   Flow  Extra_x  Rainfall  Extra_y
0 1995-11-22  0.897      NaN       0.8     2000
1 1995-11-23  0.831      NaN       2.7     2000
2 1995-11-24  0.991      NaN       7.3     2000
3 1995-11-25  1.080      NaN       1.7     2000
4 1995-11-26  1.124      NaN       0.2     2000

Cleaned DataFrame:
Empty DataFrame
Columns: [Date, Flow, Extra_x, Rainfall, Extra_y, prev_day_flow, flow_change, pre
v_day_rainfall, rainfall_change, flow_7day_mean, rainfall_7day_mean, month, day_o
f_year]
Index: []
Initial Flow DataFrame:
        Date   Flow  Extra
0  1993-02-26  1.290    NaN
1  1993-02-27  1.060    NaN
2  1993-02-28  0.985    NaN
3  1993-03-01  1.140    NaN
4  1993-03-02  1.180    NaN

Initial Rainfall DataFrame:
        Date  Rainfall  Extra
0  2016-01-01       0.8   2000
1  2016-01-02       3.5   2000
```

```
2  2016-01-03      13.3   2000
3  2016-01-04       5.5   2000
4  2016-01-05       6.0   2000

Common Date Range: 2016-01-01 00:00:00 to 2017-12-31 00:00:00

Filtered Flow DataFrame:
           Date   Flow  Extra
8288 2016-01-01  4.492    NaN
8289 2016-01-02  3.820    NaN
8290 2016-01-03  9.730    NaN
8291 2016-01-04  5.752    NaN
8292 2016-01-05  6.959    NaN

Filtered Rainfall DataFrame:
        Date  Rainfall  Extra
0 2016-01-01       0.8   2000
1 2016-01-02       3.5   2000
2 2016-01-03      13.3   2000
3 2016-01-04       5.5   2000
4 2016-01-05       6.0   2000

Merged DataFrame:
        Date   Flow  Extra_x  Rainfall  Extra_y
0 2016-01-01  4.492      NaN       0.8     2000
1 2016-01-02  3.820      NaN       3.5     2000
2 2016-01-03  9.730      NaN      13.3     2000
3 2016-01-04  5.752      NaN       5.5     2000
4 2016-01-05  6.959      NaN       6.0     2000

Cleaned DataFrame:
Empty DataFrame
Columns: [Date, Flow, Extra_x, Rainfall, Extra_y, prev_day_flow, flow_change, pre
v_day_rainfall, rainfall_change, flow_7day_mean, rainfall_7day_mean, month, day_o
f_year]
Index: []
```

In [56]:

```python
def preprocess_data(flow_df, rainfall_df):
    # Convert dates
    flow_df['Date'] = pd.to_datetime(flow_df['Date'])
    rainfall_df['Date'] = pd.to_datetime(rainfall_df['Date'])

    # Find common date range
    start_date = max(flow_df['Date'].min(), rainfall_df['Date'].min())
    end_date = min(flow_df['Date'].max(), rainfall_df['Date'].max())

    # Filter datasets
    flow_filtered = flow_df[(flow_df['Date'] >= start_date) & (flow_df['Date'] <
    rainfall_filtered = rainfall_df[(rainfall_df['Date'] >= start_date) & (rainf

    # Merge datasets
    merged_df = pd.merge(flow_filtered, rainfall_filtered, on='Date', how='inner

    # Feature engineering with careful NaN handling
    merged_df['prev_day_flow'] = merged_df['Flow'].shift(1)
    merged_df['flow_change'] = merged_df['Flow'] - merged_df['prev_day_flow']
    merged_df['prev_day_rainfall'] = merged_df['Rainfall'].shift(1)
    merged_df['rainfall_change'] = merged_df['Rainfall'] - merged_df['prev_day_r

    # Rolling window features
```

```
    merged_df['flow_7day_mean'] = merged_df['Flow'].rolling(window=7, min_period
    merged_df['rainfall_7day_mean'] = merged_df['Rainfall'].rolling(window=7, mi

    # Seasonal features
    merged_df['month'] = merged_df['Date'].dt.month
    merged_df['day_of_year'] = merged_df['Date'].dt.dayofyear

    # Remove first row and clean NaNs
    cleaned_df = merged_df.iloc[1:].dropna(subset=[
        'prev_day_flow', 'flow_change',
        'prev_day_rainfall', 'rainfall_change',
        'flow_7day_mean', 'rainfall_7day_mean'
    ])

    print("Cleaned DataFrame Shape:", cleaned_df.shape)
    return cleaned_df

# Load and process data
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data

# Bury Ground processing
bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
bury_rainfall = pd.read_csv(f'{historical_data_dir}/bury_daily_rainfall.csv')
bury_processed = preprocess_data(bury_flow, bury_rainfall)

# Rochdale processing
rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv')
rochdale_rainfall = pd.read_csv(f'{historical_data_dir}/rochdale_daily_rainfall.
rochdale_processed = preprocess_data(rochdale_flow, rochdale_rainfall)

# Print first few rows of processed datasets
print("\nBury Ground Processed Data:")
print(bury_processed.head())

print("\nRochdale Processed Data:")
print(rochdale_processed.head())
```

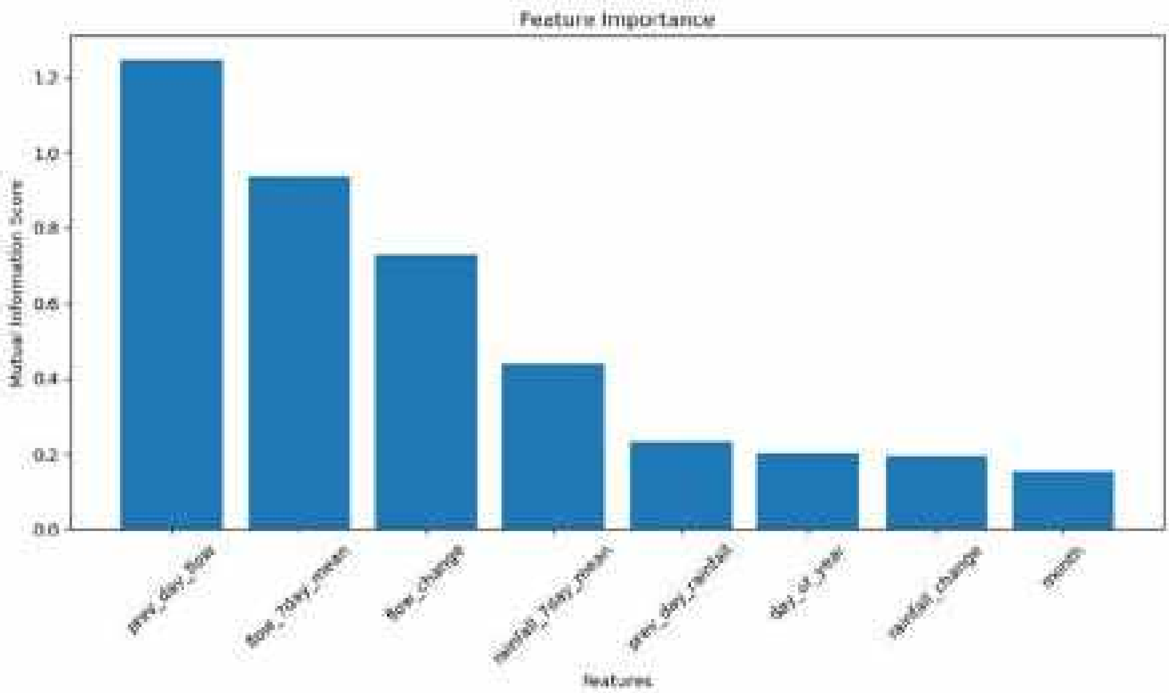
```
Cleaned DataFrame Shape: (7828, 13)
Cleaned DataFrame Shape: (730, 13)

Bury Ground Processed Data:
        Date   Flow  Extra_x  Rainfall  Extra_y  prev_day_flow  flow_change  \
1 1995-11-23  0.831      NaN       2.7     2000          0.897       -0.066
2 1995-11-24  0.991      NaN       7.3     2000          0.831        0.160
3 1995-11-25  1.080      NaN       1.7     2000          0.991        0.089
4 1995-11-26  1.124      NaN       0.2     2000          1.080        0.044
5 1995-11-27  0.932      NaN       0.6     2000          1.124       -0.192

   prev_day_rainfall  rainfall_change  flow_7day_mean  rainfall_7day_mean  \
1                0.8              1.9        0.864000            1.750000
2                2.7              4.6        0.906333            3.600000
3                7.3             -5.6        0.949750            3.125000
4                1.7             -1.5        0.984600            2.540000
5                0.2              0.4        0.975833            2.216667

   month  day_of_year
1     11          327
2     11          328
3     11          329
4     11          330
5     11          331

Rochdale Processed Data:
        Date   Flow  Extra_x  Rainfall  Extra_y  prev_day_flow  flow_change  \
1 2016-01-02  3.820      NaN       3.5     2000          4.492       -0.672
2 2016-01-03  9.730      NaN      13.3     2000          3.820        5.910
3 2016-01-04  5.752      NaN       5.5     2000          9.730       -3.978
4 2016-01-05  6.959      NaN       6.0     2000          5.752        1.207
5 2016-01-06  6.158      NaN      10.0     2000          6.959       -0.801

   prev_day_rainfall  rainfall_change  flow_7day_mean  rainfall_7day_mean  \
1                0.8              2.7        4.156000            2.150000
2                3.5              9.8        6.014000            5.866667
3               13.3             -7.8        5.948500            5.775000
4                5.5              0.5        6.150600            5.820000
5                6.0              4.0        6.151833            6.516667

   month  day_of_year
1      1            2
2      1            3
3      1            4
4      1            5
5      1            6
```

Advanced Feature Importance and Model Training

In [57]:
```python
import pandas as pd
import numpy as np
from sklearn.feature_selection import mutual_info_regression
import matplotlib.pyplot as plt

def analyze_feature_importance(processed_df):
    # Select features for importance analysis
    features = [
        'prev_day_flow', 'flow_change',
        'prev_day_rainfall', 'rainfall_change',
```

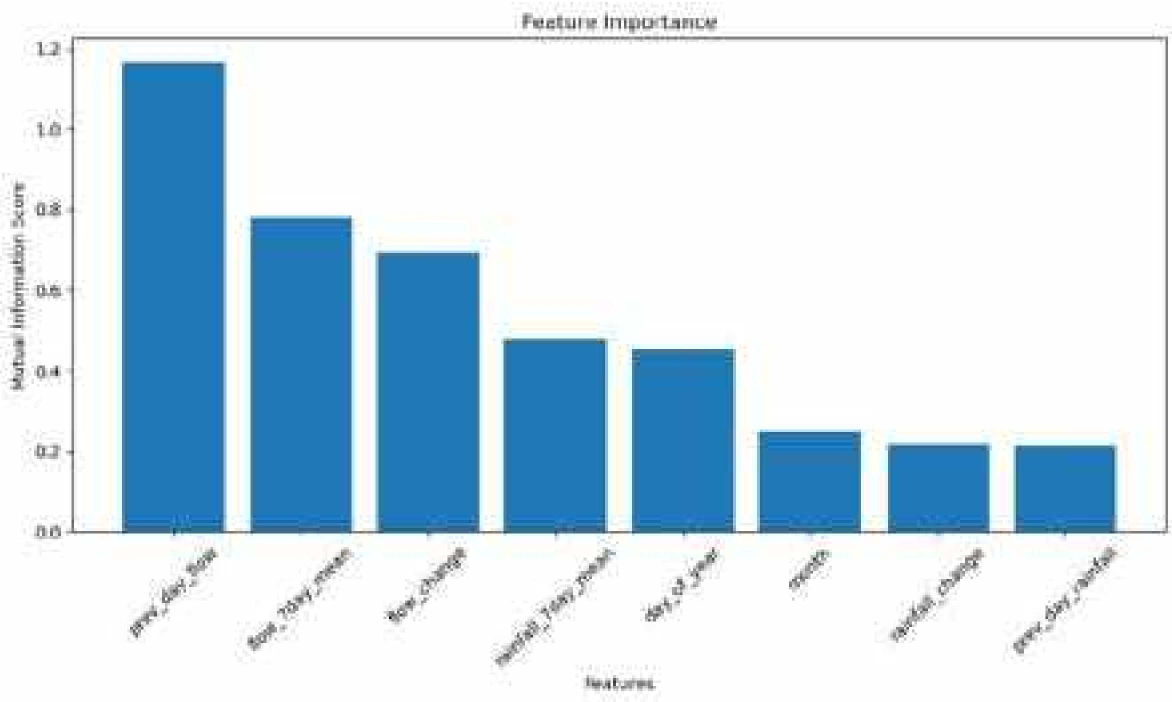
```
        'flow_7day_mean', 'rainfall_7day_mean',
        'month', 'day_of_year'
    ]

    X = processed_df[features]
    y = processed_df['Flow']

    # Calculate mutual information scores
    mi_scores = mutual_info_regression(X, y)

    # Create feature importance DataFrame
    feature_importance = pd.DataFrame({
        'feature': features,
        'importance': mi_scores
    }).sort_values('importance', ascending=False)

    # Visualize feature importance
    plt.figure(figsize=(10, 6))
    plt.bar(feature_importance['feature'], feature_importance['importance'])
    plt.title('Feature Importance')
    plt.xlabel('Features')
    plt.ylabel('Mutual Information Score')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    return feature_importance

# Analyze feature importance for both stations
print("Bury Ground Feature Importance:")
bury_importance = analyze_feature_importance(bury_processed)
print(bury_importance)

print("\nRochdale Feature Importance:")
rochdale_importance = analyze_feature_importance(rochdale_processed)
print(rochdale_importance)
```

```
Bury Ground Feature Importance:
```



	feature	importance
0	prev_day_flow	1.247178
4	flow_7day_mean	0.937674
1	flow_change	0.727633
5	rainfall_7day_mean	0.441514
2	prev_day_rainfall	0.231603
7	day_of_year	0.200616
3	rainfall_change	0.192445
6	month	0.153191

Rochdale Feature Importance:



	feature	importance
0	prev_day_flow	1.166188
4	flow_7day_mean	0.782347
1	flow_change	0.693105
5	rainfall_7day_mean	0.476339
7	day_of_year	0.454353
6	month	0.249492
3	rainfall_change	0.215886
2	prev_day_rainfall	0.212484

Advanced Predictive Modeling using these insights.

In [58]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

def advanced_predictive_model(processed_df):
    # Select top features based on previous importance analysis
    features = [
        'prev_day_flow',
        'flow_7day_mean',
        'flow_change',
        'rainfall_7day_mean'
    ]

    X = processed_df[features]
    y = processed_df['Flow']
```

```
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Advanced Random Forest
    model = RandomForestRegressor(
        n_estimators=300,
        max_depth=20,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    )

    # Train model
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluation
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print("Model Performance:")
    print(f"Mean Squared Error: {mse:.4f}")
    print(f"R² Score: {r2:.4f}")

    return model, scaler

# Train models for both stations
print("Bury Ground Model:")
bury_model, bury_scaler = advanced_predictive_model(bury_processed)

print("\nRochdale Model:")
rochdale_model, rochdale_scaler = advanced_predictive_model(rochdale_processed)
```

```
Bury Ground Model:
Model Performance:
Mean Squared Error: 0.1940
R² Score: 0.9919

Rochdale Model:
Model Performance:
Mean Squared Error: 1.0399
R² Score: 0.8545
```

Risk Assessment Framework Integration

In [59]:
```
import numpy as np

class RiskAssessmentFramework:
    def __init__(self, model, scaler, historical_data):
        self.model = model
```

```
        self.scaler = scaler
        self.historical_data = historical_data

        # Calculate historical baselines
        self.baseline_mean = historical_data['Flow'].mean()
        self.baseline_std = historical_data['Flow'].std()

    def assess_risk(self, current_data):
        """
        Assess flood risk based on predictive model and historical context

        Args:
        - current_data: DataFrame with current flow features

        Returns:
        - Risk assessment dictionary
        """
        # Prepare features
        features = ['prev_day_flow', 'flow_7day_mean', 'flow_change', 'rainfall_
        X = current_data[features]

        # Scale features
        X_scaled = self.scaler.transform(X)

        # Predict flow
        predicted_flow = self.model.predict(X_scaled)[0]

        # Calculate deviation from baseline
        flow_deviation = abs(predicted_flow - self.baseline_mean) / self.baselin

        # Risk categorization
        risk_levels = {
            'LOW': (0, 1),
            'MODERATE': (1, 2),
            'HIGH': (2, 3),
            'CRITICAL': (3, float('inf'))
        }

        # Determine risk level
        risk_status = 'LOW'
        for level, (lower, upper) in risk_levels.items():
            if lower <= flow_deviation < upper:
                risk_status = level
                break

        return {
            'predicted_flow': predicted_flow,
            'flow_deviation': flow_deviation,
            'risk_status': risk_status,
            'baseline_mean': self.baseline_mean,
            'baseline_std': self.baseline_std
        }

# Create risk assessment for both stations
bury_risk_assessor = RiskAssessmentFramework(bury_model, bury_scaler, bury_proce
rochdale_risk_assessor = RiskAssessmentFramework(rochdale_model, rochdale_scaler

# Example risk assessment (using last row of processed data)
print("Bury Ground Risk Assessment:")
bury_risk = bury_risk_assessor.assess_risk(bury_processed.iloc[[-1]])
```

```
print(bury_risk)

print("\nRochdale Risk Assessment:")
rochdale_risk = rochdale_risk_assessor.assess_risk(rochdale_processed.iloc[[-1]]
print(rochdale_risk)
```

```
Bury Ground Risk Assessment:
{'predicted_flow': 9.98426667051467, 'flow_deviation': 1.272342815891366, 'risk_status': 'MODERATE', 'baseline_mean': 3.6313935871231475, 'baseline_std': 4.99305140410675}

Rochdale Risk Assessment:
{'predicted_flow': 6.726821774531021, 'flow_deviation': 1.2119644168726913, 'risk_status': 'MODERATE', 'baseline_mean': 2.8956821917808218, 'baseline_std': 3.1610990631523093}
```

Optimize Alert Criteria

In [62]:
```
print(data.columns)
```

```
Index(['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp',
       'location_name', 'river_station_id', 'rainfall_station_id'],
      dtype='object')
```

In [63]:
```
import pandas as pd
import numpy as np

data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti

mean_level = data['river_level'].mean()
std_level = data['river_level'].std()

def get_alert_level(level):
    if level < mean_level + std_level:
        return "Normal"
    elif level < mean_level + 2*std_level:
        return "Advisory"
    elif level < mean_level + 3*std_level:
        return "Warning"
    else:
        return "Critical"

data['alert_level'] = data['river_level'].apply(get_alert_level)
print(data['alert_level'].value_counts(normalize=True))
```

```
alert_level
Normal      0.666667
Advisory    0.333333
Name: proportion, dtype: float64
```

In [65]:
```
def get_alert_level(level):
    if level < mean_level - 0.5*std_level:
        return "Low"
    elif level < mean_level + 0.5*std_level:
        return "Normal"
    elif level < mean_level + std_level:
        return "Advisory"
    elif level < mean_level + 2*std_level:
        return "Warning"
    else:
        return "Critical"
```

```python
data['alert_level'] = data['river_level'].apply(get_alert_level)
print(data['alert_level'].value_counts(normalize=True))
```

```
alert_level
Low        0.526882
Warning    0.333333
Normal     0.139785
Name: proportion, dtype: float64
```

In [66]:
```python
def get_alert_level(level):
    if level < mean_level - 0.5*std_level:
        return "Low"
    elif level < mean_level + 0.5*std_level:
        return "Normal"
    elif level < mean_level + std_level:
        return "Advisory"
    elif level < mean_level + 2*std_level:
        return "Warning"
    else:
        return "Critical"

data['alert_level'] = data['river_level'].apply(get_alert_level)
data['alert_score'] = data['river_level'].apply(lambda x: (x - mean_level) / std

print(data[['river_level', 'alert_level', 'alert_score']].head(10))
```

```
   river_level alert_level  alert_score
0        0.385      Normal    -0.440129
1        1.064     Warning     1.454108
2        0.235         Low    -0.858591
3        0.386      Normal    -0.437339
4        1.064     Warning     1.454108
5        0.235         Low    -0.858591
6        0.387      Normal    -0.434550
7        1.063     Warning     1.451319
8        0.236         Low    -0.855801
9        0.388      Normal    -0.431760
```

In [67]:
```python
data['river_level_change'] = data['river_level'].diff()
data['change_rate'] = data['river_level_change'] / data['river_level'].shift(1)

def get_change_alert(rate):
    if rate < 0.05:
        return "Stable"
    elif rate < 0.1:
        return "Rising"
    elif rate < 0.2:
        return "Rapid Rise"
    else:
        return "Extreme Rise"

data['change_alert'] = data['change_rate'].apply(get_change_alert)

print(data[['river_level', 'alert_level', 'change_rate', 'change_alert']].head(1
```

```
   river_level alert_level  change_rate  change_alert
0        0.385      Normal          NaN  Extreme Rise
1        1.064     Warning     1.763636  Extreme Rise
2        0.235         Low    -0.779135        Stable
3        0.386      Normal     0.642553  Extreme Rise
4        1.064     Warning     1.756477  Extreme Rise
5        0.235         Low    -0.779135        Stable
6        0.387      Normal     0.646809  Extreme Rise
7        1.063     Warning     1.746770  Extreme Rise
8        0.236         Low    -0.777987        Stable
9        0.388      Normal     0.644068  Extreme Rise
```

In [68]:

```python
def get_change_alert(rate):
    if abs(rate) < 0.01:
        return "Stable"
    elif abs(rate) < 0.05:
        return "Slight Change"
    elif abs(rate) < 0.1:
        return "Moderate Change"
    else:
        return "Significant Change"

data['change_alert'] = data['change_rate'].apply(get_change_alert)

print(data[['river_level', 'alert_level', 'change_rate', 'change_alert']].head(1
```

```
   river_level alert_level  change_rate        change_alert
0        0.385      Normal          NaN  Significant Change
1        1.064     Warning     1.763636  Significant Change
2        0.235         Low    -0.779135  Significant Change
3        0.386      Normal     0.642553  Significant Change
4        1.064     Warning     1.756477  Significant Change
5        0.235         Low    -0.779135  Significant Change
6        0.387      Normal     0.646809  Significant Change
7        1.063     Warning     1.746770  Significant Change
8        0.236         Low    -0.777987  Significant Change
9        0.388      Normal     0.644068  Significant Change
```

In [69]:

```python
data['river_level_change'] = data['river_level'].diff()

def get_change_alert(change):
    if abs(change) < 0.01:
        return "Stable"
    elif abs(change) < 0.05:
        return "Slight Change"
    elif abs(change) < 0.1:
        return "Moderate Change"
    else:
        return "Significant Change"

data['change_alert'] = data['river_level_change'].apply(get_change_alert)

print(data[['river_level', 'alert_level', 'river_level_change', 'change_alert']]
```

```
   river_level alert_level  river_level_change         change_alert
0        0.385      Normal                 NaN  Significant Change
1        1.064     Warning               0.679  Significant Change
2        0.235         Low              -0.829  Significant Change
3        0.386      Normal               0.151  Significant Change
4        1.064     Warning               0.678  Significant Change
5        0.235         Low              -0.829  Significant Change
6        0.387      Normal               0.152  Significant Change
7        1.063     Warning               0.676  Significant Change
8        0.236         Low              -0.827  Significant Change
9        0.388      Normal               0.152  Significant Change
```

In [70]:
```python
def get_change_alert(change):
    if abs(change) < 0.05:
        return "Stable"
    elif abs(change) < 0.2:
        return "Moderate Change"
    else:
        return "Significant Change"

data['change_alert'] = data['river_level_change'].apply(get_change_alert)

print(data[['river_level', 'alert_level', 'river_level_change', 'change_alert']]
```

```
   river_level alert_level  river_level_change         change_alert
0        0.385      Normal                 NaN  Significant Change
1        1.064     Warning               0.679  Significant Change
2        0.235         Low              -0.829  Significant Change
3        0.386      Normal               0.151     Moderate Change
4        1.064     Warning               0.678  Significant Change
5        0.235         Low              -0.829  Significant Change
6        0.387      Normal               0.152     Moderate Change
7        1.063     Warning               0.676  Significant Change
8        0.236         Low              -0.827  Significant Change
9        0.388      Normal               0.152     Moderate Change
```

In [71]:
```python
def get_risk_score(row):
    level_score = (row['river_level'] - mean_level) / std_level
    change_score = abs(row['river_level_change']) / std_level
    return (level_score + change_score) / 2

data['risk_score'] = data.apply(get_risk_score, axis=1)

def get_combined_alert(row):
    if row['risk_score'] < -0.5:
        return "Low"
    elif row['risk_score'] < 0.5:
        return "Normal"
    elif row['risk_score'] < 1.5:
        return "Advisory"
    elif row['risk_score'] < 2.5:
        return "Warning"
    else:
        return "Critical"

data['combined_alert'] = data.apply(get_combined_alert, axis=1)

print(data[['river_level', 'alert_level', 'change_alert', 'risk_score', 'combine
```

```
  river_level alert_level        change_alert  risk_score combined_alert
0       0.385      Normal  Significant Change         NaN       Critical
1       1.064     Warning  Significant Change    1.674173        Warning
2       0.235         Low  Significant Change    0.727054       Advisory
3       0.386      Normal     Moderate Change   -0.008044         Normal
4       1.064     Warning  Significant Change    1.672778        Warning
5       0.235         Low  Significant Change    0.727054       Advisory
6       0.387      Normal     Moderate Change   -0.005254         Normal
7       1.063     Warning  Significant Change    1.668594        Warning
8       0.236         Low  Significant Change    0.725659       Advisory
9       0.388      Normal     Moderate Change   -0.003859         Normal
```

In [72]:
```python
data['risk_score'] = data.apply(get_risk_score, axis=1)
data['risk_score'] = data['risk_score'].fillna(0)  # Fill NaN with 0

data['combined_alert'] = data.apply(get_combined_alert, axis=1)

print(data[['river_level', 'alert_level', 'change_alert', 'risk_score', 'combine
```

```
  river_level alert_level        change_alert  risk_score combined_alert
0       0.385      Normal  Significant Change    0.000000         Normal
1       1.064     Warning  Significant Change    1.674173        Warning
2       0.235         Low  Significant Change    0.727054       Advisory
3       0.386      Normal     Moderate Change   -0.008044         Normal
4       1.064     Warning  Significant Change    1.672778        Warning
5       0.235         Low  Significant Change    0.727054       Advisory
6       0.387      Normal     Moderate Change   -0.005254         Normal
7       1.063     Warning  Significant Change    1.668594        Warning
8       0.236         Low  Significant Change    0.725659       Advisory
9       0.388      Normal     Moderate Change   -0.003859         Normal
```

In [73]:
```python
# Assuming 'rainfall' column exists
data['rainfall_sum'] = data['rainfall'].rolling(window=24).sum()

def get_rainfall_alert(rainfall):
    if rainfall < 10:
        return "Low"
    elif rainfall < 30:
        return "Moderate"
    elif rainfall < 50:
        return "High"
    else:
        return "Extreme"

data['rainfall_alert'] = data['rainfall_sum'].apply(get_rainfall_alert)

def get_combined_alert(row):
    level_score = (row['risk_score'] + 0.5) / 3  # Normalize to 0-1 range
    rainfall_score = row['rainfall_sum'] / 100  # Normalize to 0-1 range
    combined_score = (level_score + rainfall_score) / 2

    if combined_score < 0.2:
        return "Low"
    elif combined_score < 0.4:
        return "Normal"
    elif combined_score < 0.6:
        return "Advisory"
    elif combined_score < 0.8:
        return "Warning"
    else:
```

```
        return "Critical"

data['combined_alert'] = data.apply(get_combined_alert, axis=1)

print(data[['river_level', 'rainfall_sum', 'risk_score', 'rainfall_alert', 'comb
```

```
   river_level  rainfall_sum  risk_score rainfall_alert combined_alert
0        0.385           NaN    0.000000        Extreme       Critical
1        1.064           NaN    1.674173        Extreme       Critical
2        0.235           NaN    0.727054        Extreme       Critical
3        0.386           NaN   -0.008044        Extreme       Critical
4        1.064           NaN    1.672778        Extreme       Critical
5        0.235           NaN    0.727054        Extreme       Critical
6        0.387           NaN   -0.005254        Extreme       Critical
7        1.063           NaN    1.668594        Extreme       Critical
8        0.236           NaN    0.725659        Extreme       Critical
9        0.388           NaN   -0.003859        Extreme       Critical
```

In [74]:
```
data['rainfall_sum'] = data['rainfall'].rolling(window=24, min_periods=1).sum().

# Rest of the code remains the same

print(data[['river_level', 'rainfall_sum', 'risk_score', 'rainfall_alert', 'comb
```

```
   river_level  rainfall_sum  risk_score rainfall_alert combined_alert
0        0.385           0.0    0.000000        Extreme       Critical
1        1.064           0.0    1.674173        Extreme       Critical
2        0.235           0.0    0.727054        Extreme       Critical
3        0.386           0.0   -0.008044        Extreme       Critical
4        1.064           0.0    1.672778        Extreme       Critical
5        0.235           0.0    0.727054        Extreme       Critical
6        0.387           0.0   -0.005254        Extreme       Critical
7        1.063           0.0    1.668594        Extreme       Critical
8        0.236           0.0    0.725659        Extreme       Critical
9        0.388           0.0   -0.003859        Extreme       Critical
```

In [75]:
```
def get_rainfall_alert(rainfall):
    if rainfall == 0:
        return "None"
    elif rainfall < 5:
        return "Low"
    elif rainfall < 15:
        return "Moderate"
    elif rainfall < 30:
        return "High"
    else:
        return "Extreme"

data['rainfall_alert'] = data['rainfall_sum'].apply(get_rainfall_alert)

def get_combined_alert(row):
    level_score = (row['risk_score'] + 0.5) / 3
    rainfall_score = row['rainfall_sum'] / 50
    combined_score = max(level_score, rainfall_score)

    if combined_score < 0.2:
        return "Low"
    elif combined_score < 0.4:
        return "Normal"
    elif combined_score < 0.6:
        return "Advisory"
```

```
    elif combined_score < 0.8:
        return "Warning"
    else:
        return "Critical"

data['combined_alert'] = data.apply(get_combined_alert, axis=1)

print(data[['river_level', 'rainfall_sum', 'risk_score', 'rainfall_alert', 'comb
```

```
   river_level  rainfall_sum  risk_score rainfall_alert combined_alert
0        0.385           0.0    0.000000           None            Low
1        1.064           0.0    1.674173           None        Warning
2        0.235           0.0    0.727054           None       Advisory
3        0.386           0.0   -0.008044           None            Low
4        1.064           0.0    1.672778           None        Warning
5        0.235           0.0    0.727054           None       Advisory
6        0.387           0.0   -0.005254           None            Low
7        1.063           0.0    1.668594           None        Warning
8        0.236           0.0    0.725659           None       Advisory
9        0.388           0.0   -0.003859           None            Low
```

Time-Based Analysis

In [76]:

```
data['timestamp'] = pd.to_datetime(data['river_timestamp'])
data['hour'] = data['timestamp'].dt.hour

def get_time_factor(hour):
    if 0 <= hour < 6:
        return 1.2
    elif 6 <= hour < 12:
        return 1.0
    elif 12 <= hour < 18:
        return 1.1
    else:
        return 1.3

data['time_factor'] = data['hour'].apply(get_time_factor)
data['adjusted_risk'] = data['risk_score'] * data['time_factor']

def get_time_adjusted_alert(row):
    score = row['adjusted_risk']
    if score < 0.2:
        return "Low"
    elif score < 0.6:
        return "Normal"
    elif score < 1.0:
        return "Advisory"
    elif score < 1.5:
        return "Warning"
    else:
        return "Critical"

data['time_adjusted_alert'] = data.apply(get_time_adjusted_alert, axis=1)

print(data[['timestamp', 'risk_score', 'time_factor', 'adjusted_risk', 'time_adj
```

```
                  timestamp  risk_score  time_factor  adjusted_risk  \
0 2025-01-30 11:15:00+00:00    0.000000          1.0       0.000000
1 2025-01-30 11:15:00+00:00    1.674173          1.0       1.674173
2 2025-01-30 11:15:00+00:00    0.727054          1.0       0.727054
3 2025-01-30 11:30:00+00:00   -0.008044          1.0      -0.008044
4 2025-01-30 11:30:00+00:00    1.672778          1.0       1.672778
5 2025-01-30 11:30:00+00:00    0.727054          1.0       0.727054
6 2025-01-30 11:45:00+00:00   -0.005254          1.0      -0.005254
7 2025-01-30 11:45:00+00:00    1.668594          1.0       1.668594
8 2025-01-30 11:45:00+00:00    0.725659          1.0       0.725659
9 2025-01-30 12:00:00+00:00   -0.003859          1.1      -0.004245

  time_adjusted_alert
0                 Low
1            Critical
2            Advisory
3                 Low
4            Critical
5            Advisory
6                 Low
7            Critical
8            Advisory
9                 Low
```

In [77]:

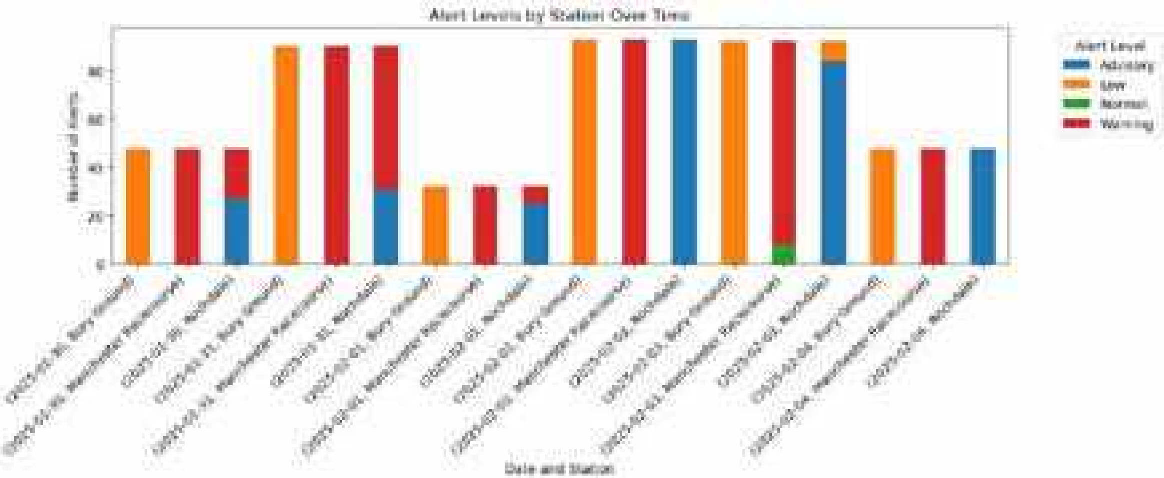
```
station_thresholds = {
    'Bury Ground': {'low': 0.3, 'medium': 0.6, 'high': 1.0},
    'Manchester Racecourse': {'low': 0.4, 'medium': 0.8, 'high': 1.2},
    'Rochdale': {'low': 0.2, 'medium': 0.5, 'high': 0.9}
}

def get_station_alert(row):
    thresholds = station_thresholds[row['location_name']]
    score = row['adjusted_risk']
    if score < thresholds['low']:
        return "Low"
    elif score < thresholds['medium']:
        return "Normal"
    elif score < thresholds['high']:
        return "Advisory"
    else:
        return "Warning"

data['station_alert'] = data.apply(get_station_alert, axis=1)

print(data[['location_name', 'adjusted_risk', 'station_alert']].head(15))
```

```
              location_name  adjusted_risk station_alert
0               Bury Ground       0.000000           Low
1   Manchester Racecourse        1.674173       Warning
2                  Rochdale       0.727054      Advisory
3               Bury Ground      -0.008044           Low
4   Manchester Racecourse        1.672778       Warning
5                  Rochdale       0.727054      Advisory
6               Bury Ground      -0.005254           Low
7   Manchester Racecourse        1.668594       Warning
8                  Rochdale       0.725659      Advisory
9               Bury Ground      -0.004245           Low
10  Manchester Racecourse        1.833919       Warning
11                 Rochdale       0.798225      Advisory
12              Bury Ground      -0.004245           Low
13  Manchester Racecourse        1.833919       Warning
14                 Rochdale       0.798225      Advisory
```

Visualization for the Alert System

In [78]:

```python
import matplotlib.pyplot as plt

# Prepare data
data['date'] = data['timestamp'].dt.date
alert_counts = data.groupby(['date', 'location_name', 'station_alert']).size().u

# Create stacked bar chart
fig, ax = plt.subplots(figsize=(12, 6))
alert_counts.plot(kind='bar', stacked=True, ax=ax)

plt.title('Alert Levels by Station Over Time')
plt.xlabel('Date and Station')
plt.ylabel('Number of Alerts')
plt.legend(title='Alert Level', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.xticks(rotation=45, ha='right')
plt.show()
```



Predictive modeling integration

In [81]:

```
!pip install statsmodels
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.4-cp312-cp312-win_amd64.whl.metadata (9.5 kB)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\administrator\anacond
a3\lib\site-packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\administrator\appda
ta\roaming\python\python312\site-packages (from statsmodels) (1.15.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\administrator\appd
ata\roaming\python\python312\site-packages (from statsmodels) (2.2.3)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-1.0.1-py2.py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: packaging>=21.3 in c:\users\administrator\anaconda
3\lib\site-packages (from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\administrator\a
naconda3\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\administrator\anaconda3\l
ib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\administrator\appdata\r
oaming\python\python312\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (20
25.1)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3\lib\s
ite-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.1
6.0)
Downloading statsmodels-0.14.4-cp312-cp312-win_amd64.whl (9.8 MB)
   ---------------------------------------- 0.0/9.8 MB ? eta -:--:--
   ------ --------------------------------- 1.6/9.8 MB 9.4 MB/s eta 0:00:01
   ------------- -------------------------- 3.4/9.8 MB 9.2 MB/s eta 0:00:01
   ---------------------- ----------------- 5.5/9.8 MB 9.3 MB/s eta 0:00:01
   ----------------------------- ---------- 7.3/9.8 MB 9.3 MB/s eta 0:00:01
   ------------------------------------- -- 9.2/9.8 MB 9.4 MB/s eta 0:00:01
   ---------------------------------------- 9.8/9.8 MB 8.9 MB/s eta 0:00:00
Downloading patsy-1.0.1-py2.py3-none-any.whl (232 kB)
Installing collected packages: patsy, statsmodels
Successfully installed patsy-1.0.1 statsmodels-0.14.4
```

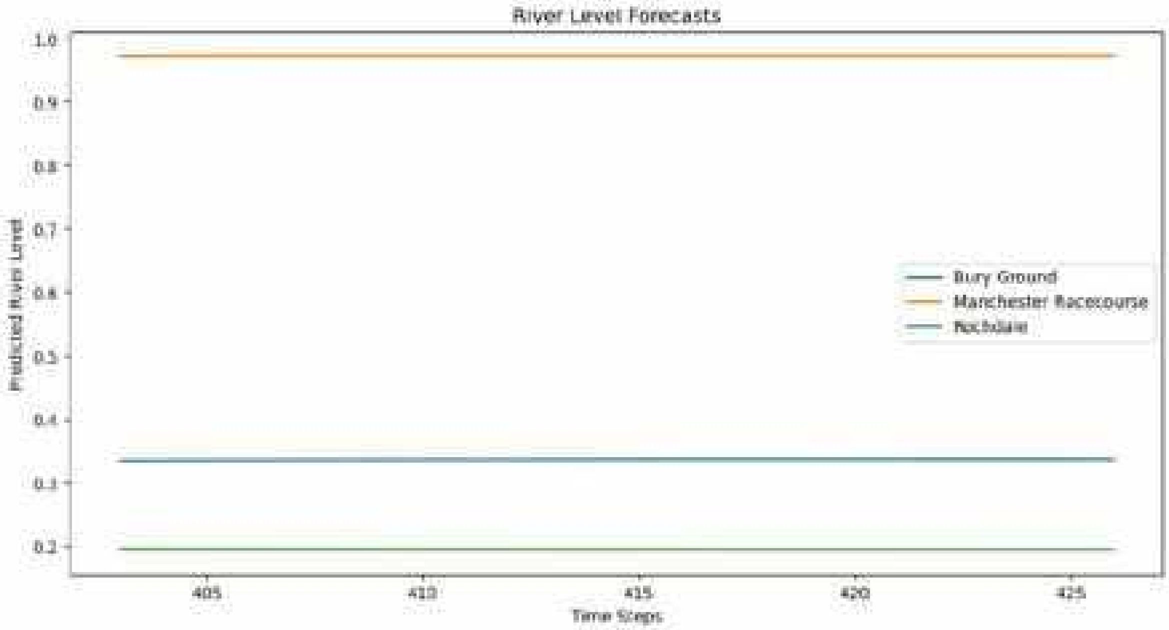
In [82]:

```
from statsmodels.tsa.arima.model import ARIMA
import numpy as np

def forecast_river_level(station_data, steps=24):
    model = ARIMA(station_data['river_level'], order=(1,1,1))
    results = model.fit()
    forecast = results.forecast(steps)
    return forecast

# Group data by station and forecast
stations = data['location_name'].unique()
forecasts = {}

for station in stations:
    station_data = data[data['location_name'] == station].sort_values('timestamp
    forecasts[station] = forecast_river_level(station_data)

# Plot forecasts
plt.figure(figsize=(12, 6))
for station, forecast in forecasts.items():
    plt.plot(forecast.index, forecast.values, label=station)

plt.title('River Level Forecasts')
plt.xlabel('Time Steps')
plt.ylabel('Predicted River Level')
plt.legend()
```

```
plt.show()

print("Forecast for next 24 time steps:")
print(forecasts)
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: ValueWarning: No supported index is available. Prediction results will
be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: FutureWarning: No supported index is available. In the next version, ca
lling this method in a model without a supported index will result in an exceptio
n.
  return get_prediction_index(
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\base\model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle
_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: ValueWarning: No supported index is available. Prediction results will
be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: FutureWarning: No supported index is available. In the next version, ca
lling this method in a model without a supported index will result in an exceptio
n.
  return get_prediction_index(
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
```

```
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts
cannot be generated. To use the model for forecasting, use one of the supported c
lasses of index.
  self._init_dates(dates, freq)
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\sar
imax.py:966: UserWarning: Non-stationary starting autoregressive parameters foun
d. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: ValueWarning: No supported index is available. Prediction results will
be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\Administrator\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:837: FutureWarning: No supported index is available. In the next version, ca
lling this method in a model without a supported index will result in an exceptio
n.
  return get_prediction_index(
```



```
Forecast for next 24 time steps:
{'Bury Ground': 403    0.334258
404    0.334478
405    0.334667
406    0.334829
407    0.334967
408    0.335086
409    0.335188
410    0.335275
411    0.335349
412    0.335413
413    0.335468
414    0.335515
415    0.335555
416    0.335589
417    0.335619
418    0.335644
419    0.335665
420    0.335684
421    0.335700
422    0.335713
423    0.335725
424    0.335735
425    0.335743
426    0.335751
Name: predicted_mean, dtype: float64, 'Manchester Racecourse': 403    0.971279
404    0.971436
405    0.971524
406    0.971573
407    0.971600
408    0.971616
409    0.971624
410    0.971629
411    0.971632
412    0.971633
413    0.971634
414    0.971635
415    0.971635
416    0.971635
417    0.971635
418    0.971635
419    0.971635
420    0.971635
421    0.971635
422    0.971635
423    0.971635
424    0.971635
425    0.971635
426    0.971635
Name: predicted_mean, dtype: float64, 'Rochdale': 403    0.194930
404    0.194865
405    0.194804
406    0.194748
407    0.194695
408    0.194646
409    0.194600
410    0.194557
411    0.194517
412    0.194480
413    0.194446
```

```
414    0.194413
415    0.194383
416    0.194355
417    0.194329
418    0.194304
419    0.194282
420    0.194260
421    0.194241
422    0.194222
423    0.194205
424    0.194189
425    0.194174
426    0.194160
Name: predicted_mean, dtype: float64}
```

Integrate forecasts into the alert system

In [84]:
```
def get_forecast_alert(row, station_forecast):
    thresholds = station_thresholds[row['location_name']]
    current_score = row['adjusted_risk']
    forecast_score = (station_forecast.iloc[0] - row['river_level']) / row['rive

    combined_score = max(current_score, forecast_score)

    if combined_score < thresholds['low']:
        return "Low"
    elif combined_score < thresholds['medium']:
        return "Normal"
    elif combined_score < thresholds['high']:
        return "Advisory"
    else:
        return "Warning"

data['forecast_alert'] = data.apply(lambda row: get_forecast_alert(row, forecast

print(data[['location_name', 'station_alert', 'forecast_alert']].head(15))
```

```
          location_name station_alert forecast_alert
0           Bury Ground           Low            Low
1   Manchester Racecourse       Warning        Warning
2              Rochdale      Advisory       Advisory
3           Bury Ground           Low            Low
4   Manchester Racecourse       Warning        Warning
5              Rochdale      Advisory       Advisory
6           Bury Ground           Low            Low
7   Manchester Racecourse       Warning        Warning
8              Rochdale      Advisory       Advisory
9           Bury Ground           Low            Low
10  Manchester Racecourse       Warning        Warning
11             Rochdale      Advisory       Advisory
12          Bury Ground           Low            Low
13  Manchester Racecourse       Warning        Warning
14             Rochdale      Advisory       Advisory
```

Web Interface

In [85]:
```
pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.42.0-py2.py3-none-any.whl.metadata (8.9 kB)
Collecting altair<6,>=4.0 (from streamlit)
  Downloading altair-5.5.0-py3-none-any.whl.metadata (11 kB)
Collecting blinker<2,>=1.0.0 (from streamlit)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting cachetools<6,>=4.0 (from streamlit)
  Downloading cachetools-5.5.1-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: click<9,>=7.0 in c:\users\administrator\anaconda3
\lib\site-packages (from streamlit) (8.1.7)
Requirement already satisfied: numpy<3,>=1.23 in c:\users\administrator\anaconda3
\lib\site-packages (from streamlit) (1.26.4)
Requirement already satisfied: packaging<25,>=20 in c:\users\administrator\anacon
da3\lib\site-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.4.0 in c:\users\administrator\appdata
\roaming\python\python312\site-packages (from streamlit) (2.2.3)
Requirement already satisfied: pillow<12,>=7.1.0 in c:\users\administrator\anacon
da3\lib\site-packages (from streamlit) (10.4.0)
Collecting protobuf<6,>=3.20 (from streamlit)
  Downloading protobuf-5.29.3-cp310-abi3-win_amd64.whl.metadata (592 bytes)
Collecting pyarrow>=7.0 (from streamlit)
  Downloading pyarrow-19.0.0-cp312-cp312-win_amd64.whl.metadata (3.4 kB)
Requirement already satisfied: requests<3,>=2.27 in c:\users\administrator\anacon
da3\lib\site-packages (from streamlit) (2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in c:\users\administrator\anacon
da3\lib\site-packages (from streamlit) (13.7.1)
Collecting tenacity<10,>=8.1.0 (from streamlit)
  Downloading tenacity-9.0.0-py3-none-any.whl.metadata (1.2 kB)
Collecting toml<2,>=0.10.1 (from streamlit)
  Downloading toml-0.10.2-py2.py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in c:\users\administra
tor\anaconda3\lib\site-packages (from streamlit) (4.11.0)
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-win_amd64.whl.metadata (44 kB)
Collecting gitpython!=3.1.19,<4,>=3.0.7 (from streamlit)
  Downloading GitPython-3.1.44-py3-none-any.whl.metadata (13 kB)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in c:\users\administrator\anacon
da3\lib\site-packages (from streamlit) (6.4.1)
Requirement already satisfied: jinja2 in c:\users\administrator\anaconda3\lib\sit
e-packages (from altair<6,>=4.0->streamlit) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in c:\users\administrator\anaconda
3\lib\site-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Collecting narwhals>=1.14.2 (from altair<6,>=4.0->streamlit)
  Downloading narwhals-1.25.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: colorama in c:\users\administrator\anaconda3\lib\s
ite-packages (from click<9,>=7.0->streamlit) (0.4.6)
Collecting gitdb<5,>=4.0.1 (from gitpython!=3.1.19,<4,>=3.0.7->streamlit)
  Downloading gitdb-4.0.12-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\administrator\a
naconda3\lib\site-packages (from pandas<3,>=1.4.0->streamlit) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\administrator\anaconda3\l
ib\site-packages (from pandas<3,>=1.4.0->streamlit) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\administrator\appdata\r
oaming\python\python312\site-packages (from pandas<3,>=1.4.0->streamlit) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\administrator
\anaconda3\lib\site-packages (from requests<3,>=2.27->streamlit) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\administrator\anaconda3\l
ib\site-packages (from requests<3,>=2.27->streamlit) (3.7)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\administrator\anaco
nda3\lib\site-packages (from requests<3,>=2.27->streamlit) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\administrator\anaco
nda3\lib\site-packages (from requests<3,>=2.27->streamlit) (2025.1.31)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\administrator\an
aconda3\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\administrator
\anaconda3\lib\site-packages (from rich<14,>=10.14.0->streamlit) (2.15.1)
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->s
treamlit)
  Downloading smmap-5.0.2-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\administrator\anaconda
3\lib\site-packages (from jinja2->altair<6,>=4.0->streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in c:\users\administrator\anaconda3
\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\a
dministrator\anaconda3\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->s
treamlit) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\administrator\anac
onda3\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.30.
2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\administrator\anaconda3
\lib\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.10.6)
Requirement already satisfied: mdurl~=0.1 in c:\users\administrator\anaconda3\lib
\site-packages (from markdown-it-py>=2.2.0->rich<14,>=10.14.0->streamlit) (0.1.0)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3\lib\s
ite-packages (from python-dateutil>=2.8.2->pandas<3,>=1.4.0->streamlit) (1.16.0)
Downloading streamlit-1.42.0-py2.py3-none-any.whl (9.6 MB)
   ---------------------------------------- 0.0/9.6 MB ? eta -:--:--
   ------ --------------------------------- 1.6/9.6 MB 9.4 MB/s eta 0:00:01
   --------------- ------------------------ 3.7/9.6 MB 9.1 MB/s eta 0:00:01
   ------------------------ --------------- 5.8/9.6 MB 9.3 MB/s eta 0:00:01
   ------------------------------- -------- 7.6/9.6 MB 9.2 MB/s eta 0:00:01
   ---------------------------------------  9.4/9.6 MB 9.2 MB/s eta 0:00:01
   ---------------------------------------- 9.6/9.6 MB 8.5 MB/s eta 0:00:00
Downloading altair-5.5.0-py3-none-any.whl (731 kB)
   ---------------------------------------- 0.0/731.2 kB ? eta -:--:--
   ---------------------------------------- 731.2/731.2 kB 7.4 MB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading cachetools-5.5.1-py3-none-any.whl (9.5 kB)
Downloading GitPython-3.1.44-py3-none-any.whl (207 kB)
Downloading protobuf-5.29.3-cp310-abi3-win_amd64.whl (434 kB)
Downloading pyarrow-19.0.0-cp312-cp312-win_amd64.whl (25.2 MB)
   ---------------------------------------- 0.0/25.2 MB ? eta -:--:--
   -- ------------------------------------- 1.8/25.2 MB 9.1 MB/s eta 0:00:03
   ------ --------------------------------- 3.9/25.2 MB 9.4 MB/s eta 0:00:03
   --------- ------------------------------ 5.8/25.2 MB 9.3 MB/s eta 0:00:03
   ------------ --------------------------- 7.9/25.2 MB 9.4 MB/s eta 0:00:02
   --------------- ------------------------ 10.0/25.2 MB 9.3 MB/s eta 0:00:02
   ------------------ --------------------- 11.8/25.2 MB 9.2 MB/s eta 0:00:02
   ---------------------- ----------------- 13.9/25.2 MB 9.2 MB/s eta 0:00:02
   ------------------------- -------------- 16.0/25.2 MB 9.2 MB/s eta 0:00:02
   ---------------------------- ----------- 17.8/25.2 MB 9.2 MB/s eta 0:00:01
   ------------------------------- -------- 19.9/25.2 MB 9.3 MB/s eta 0:00:01
   ---------------------------------- ----- 21.5/25.2 MB 9.2 MB/s eta 0:00:01
   ------------------------------------- -- 23.6/25.2 MB 9.2 MB/s eta 0:00:01
   ---------------------------------------  25.2/25.2 MB 9.2 MB/s eta 0:00:01
   ---------------------------------------- 25.2/25.2 MB 8.7 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ---------------------------------------- 0.0/6.9 MB ? eta -:--:--
```

```
   ---------- ------------------------------ 1.8/6.9 MB 7.7 MB/s eta 0:00:01
   ---------------------- ------------------ 3.7/6.9 MB 8.4 MB/s eta 0:00:01
   ------------------------------ --------- 5.2/6.9 MB 8.2 MB/s eta 0:00:01
   ---------------------------------------  6.8/6.9 MB 8.1 MB/s eta 0:00:01
   ---------------------------------------- 6.9/6.9 MB 7.6 MB/s eta 0:00:00
Downloading tenacity-9.0.0-py3-none-any.whl (28 kB)
Downloading toml-0.10.2-py2.py3-none-any.whl (16 kB)
Downloading watchdog-6.0.0-py3-none-win_amd64.whl (79 kB)
Downloading gitdb-4.0.12-py3-none-any.whl (62 kB)
Downloading narwhals-1.25.2-py3-none-any.whl (305 kB)
Downloading smmap-5.0.2-py3-none-any.whl (24 kB)
Installing collected packages: watchdog, toml, tenacity, smmap, pyarrow, protobu
f, narwhals, cachetools, blinker, pydeck, gitdb, gitpython, altair, streamlit
Successfully installed altair-5.5.0 blinker-1.9.0 cachetools-5.5.1 gitdb-4.0.12 g
itpython-3.1.44 narwhals-1.25.2 protobuf-5.29.3 pyarrow-19.0.0 pydeck-0.9.1 smmap
-5.0.2 streamlit-1.42.0 tenacity-9.0.0 toml-0.10.2 watchdog-6.0.0
Note: you may need to restart the kernel to use updated packages.
```

In [93]:

```python
import pandas as pd

# Load existing project data for reference
data = pd.read_csv('C:/Users/Administrator/NEWPROJECT/cleaned_data/merged_realti

# Project Web Interface Requirements Assessment
def assess_web_interface_requirements():
    print("Web Interface Development - Initial Requirements")

    # Data Overview
    print("\n1. Data Characteristics:")
    print(f"Total Stations: {data['location_name'].nunique()}")
    print("Stations:", ", ".join(data['location_name'].unique()))

    # Print available columns to verify
    print("\nAvailable Columns:")
    print(data.columns.tolist())

    # Basic data summary
    print("\n2. Data Summary by Station:")
    station_summary = data.groupby('location_name').agg({
        'river_level': ['mean', 'min', 'max'],
        'rainfall': ['mean', 'min', 'max']
    })
    print(station_summary)

    # Proposed Dashboard Features
    required_features = [
        'River Level Monitoring',
        'Real-time Risk Assessment',
        'Historical Trend Visualization',
        'Station-specific Alerts',
        'Rainfall Correlation Display'
    ]

    print("\n3. Proposed Dashboard Features:")
    for feature in required_features:
        print(f"- {feature}")

    # Technical Requirements
    print("\n4. Technical Requirements:")
    print("- Responsive Design")
```

```
    print("- Real-time Data Updates")
    print("- Interactive Visualizations")
    print("- Mobile-Friendly Interface")

# Run the assessment
assess_web_interface_requirements()
```

```
Web Interface Development - Initial Requirements

1. Data Characteristics:
Total Stations: 3
Stations: Bury Ground, Manchester Racecourse, Rochdale

Available Columns:
['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp', 'location_na
me', 'river_station_id', 'rainfall_station_id']

2. Data Summary by Station:
                      river_level                 rainfall
                             mean    min    max       mean  min  max
location_name
Bury Ground              0.365196  0.333  0.441  0.020347  0.0  1.0
Manchester Racecourse    1.039347  0.962  1.203  0.020099  0.0  1.0
Rochdale                 0.223757  0.195  0.293  0.016873  0.0  0.6

3. Proposed Dashboard Features:
- River Level Monitoring
- Real-time Risk Assessment
- Historical Trend Visualization
- Station-specific Alerts
- Rainfall Correlation Display

4. Technical Requirements:
- Responsive Design
- Real-time Data Updates
- Interactive Visualizations
- Mobile-Friendly Interface
```

In [96]:

```
import pandas as pd
import os

# Specify the directory containing your CSV files
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'

# Find the most recent CSV file
csv_files = [f for f in os.listdir(data_directory) if f.startswith('combined_dat

if csv_files:
    # Get the most recent file
    latest_file = max(csv_files, key=lambda f: os.path.getctime(os.path.join(dat
    full_path = os.path.join(data_directory, latest_file)

    # Read the CSV file
    df = pd.read_csv(full_path)

    # Print column names and first few rows
    print("Columns in the CSV:")
    print(df.columns.tolist())

    print("\nFirst few rows:")
```

```
    print(df.head())
else:
    print("No CSV files found in the directory.")
```

```
Columns in the CSV:
['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp', 'location_na
me', 'river_station_id', 'rainfall_station_id']

First few rows:
   river_level       river_timestamp  rainfall    rainfall_timestamp  \
0        0.181  2025-02-07T21:30:00Z       0.0  2025-02-07T21:30:00Z
1        0.951  2025-02-07T21:30:00Z       0.0  2025-02-07T21:30:00Z
2        0.318  2025-02-07T21:30:00Z       0.0  2025-02-07T21:30:00Z

            location_name  river_station_id  rainfall_station_id
0                Rochdale            690203               561613
1  Manchester Racecourse             690510               562992
2             Bury Ground            690160               562656
```

Risk Assessment

In [98]:

```
# Open Jupyter Notebook
# Create a new cell and run this code

import pandas as pd

# Load the most recent CSV file
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
latest_csv = max(
    [f for f in os.listdir(data_directory) if f.startswith('combined_data_') and
    key=lambda f: os.path.getctime(os.path.join(data_directory, f))
)

# Read the CSV
current_data = pd.read_csv(os.path.join(data_directory, latest_csv))

# Display current risk assessment method
print("Current Risk Assessment Thresholds:")
print("High Risk: River Level > 0.7 m")
print("Moderate Risk: River Level > 0.4 m")
print("Low Risk: River Level <= 0.4 m")

# Show current risk levels
def current_risk_assessment(river_level):
    if river_level > 0.7:
        return 'High Risk'
    elif river_level > 0.4:
        return 'Moderate Risk'
    else:
        return 'Low Risk'

current_data['Current Risk Level'] = current_data['river_level'].apply(current_r
print("\nCurrent Risk Levels:")
print(current_data[['location_name', 'river_level', 'Current Risk Level']])
```

```
Current Risk Assessment Thresholds:
High Risk: River Level > 0.7 m
Moderate Risk: River Level > 0.4 m
Low Risk: River Level <= 0.4 m

Current Risk Levels:
           location_name  river_level Current Risk Level
0               Rochdale        0.170           Low Risk
1  Manchester Racecourse        0.933          High Risk
2            Bury Ground        0.316           Low Risk
```

Enhanced Risk Assessment

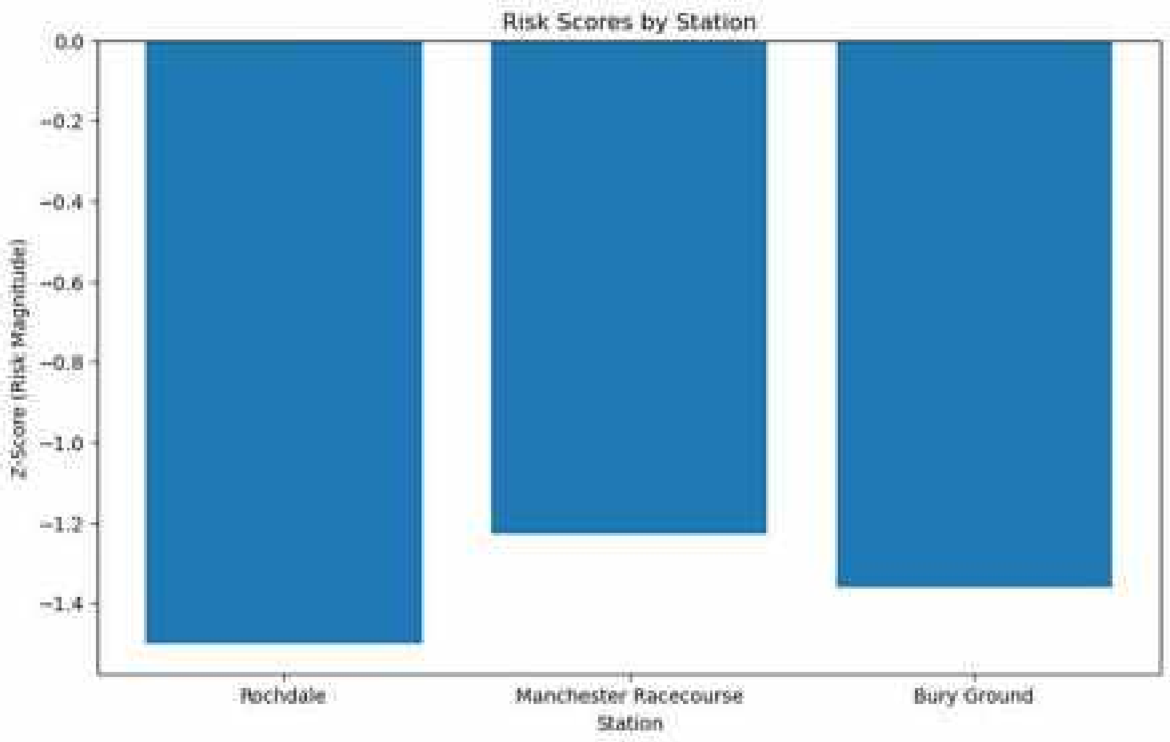
In [100…

```python
import pandas as pd
import os
import glob
import numpy as np
import matplotlib.pyplot as plt

# Function to load the latest CSV file
def load_latest_csv(data_directory):
    # Find all CSV files
    csv_files = glob.glob(os.path.join(data_directory, 'combined_data_*.csv'))

    # Check if any files exist
    if not csv_files:
        raise FileNotFoundError(f"No CSV files found in {data_directory}")

    # Get the most recent file based on creation time
    latest_file = max(csv_files, key=os.path.getctime)

    # Print the file being loaded for verification
    print(f"Loading file: {latest_file}")

    # Read the CSV
    return pd.read_csv(latest_file)

# Function to load historical data
def load_historical_data(data_directory):
    # Find all CSV files
    csv_files = glob.glob(os.path.join(data_directory, 'combined_data_*.csv'))

    # Read and combine all historical files
    historical_dataframes = []
    for file in csv_files:
        df = pd.read_csv(file)
        historical_dataframes.append(df)

    historical_data = pd.concat(historical_dataframes, ignore_index=True)
    return historical_data

# Enhanced risk assessment function
def enhanced_risk_assessment(current_data, historical_data):
    # Calculate station statistics
    station_stats = historical_data.groupby('location_name').agg({
        'river_level': ['mean', 'std', 'min', 'max']
    })

    # Function to calculate z-score
```

```
    def calculate_z_score(value, mean, std):
        return (value - mean) / std if std != 0 else 0

    # Enhanced risk classification
    def classify_risk(row, stats):
        station = row['location_name']
        river_level = row['river_level']

        # Calculate z-score
        mean = stats.loc[station, ('river_level', 'mean')]
        std = stats.loc[station, ('river_level', 'std')]
        z_score = calculate_z_score(river_level, mean, std)

        # More nuanced risk assessment
        if z_score > 2:
            return 'Critical Risk', z_score
        elif z_score > 1:
            return 'High Risk', z_score
        elif z_score > 0.5:
            return 'Moderate Risk', z_score
        elif z_score < -1:
            return 'Unusually Low', z_score
        else:
            return 'Low Risk', z_score

    # Apply enhanced risk assessment
    current_data[['Enhanced Risk Level', 'Z-Score']] = current_data.apply(
        lambda row: classify_risk(row, station_stats),
        axis=1,
        result_type='expand'
    )

    return current_data

# Main execution
def main():
    # Specify the directory
    data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'

    try:
        # Load current data
        current_data = load_latest_csv(data_directory)

        # Load historical data
        historical_data = load_historical_data(data_directory)

        # Apply enhanced risk assessment
        enhanced_results = enhanced_risk_assessment(current_data, historical_dat

        # Display results
        print("\nEnhanced Risk Assessment Results:")
        print(enhanced_results[['location_name', 'river_level', 'Enhanced Risk L

        # Visualize results
        plt.figure(figsize=(10, 6))
        plt.bar(enhanced_results['location_name'], enhanced_results['Z-Score'])
        plt.title('Risk Scores by Station')
        plt.xlabel('Station')
        plt.ylabel('Z-Score (Risk Magnitude)')
        plt.axhline(y=0, color='r', linestyle='--')
```

```
        plt.show()

    except Exception as e:
        print(f"An error occurred: {e}")

# Run the main function
main()
```

```
Loading file: C:/Users/Administrator/NEWPROJECT/combined_data\combined_data_20250
208_142915.csv

Enhanced Risk Assessment Results:
           location_name  river_level Enhanced Risk Level   Z-Score
0               Rochdale        0.170       Unusually Low -1.500201
1  Manchester Racecourse        0.933       Unusually Low -1.228876
2            Bury Ground        0.316       Unusually Low -1.363017
```



Predictive Modeling Development

In [101…

```
import pandas as pd
import numpy as np
import os
import glob
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Function to collect and prepare data for predictive modeling
def prepare_predictive_modeling_data(data_directory):
    # Find all CSV files
    csv_files = glob.glob(os.path.join(data_directory, 'combined_data_*.csv'))

    # Read and combine all files
    dataframes = [pd.read_csv(file) for file in csv_files]
    combined_data = pd.concat(dataframes, ignore_index=True)

    # Convert timestamps
    combined_data['river_timestamp'] = pd.to_datetime(combined_data['river_times
```

```
# Feature engineering
def create_features(df):
    # Sort by timestamp
    df = df.sort_values('river_timestamp')

    # Create time-based features
    df['hour'] = df['river_timestamp'].dt.hour
    df['day_of_week'] = df['river_timestamp'].dt.dayofweek
    df['month'] = df['river_timestamp'].dt.month

    # Create lag features
    df['prev_river_level'] = df.groupby('location_name')['river_level'].shif
    df['river_level_change'] = df['river_level'] - df['prev_river_level']

    # Rolling window features
    df['river_level_7day_mean'] = df.groupby('location_name')['river_level']
    df['rainfall_7day_mean'] = df.groupby('location_name')['rainfall'].rolli

    return df

# Apply feature engineering
combined_data = create_features(combined_data)

# Remove rows with NaN (first rows after creating lag features)
combined_data_clean = combined_data.dropna()

print("Data Preparation Summary:")
print(f"Total Records: {len(combined_data_clean)}")
print("\nFeatures Created:")
print(combined_data_clean.columns.tolist())

# Prepare data for each station
station_data = {}
for station in combined_data_clean['location_name'].unique():
    station_df = combined_data_clean[combined_data_clean['location_name'] ==

    # Select features
    features = [
        'prev_river_level', 'river_level_change',
        'rainfall', 'river_level_7day_mean',
        'rainfall_7day_mean', 'hour',
        'day_of_week', 'month'
    ]

    X = station_df[features]
    y = station_df['river_level']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    station_data[station] = {
        'X_train': X_train_scaled,
        'X_test': X_test_scaled,
        'y_train': y_train,
        'y_test': y_test,
```

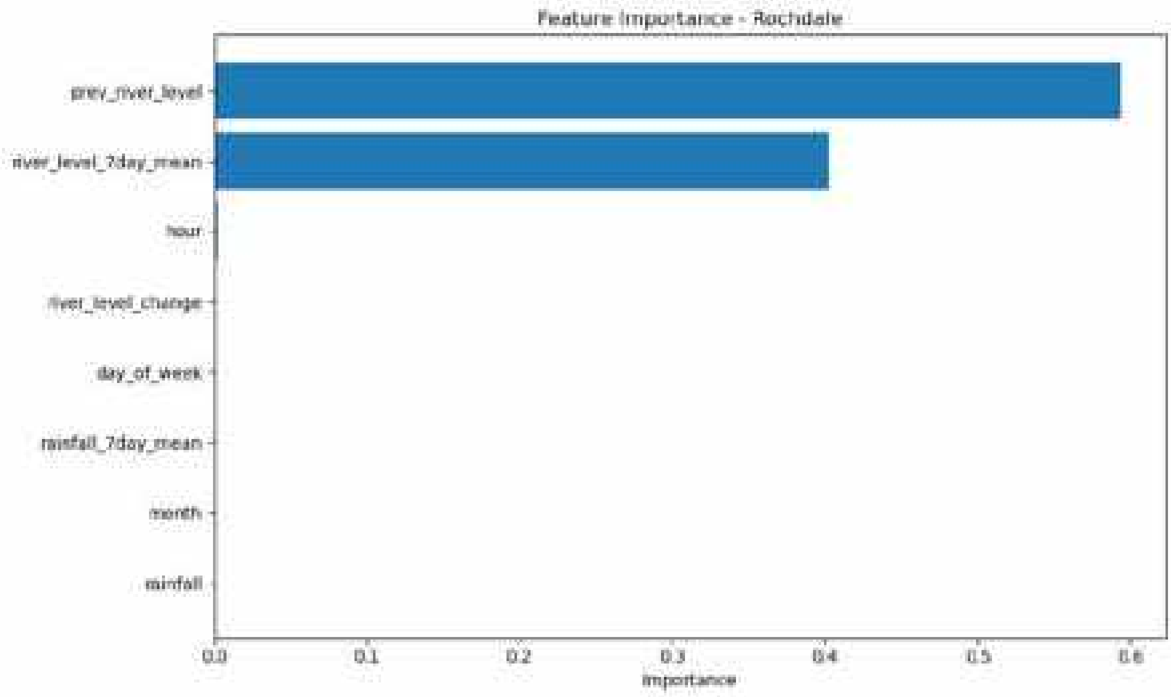
```
            'scaler': scaler
        }

    return station_data

# Prepare data
data_directory = 'C:/Users/Administrator/NEWPROJECT/combined_data'
predictive_data = prepare_predictive_modeling_data(data_directory)

# Print summary for each station
for station, data in predictive_data.items():
    print(f"\n{station} Predictive Modeling Data:")
    print(f"Training Samples: {len(data['y_train'])}")
    print(f"Testing Samples: {len(data['y_test'])}")
```

```
Data Preparation Summary:
Total Records: 2078

Features Created:
['river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp', 'location_na
me', 'river_station_id', 'rainfall_station_id', 'hour', 'day_of_week', 'month',
'prev_river_level', 'river_level_change', 'river_level_7day_mean', 'rainfall_7day
_mean']

Rochdale Predictive Modeling Data:
Training Samples: 554
Testing Samples: 139

Manchester Racecourse Predictive Modeling Data:
Training Samples: 553
Testing Samples: 139

Bury Ground Predictive Modeling Data:
Training Samples: 554
Testing Samples: 139
```

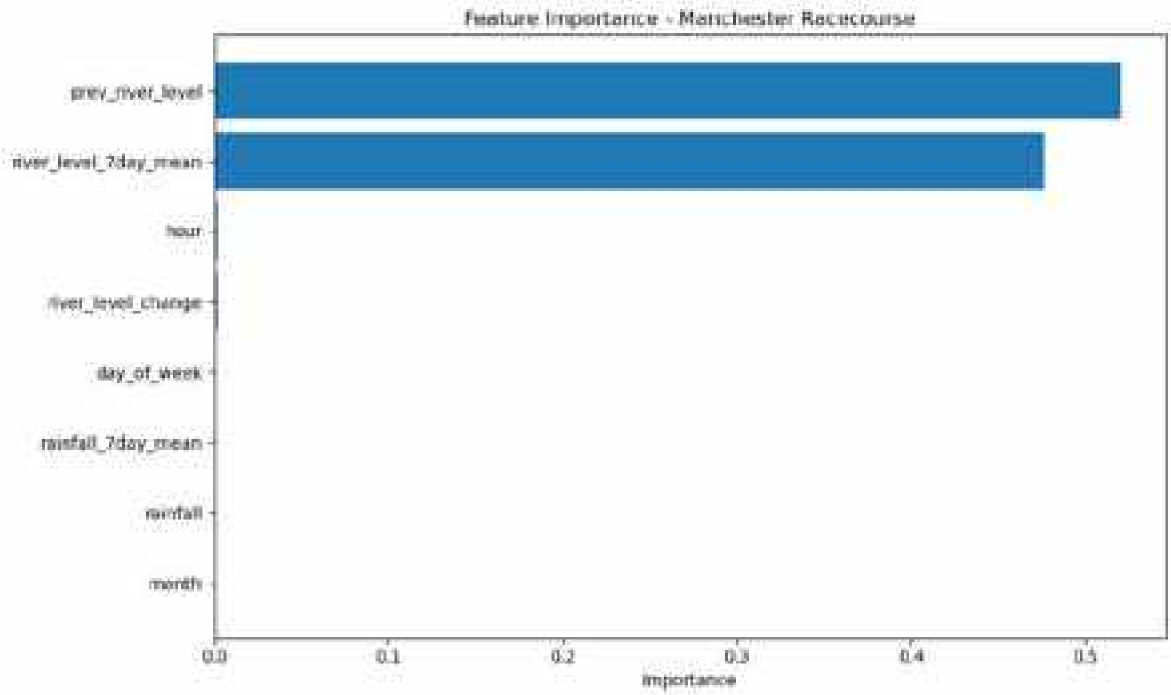
In [102…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

def train_predictive_models(predictive_data):
    models = {}

    for station, data in predictive_data.items():
        # Train Random Forest Regressor
        model = RandomForestRegressor(
            n_estimators=100,  # Number of trees
            random_state=42,
            max_depth=10
        )

        # Fit the model
        model.fit(data['X_train'], data['y_train'])

        # Make predictions
        y_pred_train = model.predict(data['X_train'])
        y_pred_test = model.predict(data['X_test'])

        # Evaluate model performance
        train_mse = mean_squared_error(data['y_train'], y_pred_train)
```

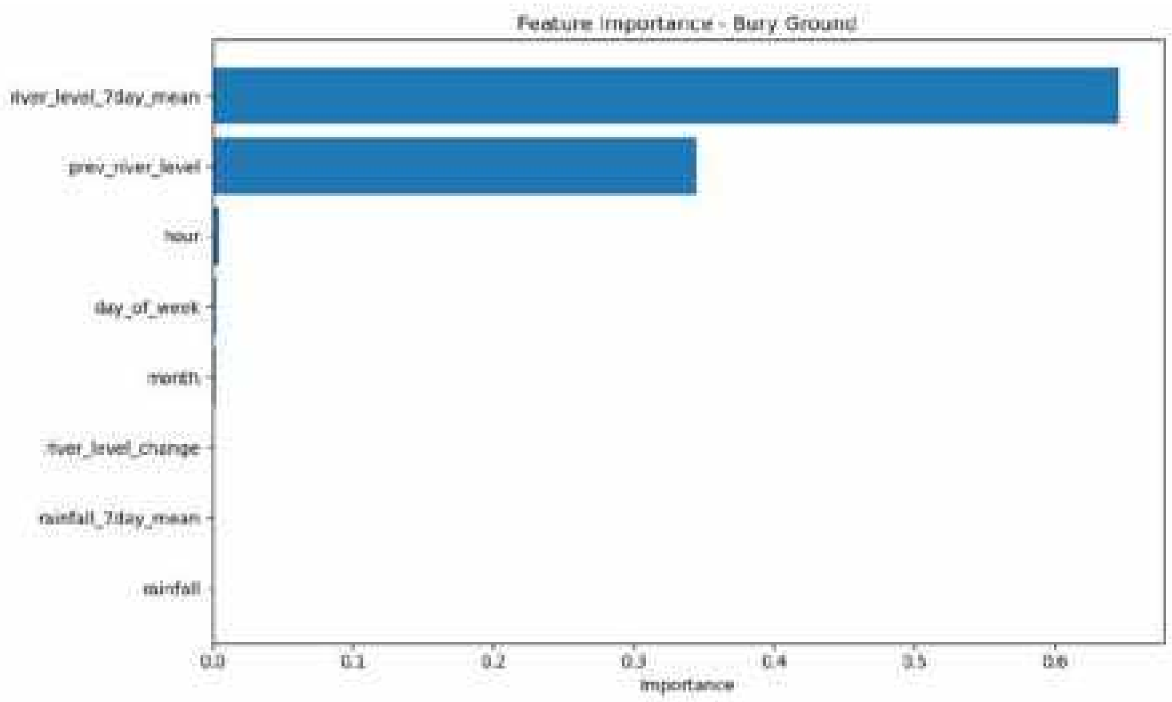
```
        test_mse = mean_squared_error(data['y_test'], y_pred_test)
        train_r2 = r2_score(data['y_train'], y_pred_train)
        test_r2 = r2_score(data['y_test'], y_pred_test)

        # Store model and performance metrics
        models[station] = {
            'model': model,
            'train_mse': train_mse,
            'test_mse': test_mse,
            'train_r2': train_r2,
            'test_r2': test_r2
        }

        # Visualize feature importance
        plt.figure(figsize=(10, 6))
        feature_importance = model.feature_importances_
        feature_names = [
            'prev_river_level', 'river_level_change',
            'rainfall', 'river_level_7day_mean',
            'rainfall_7day_mean', 'hour',
            'day_of_week', 'month'
        ]

        sorted_idx = feature_importance.argsort()
        plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx])
        plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx
        plt.title(f'Feature Importance - {station}')
        plt.xlabel('Importance')
        plt.tight_layout()
        plt.show()

        # Print model performance
        print(f"\n{station} Model Performance:")
        print(f"Training MSE: {train_mse:.4f}")
        print(f"Testing MSE: {test_mse:.4f}")
        print(f"Training R² Score: {train_r2:.4f}")
        print(f"Testing R² Score: {test_r2:.4f}")

    return models

# Train predictive models
predictive_models = train_predictive_models(predictive_data)
```




```
Rochdale Model Performance:
Training MSE: 0.0000
Testing MSE: 0.0000
Training R² Score: 0.9997
Testing R² Score: 0.9965
```




```
Manchester Racecourse Model Performance:
Training MSE: 0.0000
Testing MSE: 0.0000
Training R² Score: 0.9996
Testing R² Score: 0.9977
```




```
Bury Ground Model Performance:
Training MSE: 0.0000
Testing MSE: 0.0000
Training R² Score: 0.9997
Testing R² Score: 0.9987
```

Geospatial Integration and Visualization

In [5]:
```
pip install pyproj folium shapely geopandas
```

```
Requirement already satisfied: pyproj in c:\users\administrator\anaconda3\lib\sit
e-packages (3.6.1)
Requirement already satisfied: folium in c:\users\administrator\anaconda3\lib\sit
e-packages (0.19.4)
Requirement already satisfied: shapely in c:\users\administrator\anaconda3\lib\si
te-packages (2.0.5)
Requirement already satisfied: geopandas in c:\users\administrator\anaconda3\lib
\site-packages (0.14.4)
Requirement already satisfied: certifi in c:\users\administrator\anaconda3\lib\si
te-packages (from pyproj) (2025.1.31)
Requirement already satisfied: branca>=0.6.0 in c:\users\administrator\anaconda3
\lib\site-packages (from folium) (0.8.1)
Requirement already satisfied: jinja2>=2.9 in c:\users\administrator\anaconda3\li
b\site-packages (from folium) (3.1.4)
Requirement already satisfied: numpy in c:\users\administrator\anaconda3\lib\site
-packages (from folium) (1.26.4)
Requirement already satisfied: requests in c:\users\administrator\anaconda3\lib\s
ite-packages (from folium) (2.32.3)
Requirement already satisfied: xyzservices in c:\users\administrator\anaconda3\li
b\site-packages (from folium) (2025.1.0)
Requirement already satisfied: fiona>=1.8.21 in c:\users\administrator\anaconda3
\lib\site-packages (from geopandas) (1.10.1)
Requirement already satisfied: packaging in c:\users\administrator\anaconda3\lib
\site-packages (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in c:\users\administrator\appdata\ro
aming\python\python312\site-packages (from geopandas) (2.2.3)
Requirement already satisfied: attrs>=19.2.0 in c:\users\administrator\anaconda3
\lib\site-packages (from fiona>=1.8.21->geopandas) (23.1.0)
Requirement already satisfied: click~=8.0 in c:\users\administrator\anaconda3\lib
\site-packages (from fiona>=1.8.21->geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in c:\users\administrator\anaco
nda3\lib\site-packages (from fiona>=1.8.21->geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in c:\users\administrator\anaconda3\lib
\site-packages (from fiona>=1.8.21->geopandas) (0.7.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\administrator\anaconda
3\lib\site-packages (from jinja2>=2.9->folium) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\administrator\a
naconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\administrator\anaconda3\l
ib\site-packages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\administrator\appdata\r
oaming\python\python312\site-packages (from pandas>=1.4.0->geopandas) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\administrator
\anaconda3\lib\site-packages (from requests->folium) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\administrator\anaconda3\l
ib\site-packages (from requests->folium) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\administrator\anaco
nda3\lib\site-packages (from requests->folium) (2.2.3)
Requirement already satisfied: colorama in c:\users\administrator\anaconda3\lib\s
ite-packages (from click~=8.0->fiona>=1.8.21->geopandas) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3\lib\s
ite-packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```python
import geopandas as gpd
import pandas as pd
import folium
from shapely.geometry import Point
import pyproj
```

```
# Station coordinates
stations = {
    'Bury Ground': {
        'lat': 53.598766,
        'lon': -2.305182,
        'river_level_threshold': 0.5
    },
    'Manchester Racecourse': {
        'lat': 53.499526,
        'lon': -2.271756,
        'river_level_threshold': 0.7
    },
    'Rochdale': {
        'lat': 53.611067,
        'lon': -2.178685,
        'river_level_threshold': 0.4
    }
}

# Create DataFrame first
df = pd.DataFrame.from_dict(stations, orient='index')

# Create geometry column using Shapely
geometry = [Point(xy) for xy in zip(df['lon'], df['lat'])]

# Create GeoDataFrame with explicit geometry
gdf = gpd.GeoDataFrame(df, geometry=geometry, crs="EPSG:4326")

# Reproject to a projected CRS for accurate distance calculations
gdf_projected = gdf.to_crs("EPSG:27700")

# Advanced Distance Calculation Function
def haversine_distance(lat1, lon1, lat2, lon2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    from math import radians, sin, cos, sqrt, atan2

    # Convert decimal degrees to radians
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * atan2(sqrt(a), sqrt(1-a))

    # Radius of earth in kilometers
    radius = 6371.0

    return radius * c

# Calculate distances using Haversine method
def calculate_haversine_distances(gdf):
    distances = []
    station_names = gdf.index.tolist()

    for i, station1 in enumerate(station_names):
        row_distances = []
```

```
        for j, station2 in enumerate(station_names):
            if i == j:
                row_distances.append(0)
            else:
                dist = haversine_distance(
                    gdf.loc[station1, 'lat'],
                    gdf.loc[station1, 'lon'],
                    gdf.loc[station2, 'lat'],
                    gdf.loc[station2, 'lon']
                )
                row_distances.append(dist)
        distances.append(row_distances)

    return distances, station_names

# Calculate distances
haversine_distances, station_names = calculate_haversine_distances(gdf)

# Comprehensive Spatial Analysis
def spatial_analysis(gdf_proj, gdf_orig):
    # Manual centroid calculation for projected CRS
    proj_x = gdf_proj.geometry.x.mean()
    proj_y = gdf_proj.geometry.y.mean()
    proj_centroid = Point(proj_x, proj_y)

    # Manual centroid calculation for original CRS
    orig_x = gdf_orig.geometry.x.mean()
    orig_y = gdf_orig.geometry.y.mean()
    orig_centroid = Point(orig_x, orig_y)

    return {
        'projected_centroid': proj_centroid,
        'original_centroid': orig_centroid,
        'projected_bounds': gdf_proj.total_bounds,
        'original_bounds': gdf_orig.total_bounds
    }

# Perform spatial analysis
spatial_info = spatial_analysis(gdf_projected, gdf)

# Visualization Function
def create_enhanced_risk_map(gdf):
    # Manual center calculation
    center_lat = gdf.geometry.y.mean()
    center_lon = gdf.geometry.x.mean()

    # Create map
    m = folium.Map(location=[center_lat, center_lon], zoom_start=10)

    # Add markers with enhanced information
    for idx, row in gdf.iterrows():
        # Risk level determination
        if row['river_level_threshold'] <= 0.4:
            color, risk_level = 'green', 'Low Risk'
        elif row['river_level_threshold'] <= 0.7:
            color, risk_level = 'orange', 'Moderate Risk'
        else:
            color, risk_level = 'red', 'High Risk'

        # Calculate distances to other stations
```

```
        dist_info = []
        for j, other_station in enumerate(gdf.index):
            if idx != other_station:
                dist_info.append(f"{other_station}: {haversine_distances[list(gd

        # Create detailed popup
        popup_text = f"""
        <b>{idx}</b><br>
        Latitude: {row.lat:.6f}<br>
        Longitude: {row.lon:.6f}<br>
        Risk Threshold: {row['river_level_threshold']}m<br>
        Risk Level: {risk_level}<br>
        <b>Distances:</b><br>
        {' | '.join(dist_info)}
        """

        # Add circle marker
        folium.CircleMarker(
            location=[row.lat, row.lon],
            radius=10,
            popup=folium.Popup(popup_text, max_width=300),
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.7
        ).add_to(m)

    # Save the map
    m.save("enhanced_river_stations_map.html")
    return m

# Generate the enhanced map
enhanced_risk_map = create_enhanced_risk_map(gdf)

# Print Comprehensive Results
print("\nComprehensive Spatial Analysis:")
print("\nStation Coordinates:")
for idx, row in gdf.iterrows():
    print(f"{idx}: Lat {row.lat}, Lon {row.lon}")

print("\nDistance Matrix (Haversine Method, km):")
for i, station in enumerate(station_names):
    print(f"{station}:")
    for j, other_station in enumerate(station_names):
        print(f"  Distance to {other_station}: {haversine_distances[i][j]:.2f} k

print("\nSpatial Analysis Summary:")
print(f"Original Centroid: {spatial_info['original_centroid']}")
print(f"Projected Centroid: {spatial_info['projected_centroid']}")
print("\nBounding Box:")
print("Original CRS:", spatial_info['original_bounds'])
print("Projected CRS:", spatial_info['projected_bounds'])
```

```
Comprehensive Spatial Analysis:

Station Coordinates:
Bury Ground: Lat 53.598766, Lon -2.305182
Manchester Racecourse: Lat 53.499526, Lon -2.271756
Rochdale: Lat 53.611067, Lon -2.178685

Distance Matrix (Haversine Method, km):
Bury Ground:
  Distance to Bury Ground: 0.00 km
  Distance to Manchester Racecourse: 11.25 km
  Distance to Rochdale: 8.46 km
Manchester Racecourse:
  Distance to Bury Ground: 11.25 km
  Distance to Manchester Racecourse: 0.00 km
  Distance to Rochdale: 13.84 km
Rochdale:
  Distance to Bury Ground: 8.46 km
  Distance to Manchester Racecourse: 13.84 km
  Distance to Rochdale: 0.00 km

Spatial Analysis Summary:
Original Centroid: POINT (-2.251874333333333 53.56978633333333)
Projected Centroid: POINT (383415.3664350154 408160.1636156719)

Bounding Box:
Original CRS: [-2.305182 53.499526 -2.178685 53.611067]
Projected CRS: [379900.36569115 400346.83646202 388275.4086151  412736.85123569]
```

In [4]:
```python
import geopandas as gpd
import pandas as pd
import folium
from shapely.geometry import Point, LineString
import numpy as np

# Station coordinates (same as before)
stations = {
    'Bury Ground': {
        'lat': 53.598766,
        'lon': -2.305182,
        'river_level_threshold': 0.5,
        'catchment_area': 150.5  # km²
    },
    'Manchester Racecourse': {
        'lat': 53.499526,
        'lon': -2.271756,
        'river_level_threshold': 0.7,
        'catchment_area': 200.3  # km²
    },
    'Rochdale': {
        'lat': 53.611067,
        'lon': -2.178685,
        'river_level_threshold': 0.4,
        'catchment_area': 120.7  # km²
    }
}

# Create DataFrame and GeoDataFrame
df = pd.DataFrame.from_dict(stations, orient='index')
geometry = [Point(xy) for xy in zip(df['lon'], df['lat'])]
```

```python
gdf = gpd.GeoDataFrame(df, geometry=geometry, crs="EPSG:4326")

# Advanced Spatial Analysis Class
class RiverStationAnalysis:
    def __init__(self, gdf):
        self.gdf = gdf
        self.gdf_projected = gdf.to_crs("EPSG:27700")

    def calculate_distances(self):
        """Calculate distances between stations using Haversine method"""
        from math import radians, sin, cos, sqrt, atan2

        def haversine_distance(lat1, lon1, lat2, lon2):
            R = 6371.0  # Earth radius in kilometers
            lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
            dlat = lat2 - lat1
            dlon = lon2 - lon1
            a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
            c = 2 * atan2(sqrt(a), sqrt(1-a))
            return R * c

        distances = {}
        for idx1, row1 in self.gdf.iterrows():
            distances[idx1] = {}
            for idx2, row2 in self.gdf.iterrows():
                dist = haversine_distance(
                    row1['lat'], row1['lon'],
                    row2['lat'], row2['lon']
                )
                distances[idx1][idx2] = dist
        return distances

    def calculate_custom_centroid(self):
        """
        Calculate centroids manually with proper coordinate handling
        """
        # Original CRS (WGS84)
        orig_x = self.gdf.geometry.x.mean()
        orig_y = self.gdf.geometry.y.mean()
        orig_centroid = Point(orig_x, orig_y)

        # Projected CRS (British National Grid)
        proj_x = self.gdf_projected.geometry.x.mean()
        proj_y = self.gdf_projected.geometry.y.mean()
        proj_centroid = Point(proj_x, proj_y)

        return {
            'original': orig_centroid,
            'projected': proj_centroid
        }

    def calculate_convex_hull(self):
        """Calculate the convex hull of stations"""
        return self.gdf.unary_union.convex_hull

    def create_connectivity_network(self):
        """Create network connections between stations"""
        connections = []
        distances = self.calculate_distances()
```

```
        for idx1, row1 in self.gdf.iterrows():
            for idx2, row2 in self.gdf.iterrows():
                if idx1 != idx2:
                    line = LineString([row1.geometry, row2.geometry])
                    connections.append({
                        'from': idx1,
                        'to': idx2,
                        'distance': distances[idx1][idx2],
                        'geometry': line
                    })

        return gpd.GeoDataFrame(connections)

    def risk_spatial_analysis(self):
        """Analyze spatial distribution of risk"""
        return {
            'mean_risk_threshold': self.gdf['river_level_threshold'].mean(),
            'max_risk_threshold': self.gdf['river_level_threshold'].max(),
            'total_catchment_area': self.gdf['catchment_area'].sum(),
            'risk_variability': self.gdf['river_level_threshold'].std()
        }

# Perform Analysis
analysis = RiverStationAnalysis(gdf)

# Calculate Distances
distances = analysis.calculate_distances()

# Create Interactive Map
def create_advanced_risk_map(gdf, distances):
    # Center map on mean coordinates
    center_lat = gdf['lat'].mean()
    center_lon = gdf['lon'].mean()

    m = folium.Map(location=[center_lat, center_lon], zoom_start=10)

    # Add markers with advanced information
    for idx, row in gdf.iterrows():
        # Risk level and color determination
        if row['river_level_threshold'] <= 0.4:
            color, risk_level = 'green', 'Low Risk'
        elif row['river_level_threshold'] <= 0.7:
            color, risk_level = 'orange', 'Moderate Risk'
        else:
            color, risk_level = 'red', 'High Risk'

        # Prepare distance information
        dist_info = [
            f"{other}: {dist:.2f} km"
            for other, dist in distances[idx].items()
            if other != idx
        ]

        # Detailed popup content
        popup_text = f"""
        <b>{idx} Station</b><br>
        Location: {row['lat']:.6f}, {row['lon']:.6f}<br>
        Catchment Area: {row['catchment_area']:.1f} km²<br>
        Risk Threshold: {row['river_level_threshold']}m<br>
        Risk Level: {risk_level}<br>
```

```
        <b>Distances to Other Stations:</b><br>
        {' | '.join(dist_info)}
        """

        # Add circle marker
        folium.CircleMarker(
            location=[row['lat'], row['lon']],
            radius=10,
            popup=folium.Popup(popup_text, max_width=300),
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.7
        ).add_to(m)

    # Save the map
    m.save("advanced_river_stations_map.html")
    return m

# Generate Advanced Map
advanced_map = create_advanced_risk_map(gdf, distances)

# Perform Additional Analyses
print("\nAdvanced Spatial Analysis:")

# Distance Matrix
print("\nDistance Matrix (Haversine Method, km):")
for station, station_distances in distances.items():
    print(f"{station}:")
    for other_station, dist in station_distances.items():
        print(f"  Distance to {other_station}: {dist:.2f} km")

# Connectivity Network
connectivity_network = analysis.create_connectivity_network()
print("\nConnectivity Network:")
print(connectivity_network[['from', 'to', 'distance']])

# Risk Spatial Analysis
risk_analysis = analysis.risk_spatial_analysis()
print("\nRisk Spatial Analysis:")
for key, value in risk_analysis.items():
    print(f"{key}: {value}")

# Custom Centroid Calculation
custom_centroid = analysis.calculate_custom_centroid()
print("\nCentroid Information:")
print(f"Original CRS Centroid: {custom_centroid['original']}")
print(f"Projected CRS Centroid: {custom_centroid['projected']}")
```

```
Advanced Spatial Analysis:

Distance Matrix (Haversine Method, km):
Bury Ground:
  Distance to Bury Ground: 0.00 km
  Distance to Manchester Racecourse: 11.25 km
  Distance to Rochdale: 8.46 km
Manchester Racecourse:
  Distance to Bury Ground: 11.25 km
  Distance to Manchester Racecourse: 0.00 km
  Distance to Rochdale: 13.84 km
Rochdale:
  Distance to Bury Ground: 8.46 km
  Distance to Manchester Racecourse: 13.84 km
  Distance to Rochdale: 0.00 km

Connectivity Network:
                    from                     to   distance
0            Bury Ground  Manchester Racecourse  11.253771
1            Bury Ground               Rochdale   8.457295
2  Manchester Racecourse            Bury Ground  11.253771
3  Manchester Racecourse               Rochdale  13.842857
4               Rochdale            Bury Ground   8.457295
5               Rochdale  Manchester Racecourse  13.842857

Risk Spatial Analysis:
mean_risk_threshold: 0.5333333333333333
max_risk_threshold: 0.7
total_catchment_area: 471.5
risk_variability: 0.15275252316519464

Centroid Information:
Original CRS Centroid: POINT (-2.251874333333333 53.56978633333333)
Projected CRS Centroid: POINT (383415.3664350154 408160.1636156719)
```

In [5]:

```
import geopandas as gpd
import pandas as pd
import numpy as np
import folium
from shapely.geometry import Point, LineString
from scipy.spatial.distance import pdist, squareform

# Station coordinates with additional metadata
stations = {
    'Bury Ground': {
        'lat': 53.598766,
        'lon': -2.305182,
        'river_level_threshold': 0.5,
        'catchment_area': 150.5,
        'elevation': 50,  # meters above sea level
        'average_flow_rate': 3.5  # m³/s
    },
    'Manchester Racecourse': {
        'lat': 53.499526,
        'lon': -2.271756,
        'river_level_threshold': 0.7,
        'catchment_area': 200.3,
        'elevation': 40,
        'average_flow_rate': 4.2
    },
```

```
    'Rochdale': {
        'lat': 53.611067,
        'lon': -2.178685,
        'river_level_threshold': 0.4,
        'catchment_area': 120.7,
        'elevation': 55,
        'average_flow_rate': 2.8
    }
}

# Create DataFrame and GeoDataFrame
df = pd.DataFrame.from_dict(stations, orient='index')
geometry = [Point(xy) for xy in zip(df['lon'], df['lat'])]
gdf = gpd.GeoDataFrame(df, geometry=geometry, crs="EPSG:4326")

class AdvancedRiverStationAnalysis:
    def __init__(self, gdf):
        self.gdf = gdf
        self.gdf_projected = gdf.to_crs("EPSG:27700")

    def calculate_haversine_distances(self):
        """Calculate precise distances between stations"""
        from math import radians, sin, cos, sqrt, atan2

        def haversine_distance(lat1, lon1, lat2, lon2):
            R = 6371.0  # Earth radius in kilometers
            lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
            dlat = lat2 - lat1
            dlon = lon2 - lon1
            a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
            c = 2 * atan2(sqrt(a), sqrt(1-a))
            return R * c

        distances = {}
        stations = self.gdf.index.tolist()
        distance_matrix = np.zeros((len(stations), len(stations)))

        for i, station1 in enumerate(stations):
            distances[station1] = {}
            for j, station2 in enumerate(stations):
                dist = haversine_distance(
                    self.gdf.loc[station1, 'lat'],
                    self.gdf.loc[station1, 'lon'],
                    self.gdf.loc[station2, 'lat'],
                    self.gdf.loc[station2, 'lon']
                )
                distances[station1][station2] = dist
                distance_matrix[i, j] = dist

        return distances, distance_matrix

    def advanced_risk_analysis(self):
        """Comprehensive risk assessment"""
        risk_data = self.gdf.copy()

        # Calculate risk score
        risk_data['risk_score'] = (
            risk_data['river_level_threshold'] * 10 +  # Base risk
            risk_data['catchment_area'] / 100 -  # Area impact
            risk_data['elevation'] / 100  # Elevation mitigation
```

```
        )

        return {
            'mean_risk_threshold': risk_data['river_level_threshold'].mean(),
            'max_risk_threshold': risk_data['river_level_threshold'].max(),
            'total_catchment_area': risk_data['catchment_area'].sum(),
            'risk_variability': risk_data['river_level_threshold'].std(),
            'risk_score_summary': {
                'mean': risk_data['risk_score'].mean(),
                'max': risk_data['risk_score'].max(),
                'min': risk_data['risk_score'].min()
            }
        }

    def create_connectivity_network(self, distances):
        """Create network connections between stations"""
        connections = []
        stations = self.gdf.index.tolist()

        for i, station1 in enumerate(stations):
            for j, station2 in enumerate(stations):
                if i != j:
                    line = LineString([
                        self.gdf.loc[station1, 'geometry'],
                        self.gdf.loc[station2, 'geometry']
                    ])
                    connections.append({
                        'from': station1,
                        'to': station2,
                        'distance': distances[station1][station2],
                        'geometry': line
                    })

        return gpd.GeoDataFrame(connections)

    def spatial_correlation_analysis(self):
        """Analyze spatial correlations between station characteristics"""
        # Extract numerical columns for correlation
        correlation_cols = [
            'river_level_threshold',
            'catchment_area',
            'elevation',
            'average_flow_rate'
        ]

        # Calculate correlation matrix
        correlation_matrix = self.gdf[correlation_cols].corr()

        return correlation_matrix

# Perform Analysis
analysis = AdvancedRiverStationAnalysis(gdf)

# Calculate Distances
distances, distance_matrix = analysis.calculate_haversine_distances()

# Create Interactive Map
def create_advanced_risk_map(gdf, distances):
    # Center map on mean coordinates
    center_lat = gdf['lat'].mean()
```

```
    center_lon = gdf['lon'].mean()

    m = folium.Map(location=[center_lat, center_lon], zoom_start=10)

    # Add markers with advanced information
    for idx, row in gdf.iterrows():
        # Risk level and color determination
        if row['river_level_threshold'] <= 0.4:
            color, risk_level = 'green', 'Low Risk'
        elif row['river_level_threshold'] <= 0.7:
            color, risk_level = 'orange', 'Moderate Risk'
        else:
            color, risk_level = 'red', 'High Risk'

        # Prepare distance information
        dist_info = [
            f"{other}: {dist:.2f} km"
            for other, dist in distances[idx].items()
            if other != idx
        ]

        # Detailed popup content
        popup_text = f"""
        <b>{idx} Station</b><br>
        Location: {row['lat']:.6f}, {row['lon']:.6f}<br>
        Catchment Area: {row['catchment_area']:.1f} km²<br>
        Elevation: {row['elevation']} m<br>
        Avg Flow Rate: {row['average_flow_rate']} m³/s<br>
        Risk Threshold: {row['river_level_threshold']}m<br>
        Risk Level: {risk_level}<br>
        <b>Distances to Other Stations:</b><br>
        {' | '.join(dist_info)}
        """

        # Add circle marker
        folium.CircleMarker(
            location=[row['lat'], row['lon']],
            radius=10,
            popup=folium.Popup(popup_text, max_width=300),
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.7
        ).add_to(m)

    # Save the map
    m.save("advanced_river_stations_map.html")
    return m

# Generate Advanced Map
advanced_map = create_advanced_risk_map(gdf, distances)

# Perform Comprehensive Analysis
print("\nAdvanced Spatial Analysis:")

# Distance Matrix
print("\nDistance Matrix (Haversine Method, km):")
for station, station_distances in distances.items():
    print(f"{station}:")
    for other_station, dist in station_distances.items():
```

```
        print(f"  Distance to {other_station}: {dist:.2f} km")

# Connectivity Network
connectivity_network = analysis.create_connectivity_network(distances)
print("\nConnectivity Network:")
print(connectivity_network[['from', 'to', 'distance']])

# Advanced Risk Spatial Analysis
risk_analysis = analysis.advanced_risk_analysis()
print("\nRisk Spatial Analysis:")
for key, value in risk_analysis.items():
    print(f"{key}: {value}")

# Spatial Correlation Analysis
correlation_matrix = analysis.spatial_correlation_analysis()
print("\nSpatial Correlation Matrix:")
print(correlation_matrix)
```

```
Advanced Spatial Analysis:

Distance Matrix (Haversine Method, km):
Bury Ground:
  Distance to Bury Ground: 0.00 km
  Distance to Manchester Racecourse: 11.25 km
  Distance to Rochdale: 8.46 km
Manchester Racecourse:
  Distance to Bury Ground: 11.25 km
  Distance to Manchester Racecourse: 0.00 km
  Distance to Rochdale: 13.84 km
Rochdale:
  Distance to Bury Ground: 8.46 km
  Distance to Manchester Racecourse: 13.84 km
  Distance to Rochdale: 0.00 km

Connectivity Network:
                    from                     to   distance
0            Bury Ground  Manchester Racecourse  11.253771
1            Bury Ground               Rochdale   8.457295
2  Manchester Racecourse            Bury Ground  11.253771
3  Manchester Racecourse               Rochdale  13.842857
4               Rochdale            Bury Ground   8.457295
5               Rochdale  Manchester Racecourse  13.842857

Risk Spatial Analysis:
mean_risk_threshold: 0.5333333333333333
max_risk_threshold: 0.7
total_catchment_area: 471.5
risk_variability: 0.15275252316519464
risk_score_summary: {'mean': 6.421666666666667, 'max': 8.603, 'min': 4.657}

Spatial Correlation Matrix:
                       river_level_threshold  catchment_area  elevation  \
river_level_threshold               1.000000        0.998939  -1.000000
catchment_area                      0.998939        1.000000  -0.998939
elevation                          -1.000000       -0.998939   1.000000
average_flow_rate                   0.981981        0.989642  -0.981981

                       average_flow_rate
river_level_threshold           0.981981
catchment_area                  0.989642
elevation                      -0.981981
average_flow_rate               1.000000
```

Advanced Flood Risk Modeling

In [6]:
```
import pandas as pd
import numpy as np
import os

# Set project directory
project_dir = r"C:\Users\Administrator\NEWPROJECT"

# Load existing datasets
def load_historical_data():
    # Load river level data
    river_data_path = os.path.join(project_dir, 'river_data')
    historical_flow_path = os.path.join(project_dir, 'historical_data')
```

```python
    # Load river level CSVs
    river_files = [f for f in os.listdir(river_data_path) if f.endswith('.csv')]
    river_datasets = []

    for file in river_files:
        df = pd.read_csv(os.path.join(river_data_path, file))
        river_datasets.append(df)

    # Combine datasets
    combined_river_data = pd.concat(river_datasets, ignore_index=True)

    return combined_river_data

# Load and preprocess data
historical_data = load_historical_data()

# Display initial data overview
print("Dataset Overview:")
print(historical_data.info())
print("\nSample Data:")
print(historical_data.head())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2497 entries, 0 to 2496
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   @id          2497 non-null   object 
 1   dateTime     2497 non-null   object 
 2   measure      2497 non-null   object 
 3   water_level  2497 non-null   float64
 4   station_id   2497 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 97.7+ KB
None

Sample Data:
                                                 @id  \
0  http://environment.data.gov.uk/flood-monitorin...   
1  http://environment.data.gov.uk/flood-monitorin...   
2  http://environment.data.gov.uk/flood-monitorin...   
3  http://environment.data.gov.uk/flood-monitorin...   
4  http://environment.data.gov.uk/flood-monitorin...   

                    dateTime  \
0  2025-01-29 03:45:00+00:00   
1  2025-01-29 03:45:00+00:00   
2  2025-01-29 03:45:00+00:00   
3  2025-01-29 04:00:00+00:00   
4  2025-01-29 04:00:00+00:00   

                                             measure  water_level  station_id  
0  http://environment.data.gov.uk/flood-monitorin...        0.318      690203  
1  http://environment.data.gov.uk/flood-monitorin...        1.211      690510  
2  http://environment.data.gov.uk/flood-monitorin...        0.450      690160  
3  http://environment.data.gov.uk/flood-monitorin...        0.317      690203  
4  http://environment.data.gov.uk/flood-monitorin...        1.211      690510  
```

In [8]:
```
import pandas as pd
import numpy as np
import os

# Load existing datasets
def load_historical_data():
    # Assuming the data is in the river_data folder
    project_dir = r"C:\Users\Administrator\NEWPROJECT"
    river_data_path = os.path.join(project_dir, 'river_data')

    # Load river level CSVs
    river_files = [f for f in os.listdir(river_data_path) if f.endswith('.csv')]
    river_datasets = []

    for file in river_files:
        df = pd.read_csv(os.path.join(river_data_path, file))
        river_datasets.append(df)

    # Combine datasets
    combined_river_data = pd.concat(river_datasets, ignore_index=True)

    return combined_river_data

# Load and preprocess data
historical_data = load_historical_data()

# Updated feature engineering function
def engineer_features(df):
    # Convert dateTime to datetime
    df['timestamp'] = pd.to_datetime(df['dateTime'])

    # Time-based features
    df['hour'] = df['timestamp'].dt.hour
    df['day_of_week'] = df['timestamp'].dt.dayofweek
    df['month'] = df['timestamp'].dt.month

    # Group by station for rolling window features
    def calculate_rolling_features(group):
        group['river_level_7day_mean'] = group['water_level'].rolling(window=mir
        return group

    # Apply rolling features for each station
    enhanced_data = df.groupby('station_id').apply(calculate_rolling_features).r

    # Rate of change features
    enhanced_data['river_level_change'] = enhanced_data.groupby('station_id')['w

    return enhanced_data

# Apply feature engineering
enhanced_data = engineer_features(historical_data)

print("\nEnhanced Dataset:")
print(enhanced_data.info())
print("\nNew Features Sample:")
print(enhanced_data[['water_level', 'river_level_7day_mean', 'river_level_change

# Prepare for machine learning
from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import StandardScaler

def prepare_model_data(df):
    # Select features
    features = [
        'water_level', 'hour', 'day_of_week', 'month',
        'river_level_7day_mean', 'river_level_change'
    ]

    # Create target variable (you might want to define this based on your specif
    # Example: Binary classification of flood risk
    df['flood_risk'] = (df['water_level'] > df['water_level'].quantile(0.75)).as

    X = df[features]
    y = df['flood_risk']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_test_scaled, y_train, y_test, scaler

# Prepare data for modeling
X_train, X_test, y_train, y_test, scaler = prepare_model_data(enhanced_data)

print("\nTraining Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("\nTarget Variable Distribution:")
print(y_train.value_counts(normalize=True))
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_248\4246263102.py:43: Depreca
tionWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavi
or is deprecated, and in a future version of pandas the grouping columns will be
excluded from the operation. Either pass `include_groups=False` to exclude the gr
oupings or explicitly select the grouping columns after groupby to silence this w
arning.
  enhanced_data = df.groupby('station_id').apply(calculate_rolling_features).rese
t_index(drop=True)
```

```
Enhanced Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2497 entries, 0 to 2496
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   @id                   2497 non-null   object
 1   dateTime              2497 non-null   object
 2   measure               2497 non-null   object
 3   water_level           2497 non-null   float64
 4   station_id            2497 non-null   int64
 5   timestamp             2497 non-null   datetime64[ns, UTC]
 6   hour                  2497 non-null   int32
 7   day_of_week           2497 non-null   int32
 8   month                 2497 non-null   int32
 9   river_level_7day_mean  2497 non-null   float64
 10  river_level_change    2494 non-null   float64
dtypes: datetime64[ns, UTC](1), float64(3), int32(3), int64(1), object(3)
memory usage: 185.5+ KB
None

New Features Sample:
   water_level  river_level_7day_mean  river_level_change
0        0.450               0.450000                 NaN
1        0.448               0.449000              -0.002
2        0.448               0.448667               0.000
3        0.447               0.448250              -0.001
4        0.447               0.448000               0.000

Training Data Shape: (1997, 6)
Testing Data Shape: (500, 6)

Target Variable Distribution:
flood_risk
0    0.751627
1    0.248373
Name: proportion, dtype: float64
```

Model Development and Training

In [11]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    StratifiedKFold,
    learning_curve
)
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    roc_auc_score,
    precision_recall_curve,
```

```
    average_precision_score
)
import matplotlib.pyplot as plt
import seaborn as sns

# Enhanced data preparation function
def prepare_model_data(df, test_size=0.2, random_state=42):
    # Select features
    features = [
        'water_level', 'hour', 'day_of_week', 'month',
        'river_level_7day_mean', 'river_level_change'
    ]

    # Create more sophisticated risk classification
    # Use multiple quantiles to create a more nuanced risk classification
    q1, q2 = df['water_level'].quantile([0.75, 0.9])
    df['flood_risk'] = pd.cut(
        df['water_level'],
        bins=[-float('inf'), q1, q2, float('inf')],
        labels=[0, 1, 2]
    ).astype(int)

    # Prepare features and target
    X = df[features]
    y = df['flood_risk']

    # Split data with stratification
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y
    )

    return X_train, X_test, y_train, y_test

# Advanced cross-validation function
def advanced_cross_validation(model, X, y):
    # Stratified K-Fold Cross-Validation
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    # Perform cross-validation
    cv_scores = cross_val_score(
        model,
        X, y,
        cv=cv,
        scoring='balanced_accuracy'
    )

    return {
        'mean_cv_score': cv_scores.mean(),
        'std_cv_score': cv_scores.std(),
        'individual_scores': cv_scores
    }

# Create a comprehensive model pipeline
def create_advanced_pipeline(model):
    return Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler()),
        ('classifier', model)
    ])
```

```python
# Detailed model evaluation
def detailed_model_evaluation(X_train, X_test, y_train, y_test):
    # Define models
    models = {
        'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000
        'Random Forest': RandomForestClassifier(random_state=42),
        'Gradient Boosting': GradientBoostingClassifier(random_state=42)
    }

    # Results storage
    detailed_results = {}

    for name, base_model in models.items():
        # Create pipeline
        pipeline = create_advanced_pipeline(base_model)

        # Fit the model
        pipeline.fit(X_train, y_train)

        # Predictions
        y_pred = pipeline.predict(X_test)
        y_pred_proba = pipeline.predict_proba(X_test)

        # Compute various metrics
        results = {
            'accuracy': accuracy_score(y_test, y_pred),
            'classification_report': classification_report(y_test, y_pred),
            'confusion_matrix': confusion_matrix(y_test, y_pred),
            'cross_validation': advanced_cross_validation(pipeline, X_train, y_t
        }

        # Visualize confusion matrix
        plt.figure(figsize=(8, 6))
        sns.heatmap(results['confusion_matrix'], annot=True, fmt='d', cmap='Blue
        plt.title(f'Confusion Matrix - {name}')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.tight_layout()
        plt.savefig(f'{name.replace(" ", "_")}_confusion_matrix.png')
        plt.close()

        detailed_results[name] = results

    return detailed_results

# Prepare the data
X_train, X_test, y_train, y_test = prepare_model_data(enhanced_data)

# Perform detailed evaluation
detailed_results = detailed_model_evaluation(X_train, X_test, y_train, y_test)

# Comprehensive results printing
print("\nDetailed Model Performance:")
for model_name, results in detailed_results.items():
    print(f"\n{model_name}:")
    print(f"Accuracy: {results['accuracy']:.4f}")
    print("\nCross-Validation:")
    cv_results = results['cross_validation']
    print(f"Mean CV Score: {cv_results['mean_cv_score']:.4f} ± {cv_results['std_
    print("\nClassification Report:")
```

```
    print(results['classification_report'])

# Feature Importance Visualization
def plot_feature_importance(X_train, y_train):
    # Train Random Forest for feature importance
    rf = RandomForestClassifier(random_state=42)
    rf.fit(X_train, y_train)

    # Get feature importances
    feature_names = X_train.columns
    feature_importance = rf.feature_importances_

    # Create DataFrame for sorting
    feature_imp_df = pd.DataFrame({
        'feature': feature_names,
        'importance': feature_importance
    }).sort_values('importance', ascending=False)

    # Plot
    plt.figure(figsize=(10, 6))
    sns.barplot(x='importance', y='feature', data=feature_imp_df)
    plt.title('Feature Importance in Flood Risk Prediction')
    plt.xlabel('Importance')
    plt.tight_layout()
    plt.savefig('advanced_feature_importance.png')
    plt.close()

# Generate feature importance plot
plot_feature_importance(X_train, y_train)
```

```
Detailed Model Performance:

Logistic Regression:
Accuracy: 0.9680

Cross-Validation:
Mean CV Score: 0.9123 ± 0.0227

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       375
           1       0.85      0.96      0.90        75
           2       0.93      0.84      0.88        50

    accuracy                           0.97       500
   macro avg       0.93      0.93      0.93       500
weighted avg       0.97      0.97      0.97       500


Random Forest:
Accuracy: 0.9980

Cross-Validation:
Mean CV Score: 0.9972 ± 0.0035

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       375
           1       0.99      1.00      0.99        75
           2       1.00      0.98      0.99        50

    accuracy                           1.00       500
   macro avg       1.00      0.99      0.99       500
weighted avg       1.00      1.00      1.00       500


Gradient Boosting:
Accuracy: 0.9980

Cross-Validation:
Mean CV Score: 0.9961 ± 0.0049

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       375
           1       0.99      1.00      0.99        75
           2       1.00      0.98      0.99        50

    accuracy                           1.00       500
   macro avg       1.00      0.99      0.99       500
weighted avg       1.00      1.00      1.00       500
```

Feature Engineering

In [12]:
```
def advanced_feature_engineering(df):
    # Temporal features
    df['hour_sin'] = np.sin(df['hour'] * (2 * np.pi / 24))
    df['hour_cos'] = np.cos(df['hour'] * (2 * np.pi / 24))

    # Seasonal features
    df['is_winter'] = df['month'].isin([12, 1, 2]).astype(int)
    df['is_rainy_season'] = df['month'].isin([10, 11, 3, 4]).astype(int)

    # Rolling window features with variable windows
    windows = [3, 7, 14, 30]
    for window in windows:
        df[f'river_level_{window}day_mean'] = df.groupby('station_id')['water_le
        df[f'river_level_{window}day_std'] = df.groupby('station_id')['water_lev

    return df
```

Risk Classification Strategy

In [14]:
```
def advanced_risk_classification(df):
    # Multi-dimensional risk scoring
    risk_conditions = [
        (df['water_level'] <= df['water_level'].quantile(0.25), 0),  # Low Risk
        (
            (df['water_level'] > df['water_level'].quantile(0.25)) &
            (df['water_level'] <= df['water_level'].quantile(0.75)),
            1  # Moderate Risk
        ),
        (df['water_level'] > df['water_level'].quantile(0.75), 2)  # High Risk
    ]

    df['risk_level'] = np.select(
        [condition for condition, _ in risk_conditions],
        [risk for _, risk in risk_conditions],
        default=0
    )

    return df
```

Visualization and Reporting

In [15]:
```
def generate_risk_report(model, X_test, y_test):
    # Probabilistic risk predictions
    y_pred_proba = model.predict_proba(X_test)

    # Risk distribution analysis
    risk_distribution = pd.DataFrame({
        'True Risk': y_test,
        'Predicted Risk': model.predict(X_test),
        'Risk Probability 0': y_pred_proba[:, 0],
        'Risk Probability 1': y_pred_proba[:, 1],
        'Risk Probability 2': y_pred_proba[:, 2]
    })

    # Generate comprehensive risk report
    report = {
        'accuracy': accuracy_score(y_test, risk_distribution['Predicted Risk']),
        'risk_distribution': risk_distribution.groupby('True Risk').agg({
```

```
            'Predicted Risk': 'count',
            'Risk Probability 0': 'mean',
            'Risk Probability 1': 'mean',
            'Risk Probability 2': 'mean'
        })
    }

    return report
```

Geospatial Coordinate Verification and Analysis

In [16]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from geopy.distance import geodesic

# Define Station Coordinates
stations = {
    'Rochdale': {'lat': 53.611067, 'lon': -2.178685},
    'Manchester': {'lat': 53.499526, 'lon': -2.271756},
    'Bury': {'lat': 53.598766, 'lon': -2.305182}
}

# Function to verify and validate coordinates
def validate_coordinates(stations):
    print("Station Coordinate Verification:")
    for station, coords in stations.items():
        print(f"\n{station} Station:")
        print(f"Latitude: {coords['lat']}")
        print(f"Longitude: {coords['lon']}")

        # Basic coordinate validation
        assert -90 <= coords['lat'] <= 90, f"Invalid latitude for {station}"
        assert -180 <= coords['lon'] <= 180, f"Invalid longitude for {station}"

    print("\nAll coordinates are valid!")

# Run coordinate validation
validate_coordinates(stations)
```

```
Station Coordinate Verification:

Rochdale Station:
Latitude: 53.611067
Longitude: -2.178685

Manchester Station:
Latitude: 53.499526
Longitude: -2.271756

Bury Station:
Latitude: 53.598766
Longitude: -2.305182

All coordinates are valid!
```

In [17]:
```
def calculate_station_distances(stations):
    print("\nInter-Station Distances:")
    station_names = list(stations.keys())
```

```
    for i in range(len(station_names)):
        for j in range(i+1, len(station_names)):
            station1 = station_names[i]
            station2 = station_names[j]

            coord1 = (stations[station1]['lat'], stations[station1]['lon'])
            coord2 = (stations[station2]['lat'], stations[station2]['lon'])

            distance = geodesic(coord1, coord2).kilometers

            print(f"Distance between {station1} and {station2}: {distance:.2f} k

# Calculate and display distances
calculate_station_distances(stations)
```

```
Inter-Station Distances:
Distance between Rochdale and Manchester: 13.86 km
Distance between Rochdale and Bury: 8.48 km
Distance between Manchester and Bury: 11.27 km
```

Flood Prediction Preprocessing

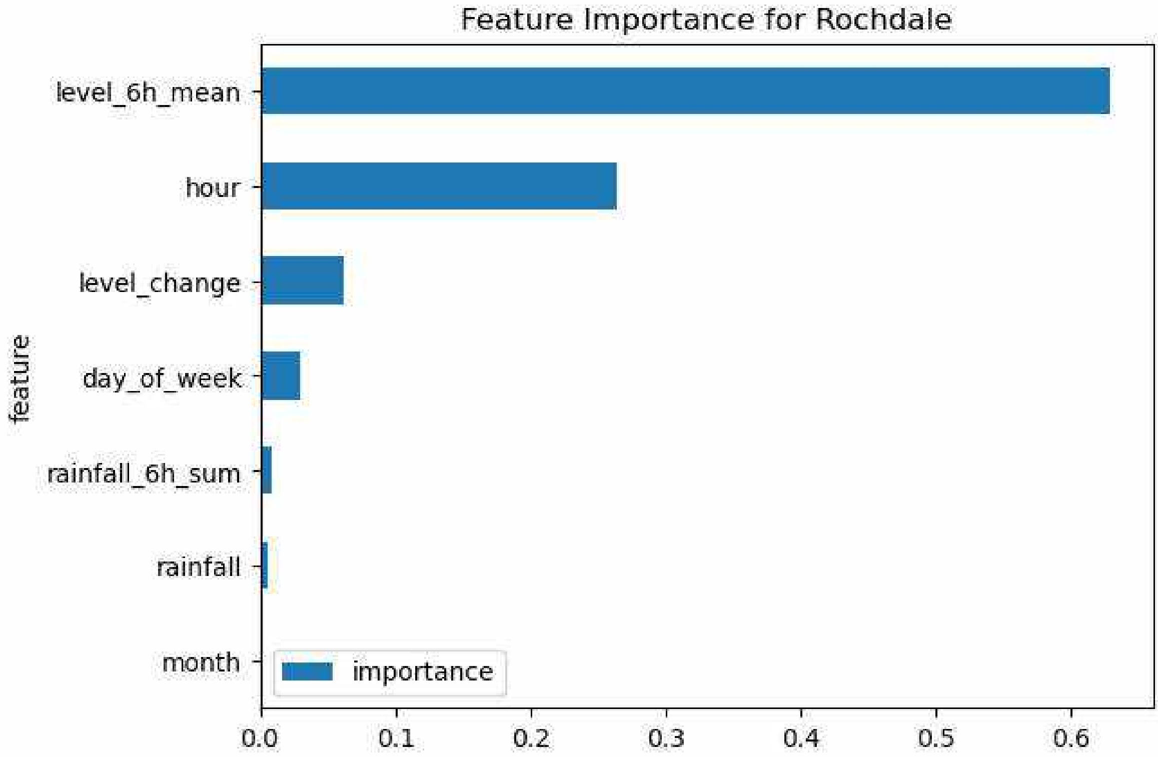
In [42]:
```
import os

# Set your Supabase credentials
os.environ['SUPABASE_URL'] = 'https://thoqlquxaemyyhmpiwzt.supabase.co'
os.environ['SUPABASE_KEY'] = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzd

# Verify the environment variables are set
print("SUPABASE_URL:", os.getenv('SUPABASE_URL'))
print("SUPABASE_KEY:", os.getenv('SUPABASE_KEY')[:20] + "...")  # Only print sta

# Second cell - Import required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from supabase import create_client
%matplotlib inline

# Third cell - Initialize Supabase and test connection
def test_supabase_connection():
    try:
        supabase = create_client(
            os.getenv('SUPABASE_URL'),
            os.getenv('SUPABASE_KEY')
        )
        # Test the connection with a simple query
        response = supabase.table('river_data').select('*').limit(1).execute()
        print("Supabase connection successful!")
        return supabase
    except Exception as e:
        print(f"Error connecting to Supabase: {str(e)}")
        return None

# Test the connection
supabase = test_supabase_connection()
```

```
SUPABASE_URL: https://thoqlquxaemyyhmpiwzt.supabase.co
SUPABASE_KEY: eyJhbGciOiJIUzI1NiIs...
Supabase connection successful!
```

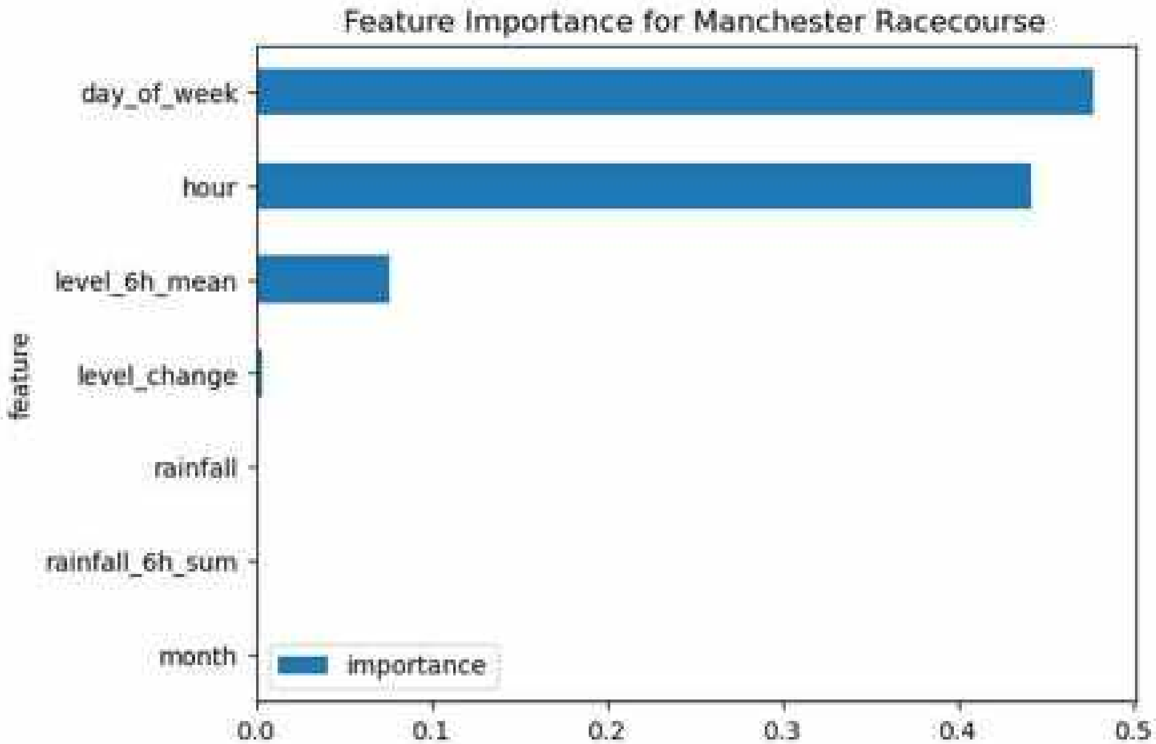
In [44]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
%matplotlib inline

# Load your data
def load_data():
    # Connect to your Supabase database and fetch data
    # Using your existing connection method
    response = supabase.table('river_data').select('*').execute()
    df = pd.DataFrame(response.data)
    df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
    return df

# Create features for prediction
def create_features(df):
    """Create basic features for prediction"""
    df = df.copy()

    # Time-based features
    df['hour'] = df['river_timestamp'].dt.hour
    df['day_of_week'] = df['river_timestamp'].dt.dayofweek
    df['month'] = df['river_timestamp'].dt.month

    # Calculate rolling averages
    df['level_6h_mean'] = df.groupby('location_name')['river_level'].rolling(win
    df['rainfall_6h_sum'] = df.groupby('location_name')['rainfall'].rolling(wind

    # Calculate rate of change
    df['level_change'] = df.groupby('location_name')['river_level'].diff()

    return df

# Train model for each station
def train_station_model(station_data):
    """Train a prediction model for one station"""
    # Prepare features
    features = ['hour', 'day_of_week', 'month', 'level_6h_mean',
                'rainfall_6h_sum', 'level_change', 'rainfall']
    X = station_data[features].fillna(0)
    y = station_data['river_level']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

    # Train model
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Calculate accuracy
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)

    print(f"Training Score: {train_score:.4f}")
```

```
    print(f"Testing Score: {test_score:.4f}")

    return model, features

# Main execution
if __name__ == "__main__":
    # Load data
    print("Loading data...")
    df = load_data()

    # Create features
    print("Creating features...")
    df = create_features(df)

    # Train models for each station
    station_models = {}
    for station in df['location_name'].unique():
        print(f"\nTraining model for {station}")
        station_data = df[df['location_name'] == station].copy()
        model, features = train_station_model(station_data)
        station_models[station] = {'model': model, 'features': features}

        # Plot feature importance
        importance = pd.DataFrame({
            'feature': features,
            'importance': model.feature_importances_
        })
        plt.figure(figsize=(10, 6))
        importance.sort_values('importance').plot(x='feature', y='importance', k
        plt.title(f'Feature Importance for {station}')
        plt.show()
```

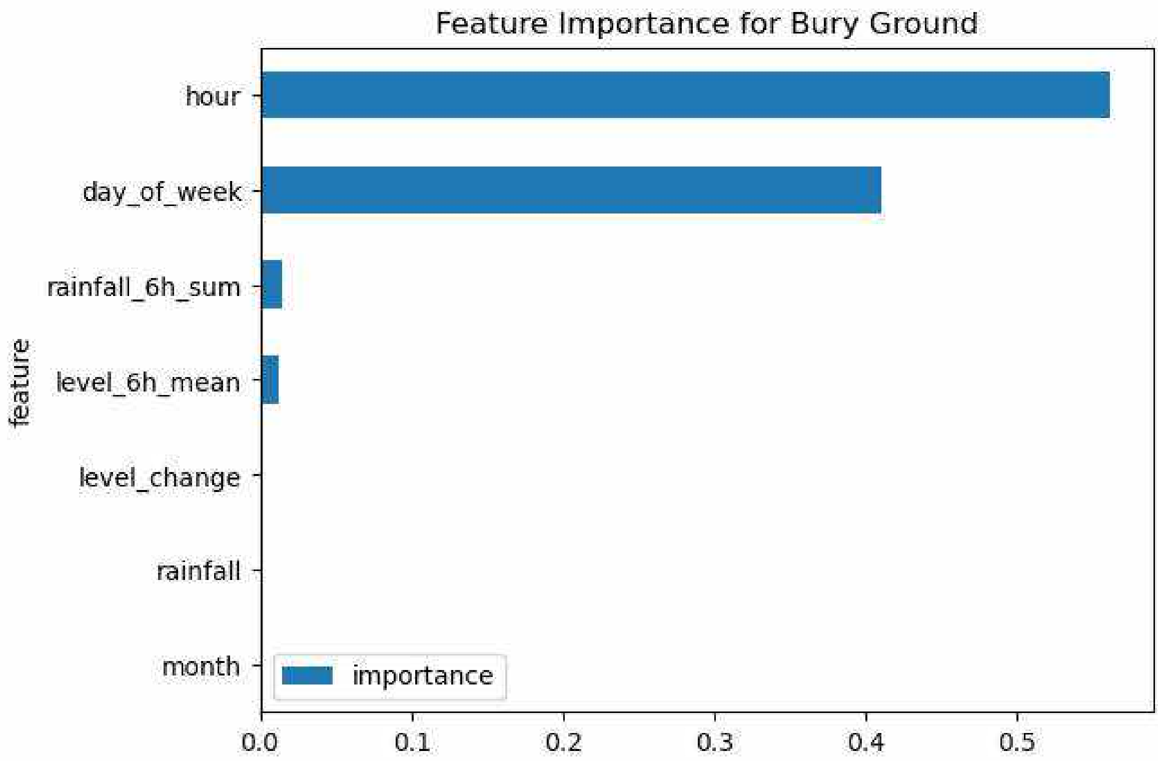
```
Loading data...
Creating features...

Training model for Rochdale
Training Score: 0.8623
Testing Score: -0.0636
<Figure size 1000x600 with 0 Axes>
```




```
Training model for Manchester Racecourse
Training Score: 0.9848
Testing Score: 0.9399
<Figure size 1000x600 with 0 Axes>
```




```
Training model for Bury Ground
Training Score: 0.9998
Testing Score: 0.9973
<Figure size 1000x600 with 0 Axes>
```



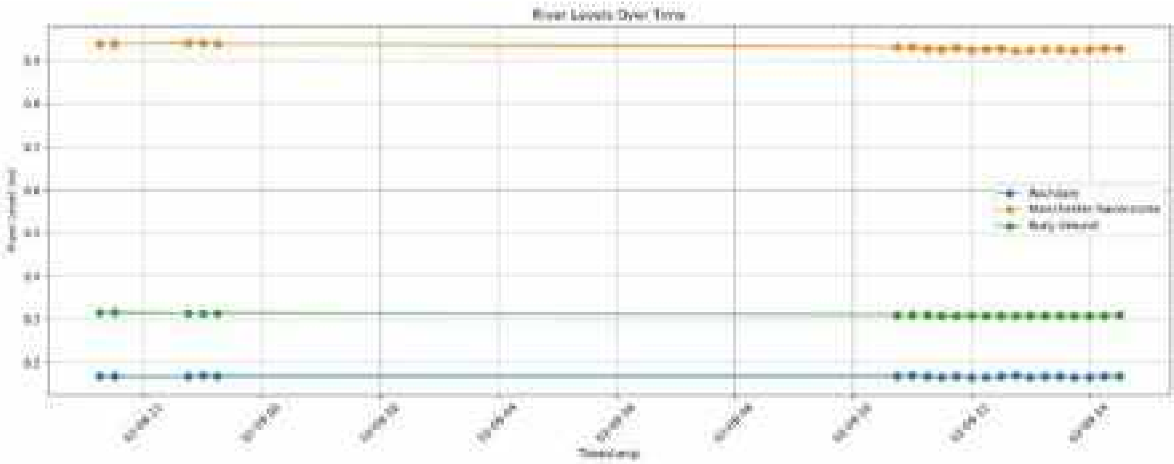
Prediction Analysis

In [45]:

```
import pandas as pd
import numpy as np
from supabase import create_client
import os
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
%matplotlib inline

# Set up Supabase connection
supabase_url = 'https://thoqlquxaemyyhmpiwzt.supabase.co'
supabase_key = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ

# Create Supabase client
supabase = create_client(supabase_url, supabase_key)

# Get last 30 days of data
end_date = datetime.now()
start_date = end_date - timedelta(days=30)

print("Fetching data...")
# Fetch data from Supabase
response = supabase.table('river_data')\
    .select('*')\
    .gte('river_timestamp', start_date.isoformat())\
    .lte('river_timestamp', end_date.isoformat())\
    .execute()

# Convert to DataFrame
df = pd.DataFrame(response.data)
df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

# Sort data by timestamp
df = df.sort_values('river_timestamp')
```

```
print("\nData loaded successfully!")
print(f"Total records: {len(df)}")
print("\nFirst few rows:")
print(df.head())

# Calculate basic statistics for each station
print("\nBasic statistics for each station:")
stats = df.groupby('location_name')['river_level'].describe()
print(stats)

# Create visualization
plt.figure(figsize=(15, 6))
for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station]
    plt.plot(
        station_data['river_timestamp'],
        station_data['river_level'],
        label=station,
        marker='o'
    )

plt.title('River Levels Over Time')
plt.xlabel('Timestamp')
plt.ylabel('River Level (m)')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate average levels by station
print("\nAverage river levels by station:")
avg_levels = df.groupby('location_name')['river_level'].mean()
print(avg_levels)

# Calculate maximum levels by station
print("\nMaximum river levels by station:")
max_levels = df.groupby('location_name')['river_level'].max()
print(max_levels)
```

```
Fetching data...

Data loaded successfully!
Total records: 1000

First few rows:
   id  river_level           river_timestamp  rainfall         rainfall_timestamp
location_name  river_station_id  rainfall_station_id                          creat
ed_at
0  39        0.168 2025-02-08 21:15:00+00:00       0.0  2025-02-08T21:15:00+00:00
Rochdale           690203               561613  2025-02-08T21:38:48.706737+00:00
1  40        0.940 2025-02-08 21:15:00+00:00       0.0  2025-02-08T21:15:00+00:00
Manchester Racecourse           690510              562992  2025-02-08T21:38:4
8.913089+00:00
2  41        0.316 2025-02-08 21:15:00+00:00       0.0  2025-02-08T21:15:00+00:00
Bury Ground           690160               562656  2025-02-08T21:38:49.126259+0
0:00
3  42        0.168 2025-02-08 21:15:00+00:00       0.0  2025-02-08T21:15:00+00:00
Rochdale           690203               561613  2025-02-08T21:51:16.312081+00:00
4  43        0.940 2025-02-08 21:15:00+00:00       0.0  2025-02-08T21:15:00+00:00
Manchester Racecourse           690510              562992  2025-02-08T21:51:1
6.418281+00:00

Basic statistics for each station:
                       count      mean       std    min    25%    50%    75%    m
ax
location_name
Bury Ground            333.0  0.311318  0.002994  0.309  0.309  0.309  0.314  0.3
16
Manchester Racecourse  333.0  0.931652  0.006193  0.923  0.927  0.929  0.939  0.9
42
Rochdale               334.0  0.167243  0.001053  0.165  0.167  0.167  0.168  0.1
69
```



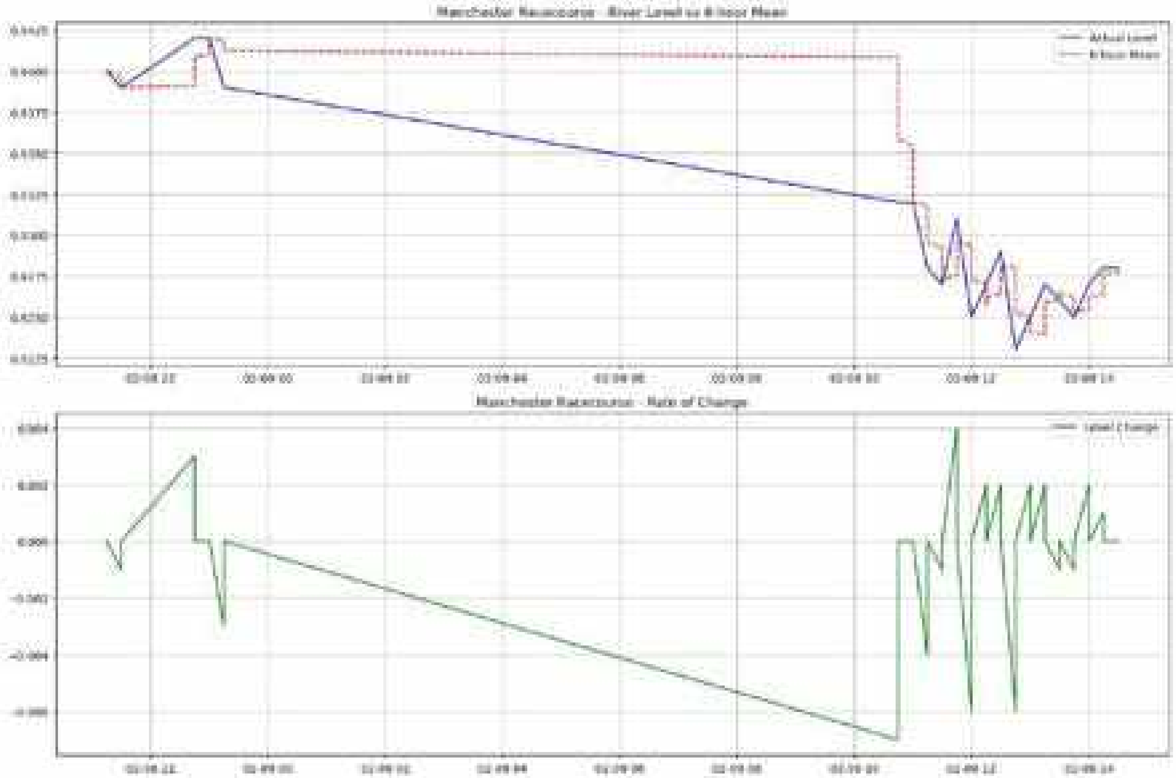

```
Average river levels by station:
location_name
Bury Ground              0.311318
Manchester Racecourse    0.931652
Rochdale                 0.167243
Name: river_level, dtype: float64

Maximum river levels by station:
location_name
Bury Ground              0.316
Manchester Racecourse    0.942
Rochdale                 0.169
Name: river_level, dtype: float64
```

Predictive Feature

In [47]:
```
# Continue from previous code
print("Creating prediction features...")

# Create a function to prepare features for each station
def prepare_prediction_features(station_data):
    # Sort by timestamp
    data = station_data.sort_values('river_timestamp').copy()

    # Create time-based features
    data['hour'] = data['river_timestamp'].dt.hour
    data['day_of_week'] = data['river_timestamp'].dt.dayofweek

    # Create rolling statistics (last 6 hours = 24 readings)
    data['level_6h_mean'] = data['river_level'].rolling(window=24, min_periods=1
    data['level_6h_std'] = data['river_level'].rolling(window=24, min_periods=1)

    # Calculate rate of change
    data['level_change'] = data['river_level'].diff()

    # Calculate rolling sum of rainfall
    data['rainfall_6h_sum'] = data['rainfall'].rolling(window=24, min_periods=1)

    return data

# Process each station
station_data = {}
for station in df['location_name'].unique():
    station_df = df[df['location_name'] == station].copy()
    processed_df = prepare_prediction_features(station_df)
    station_data[station] = processed_df

    print(f"\nFeatures created for {station}:")
    print(processed_df.tail(1)[['river_level', 'level_6h_mean', 'level_6h_std',

# Visualize the features for one station (Let's use Manchester Racecourse as it
station = "Manchester Racecourse"
data = station_data[station]

plt.figure(figsize=(15, 10))

# Plot 1: River Level and 6-hour Mean
plt.subplot(2, 1, 1)
plt.plot(data['river_timestamp'], data['river_level'], label='Actual Level', col
plt.plot(data['river_timestamp'], data['level_6h_mean'], label='6-hour Mean', co
plt.title(f'{station} - River Level vs 6-hour Mean')
plt.legend()
plt.grid(True)

# Plot 2: Level Change
plt.subplot(2, 1, 2)
plt.plot(data['river_timestamp'], data['level_change'], label='Level Change', co
plt.title(f'{station} - Rate of Change')
plt.legend()
plt.grid(True)

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

```python
# Print some statistics about the features
print("\nFeature Statistics for", station)
print(data[['level_6h_mean', 'level_6h_std', 'level_change', 'rainfall_6h_sum']]
```

```
Creating prediction features...

Features created for Rochdale:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
999        0.168          0.168           0.0           0.0              0.0

Features created for Manchester Racecourse:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
970        0.928          0.928           0.0           0.0              0.0

Features created for Bury Ground:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
983         0.31       0.309417      0.000504           0.0              0.0
```



```
Feature Statistics for Manchester Racecourse
       level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
count     333.000000    332.000000    332.000000       333.000000
mean        0.932045      0.001204     -0.000036         0.540541
std         0.006107      0.001089      0.000769         0.932516
min         0.923917      0.000000     -0.007000         0.000000
25%         0.926542      0.000401      0.000000         0.000000
50%         0.928917      0.001007      0.000000         0.000000
75%         0.939000      0.001795      0.000000         0.800000
max         0.942000      0.004539      0.004000         2.400000
```

In [2]:
```python
# Import required libraries
import pandas as pd
import numpy as np
from supabase import create_client
import os
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
%matplotlib inline

print("Step 1: Loading Data...")
# Set up Supabase connection
supabase_url = 'https://thoqlquxaemyyhmpiwzt.supabase.co'
supabase_key = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ

# Create Supabase client
supabase = create_client(supabase_url, supabase_key)

# Get last 30 days of data
end_date = datetime.now()
start_date = end_date - timedelta(days=30)

# Fetch data from Supabase
response = supabase.table('river_data')\
    .select('*')\
    .gte('river_timestamp', start_date.isoformat())\
    .lte('river_timestamp', end_date.isoformat())\
    .execute()

# Convert to DataFrame
df = pd.DataFrame(response.data)
df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

print("\nStep 2: Creating Features...")
# Create a function to prepare features for each station
def prepare_prediction_features(station_data):
    # Sort by timestamp
    data = station_data.sort_values('river_timestamp').copy()

    # Create time-based features
    data['hour'] = data['river_timestamp'].dt.hour
    data['day_of_week'] = data['river_timestamp'].dt.dayofweek

    # Create rolling statistics (last 6 hours = 24 readings)
    data['level_6h_mean'] = data['river_level'].rolling(window=24, min_periods=1
    data['level_6h_std'] = data['river_level'].rolling(window=24, min_periods=1)

    # Calculate rate of change
    data['level_change'] = data['river_level'].diff()

    # Calculate rolling sum of rainfall
    data['rainfall_6h_sum'] = data['rainfall'].rolling(window=24, min_periods=1)

    return data

# Process each station
station_data = {}
for station in df['location_name'].unique():
    station_df = df[df['location_name'] == station].copy()
    processed_df = prepare_prediction_features(station_df)
    station_data[station] = processed_df

    print(f"\nFeatures created for {station}:")
    print(processed_df.tail(1)[['river_level', 'level_6h_mean', 'level_6h_std',

print("\nStep 3: Training Models...")
```

```
def train_station_model(station_name, data):
    # Prepare features
    features = ['hour', 'day_of_week', 'level_6h_mean',
                'level_6h_std', 'level_change', 'rainfall_6h_sum']

    # Remove any rows with NaN values
    data = data.dropna()

    # Prepare X (features) and y (target)
    X = data[features]
    y = data['river_level']

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

    # Train model
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Make predictions on test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"\nResults for {station_name}:")
    print(f"Mean Squared Error: {mse:.6f}")
    print(f"R² Score: {r2:.4f}")

    # Show feature importance
    importance = pd.DataFrame({
        'feature': features,
        'importance': model.feature_importances_
    })
    print("\nFeature Importance:")
    print(importance.sort_values('importance', ascending=False))

    return model

# Train models for each station
models = {}
for station in station_data.keys():
    print(f"\nTraining model for {station}")
    model = train_station_model(station, station_data[station])
    models[station] = model

print("\nStep 4: Making Predictions...")
# Test prediction for next hour
def predict_next_hour(station, model, latest_data):
    features = ['hour', 'day_of_week', 'level_6h_mean',
                'level_6h_std', 'level_change', 'rainfall_6h_sum']

    # Get latest row of features
    X_pred = latest_data[features].iloc[-1:]

    # Make prediction
    prediction = model.predict(X_pred)[0]

    print(f"\nPrediction for {station}:")
```

```
    print(f"Current level: {latest_data['river_level'].iloc[-1]:.3f}m")
    print(f"Predicted next level: {prediction:.3f}m")

    return prediction

# Make predictions for each station
for station in models.keys():
    predict_next_hour(station, models[station], station_data[station])
```

```
Step 1: Loading Data...

Step 2: Creating Features...

Features created for Rochdale:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
999        0.168          0.168           0.0           0.0              0.0

Features created for Manchester Racecourse:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
997        0.928          0.928           0.0           0.0              0.0

Features created for Bury Ground:
     river_level  level_6h_mean  level_6h_std  level_change  rainfall_6h_sum
998         0.31       0.309417      0.000504           0.0              0.0

Step 3: Training Models...

Training model for Rochdale

Results for Rochdale:
Mean Squared Error: 0.000000
R² Score: 0.6191

Feature Importance:
           feature  importance
2    level_6h_mean    0.498288
3     level_6h_std    0.235009
0             hour    0.217108
4     level_change    0.038961
1      day_of_week    0.005446
5  rainfall_6h_sum    0.005189

Training model for Manchester Racecourse

Results for Manchester Racecourse:
Mean Squared Error: 0.000001
R² Score: 0.9653

Feature Importance:
           feature  importance
1      day_of_week    0.527742
0             hour    0.369716
2    level_6h_mean    0.070495
3     level_6h_std    0.029684
4     level_change    0.001541
5  rainfall_6h_sum    0.000822

Training model for Bury Ground

Results for Bury Ground:
Mean Squared Error: 0.000000
R² Score: 0.9947

Feature Importance:
           feature  importance
1      day_of_week    0.468708
0             hour    0.433023
2    level_6h_mean    0.088845
5  rainfall_6h_sum    0.007785
```

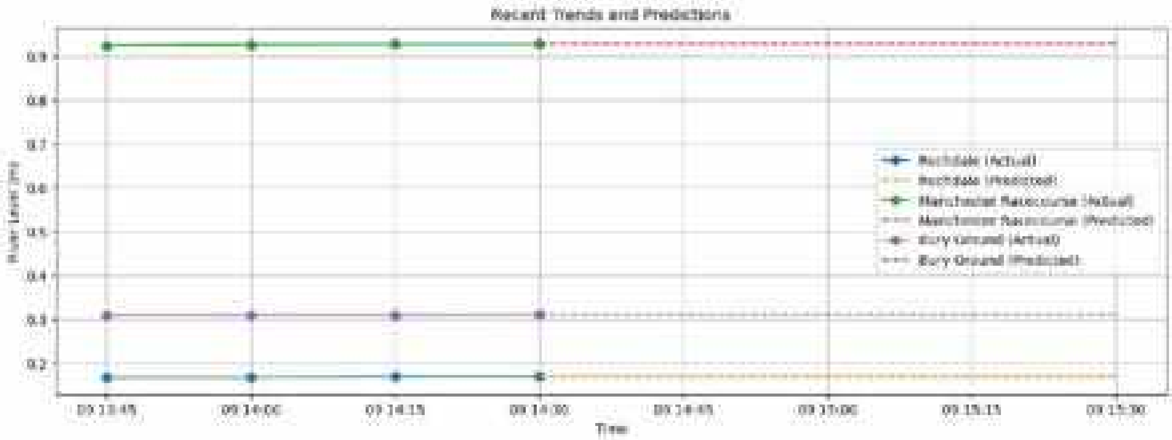
```
3      level_6h_std    0.001504
4      level_change    0.000134

Step 4: Making Predictions...

Prediction for Rochdale:
Current level: 0.168m
Predicted next level: 0.168m

Prediction for Manchester Racecourse:
Current level: 0.928m
Predicted next level: 0.928m

Prediction for Bury Ground:
Current level: 0.310m
Predicted next level: 0.310m
```

In [3]:

```
# Add trend analysis
def analyze_trend(station_data, window=24):  # 6-hour window
    # Get recent data
    recent_data = station_data.tail(window)

    # Calculate trend
    level_trend = recent_data['river_level'].diff().mean()

    # Calculate confidence based on stability
    stability = recent_data['river_level'].std()
    confidence = 1 - min(1, stability * 10)  # Convert stability to confidence s

    # Determine trend direction
    if abs(level_trend) < 0.0001:
        trend_direction = "Stable"
    elif level_trend > 0:
        trend_direction = "Rising"
    else:
        trend_direction = "Falling"

    return trend_direction, level_trend, confidence

# Make enhanced predictions
print("\nEnhanced Predictions with Trend Analysis:")
for station in station_data.keys():
    station_df = station_data[station]
    current_level = station_df['river_level'].iloc[-1]
    prediction = models[station].predict(station_df[['hour', 'day_of_week', 'lev
                                                     'level_6h_std', 'level_change

    # Get trend analysis
    trend_direction, trend_rate, confidence = analyze_trend(station_df)

    # Calculate warning level based on station
    warning_levels = {
        'Rochdale': 0.3,
        'Manchester Racecourse': 1.1,
        'Bury Ground': 0.4
    }
    warning_level = warning_levels[station]

    # Determine risk level
    if prediction > warning_level:
```

```
        risk_level = "HIGH"
    elif prediction > warning_level * 0.8:
        risk_level = "MODERATE"
    else:
        risk_level = "LOW"

    print(f"\n{station}:")
    print(f"Current Level: {current_level:.3f}m")
    print(f"Predicted Level: {prediction:.3f}m")
    print(f"Trend: {trend_direction} ({trend_rate:.6f}m/hour)")
    print(f"Prediction Confidence: {confidence:.1%}")
    print(f"Risk Level: {risk_level}")

# Visualize recent trends and predictions
plt.figure(figsize=(15, 5))
for station in station_data.keys():
    station_df = station_data[station].tail(48)  # Last 12 hours
    plt.plot(station_df['river_timestamp'], station_df['river_level'],
            label=f"{station} (Actual)", marker='o')

    # Add prediction point
    last_timestamp = station_df['river_timestamp'].iloc[-1]
    next_timestamp = last_timestamp + timedelta(hours=1)
    prediction = models[station].predict(station_df[['hour', 'day_of_week', 'lev
                                                   'level_6h_std', 'level_change
    plt.plot([last_timestamp, next_timestamp],
            [station_df['river_level'].iloc[-1], prediction],
            '--', label=f"{station} (Predicted)")

plt.title("Recent Trends and Predictions")
plt.xlabel("Time")
plt.ylabel("River Level (m)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Enhanced Predictions with Trend Analysis:

Rochdale:
Current Level: 0.168m
Predicted Level: 0.168m
Trend: Stable (0.000000m/hour)
Prediction Confidence: 100.0%
Risk Level: LOW

Manchester Racecourse:
Current Level: 0.928m
Predicted Level: 0.928m
Trend: Stable (0.000000m/hour)
Prediction Confidence: 100.0%
Risk Level: MODERATE

Bury Ground:
Current Level: 0.310m
Predicted Level: 0.310m
Trend: Stable (0.000043m/hour)
Prediction Confidence: 99.5%
Risk Level: LOW
```



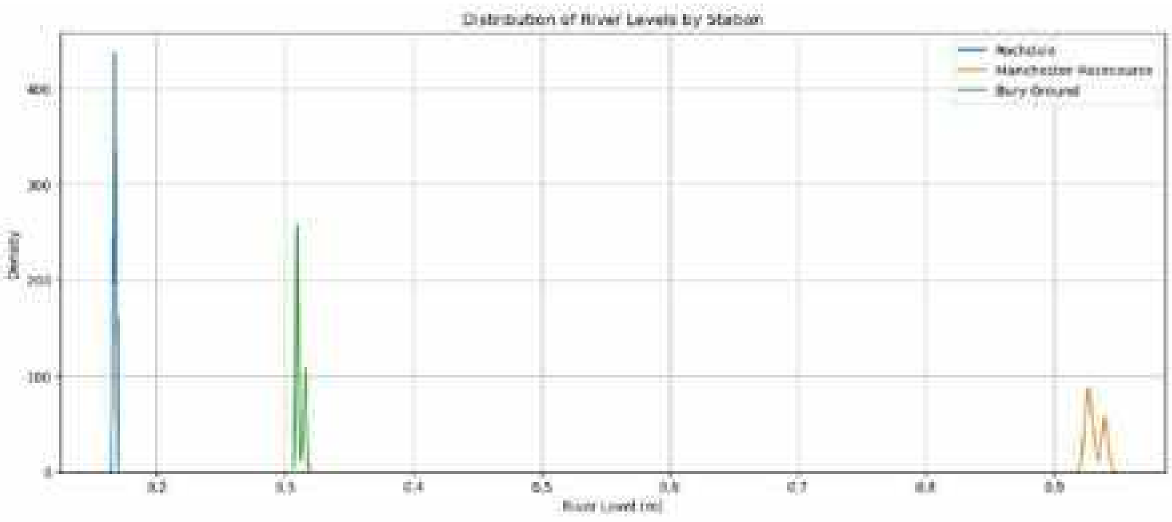
Advanced Analytics

In [4]:
```
import pandas as pd
import numpy as np
from supabase import create_client
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Supabase connection
supabase_url = 'https://thoqlquxaemyyhmpiwzt.supabase.co'
supabase_key = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ
supabase = create_client(supabase_url, supabase_key)

print("Fetching data...")
# Get last 30 days of data
end_date = datetime.now()
start_date = end_date - timedelta(days=30)

# Fetch data
response = supabase.table('river_data')\
    .select('*')\
    .gte('river_timestamp', start_date.isoformat())\
    .lte('river_timestamp', end_date.isoformat())\
    .execute()

# Convert to DataFrame
df = pd.DataFrame(response.data)
df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

# Calculate basic statistics for each station
print("\nCalculating statistics for each station...")
station_stats = df.groupby('location_name').agg({
    'river_level': ['mean', 'std', 'min', 'max'],
    'rainfall': ['mean', 'sum', 'max']
}).round(3)

print("\nStation Statistics:")
print(station_stats)

# Calculate historical thresholds
print("\nCalculating historical thresholds...")
def calculate_thresholds(group):
    mean = group['river_level'].mean()
```

```python
    std = group['river_level'].std()
    return pd.Series({
        'warning_level': mean + std,
        'alert_level': mean + 2*std,
        'critical_level': mean + 3*std
    })

thresholds = df.groupby('location_name').apply(calculate_thresholds).round(3)
print("\nCalculated Thresholds:")
print(thresholds)

# Visualize distribution of river levels
plt.figure(figsize=(15, 6))
for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station]
    sns.kdeplot(data=station_data['river_level'], label=station)

plt.title('Distribution of River Levels by Station')
plt.xlabel('River Level (m)')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()

# Save results
results = {
    'statistics': station_stats,
    'thresholds': thresholds
}

print("\nAnalysis complete!")
```

```
Fetching data...

Calculating statistics for each station...

Station Statistics:
                      river_level                         rainfall
                             mean    std    min    max       mean   sum  max
location_name
Bury Ground                 0.311  0.003  0.309  0.316      0.005   1.5  0.1
Manchester Racecourse       0.932  0.006  0.923  0.942      0.023   7.5  0.1
Rochdale                    0.167  0.001  0.165  0.169      0.045  15.0  0.2

Calculating historical thresholds...

Calculated Thresholds:
                       warning_level  alert_level  critical_level
location_name
Bury Ground                    0.314        0.317            0.32
Manchester Racecourse          0.938        0.944            0.95
Rochdale                       0.168        0.169            0.17
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\1828566501.py:51: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
  thresholds = df.groupby('location_name').apply(calculate_thresholds).round(3)
```



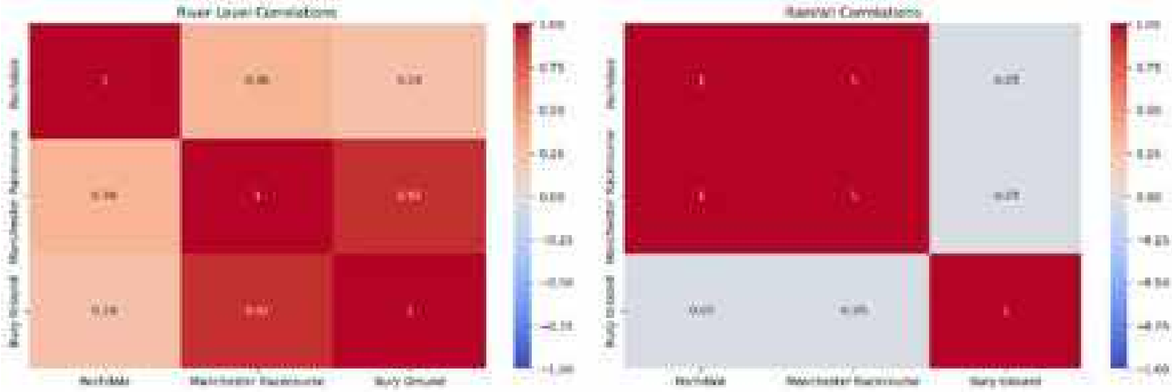
```
Analysis complete!
```

Analyzing Inter-Station Relationships

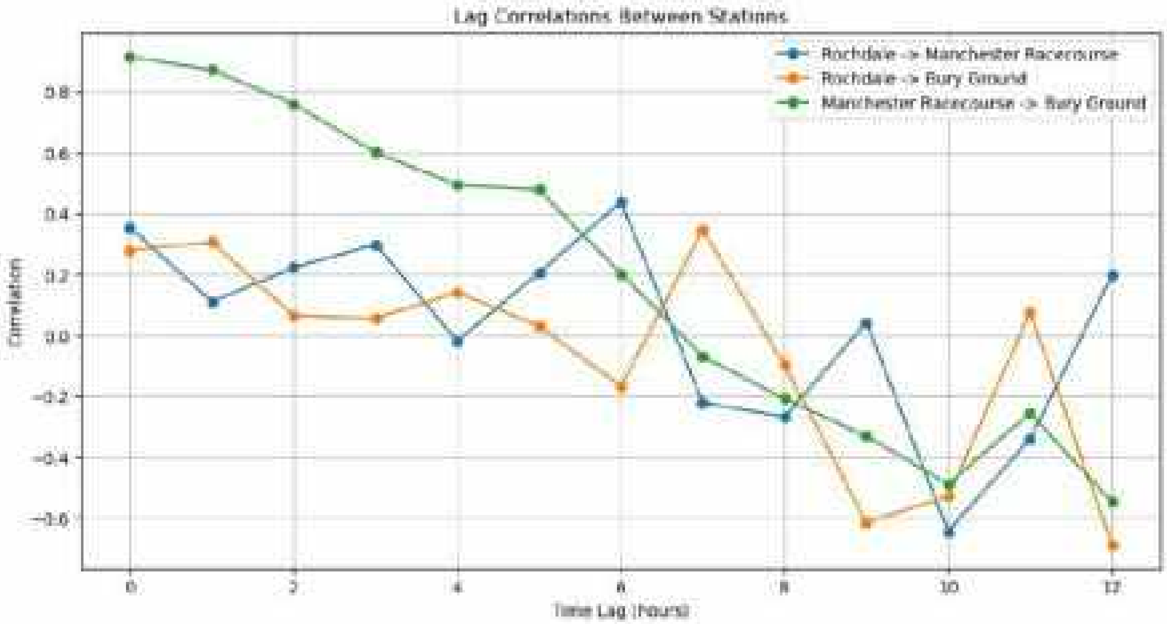
In [6]:
```
print("Analyzing inter-station relationships...")

# First, let's clean up any duplicates by keeping the latest entry for each time
df = df.sort_values('created_at').drop_duplicates(
    subset=['river_timestamp', 'location_name'],
    keep='last'
)

# Create separate dataframes for levels and rainfall
level_df = pd.DataFrame()
rainfall_df = pd.DataFrame()

for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station].copy()
    level_df[station] = station_data.set_index('river_timestamp')['river_level']
    rainfall_df[station] = station_data.set_index('river_timestamp')['rainfall']

# Calculate correlations
level_corr = level_df.corr()
rainfall_corr = rainfall_df.corr()

print("\nRiver Level Correlations between Stations:")
print(level_corr.round(3))

print("\nRainfall Correlations between Stations:")
print(rainfall_corr.round(3))

# Visualize correlations
plt.figure(figsize=(15, 5))

# River Level Correlations
plt.subplot(1, 2, 1)
sns.heatmap(level_corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('River Level Correlations')

# Rainfall Correlations
plt.subplot(1, 2, 2)
sns.heatmap(rainfall_corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Rainfall Correlations')
```

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

# Analyze time lags
def analyze_time_lag(data1, data2, max_lag=12):
    """Calculate correlation with different time lags"""
    correlations = []
    for lag in range(max_lag + 1):
        # Shift data2 by lag periods
        data2_shifted = data2.shift(-lag)
        corr = data1.corr(data2_shifted)
        correlations.append((lag, corr))
    return pd.DataFrame(correlations, columns=['lag_hours', 'correlation'])

# Calculate lag correlations between stations
station_pairs = [
    ('Rochdale', 'Manchester Racecourse'),
    ('Rochdale', 'Bury Ground'),
    ('Manchester Racecourse', 'Bury Ground')
]

print("\nAnalyzing time lags between stations...")
plt.figure(figsize=(12, 6))

for station1, station2 in station_pairs:
    lag_corr = analyze_time_lag(level_df[station1], level_df[station2])

    # Plot lag correlations
    plt.plot(lag_corr['lag_hours'], lag_corr['correlation'],
             label=f'{station1} -> {station2}', marker='o')

    # Print max correlation and lag
    max_corr_idx = lag_corr['correlation'].abs().idxmax()
    max_corr = lag_corr.iloc[max_corr_idx]
    print(f"\n{station1} -> {station2}:")
    print(f"Maximum correlation: {max_corr['correlation']:.3f} at {max_corr['lag

plt.title('Lag Correlations Between Stations')
plt.xlabel('Time Lag (hours)')
plt.ylabel('Correlation')
plt.grid(True)
plt.legend()
plt.show()

# Additional analysis: Calculate hourly rate of change
print("\nAnalyzing rate of change patterns...")
for station in df['location_name'].unique():
    station_data = level_df[station]
    rate_of_change = station_data.diff()

    print(f"\n{station} Rate of Change Statistics:")
    print(f"Mean change: {rate_of_change.mean():.6f} m/hour")
    print(f"Max increase: {rate_of_change.max():.6f} m/hour")
    print(f"Max decrease: {rate_of_change.min():.6f} m/hour")

print("\nAnalysis complete!")
```

```
Analyzing inter-station relationships...

River Level Correlations between Stations:
                     Rochdale  Manchester Racecourse  Bury Ground
Rochdale                1.000                  0.356        0.280
Manchester Racecourse   0.356                  1.000        0.916
Bury Ground             0.280                  0.916        1.000

Rainfall Correlations between Stations:
                     Rochdale  Manchester Racecourse  Bury Ground
Rochdale                 1.00                   1.00        -0.05
Manchester Racecourse    1.00                   1.00        -0.05
Bury Ground             -0.05                  -0.05         1.00
```




```
Analyzing time lags between stations...

Rochdale -> Manchester Racecourse:
Maximum correlation: -0.643 at 10.0 hours lag

Rochdale -> Bury Ground:
Maximum correlation: -0.686 at 12.0 hours lag

Manchester Racecourse -> Bury Ground:
Maximum correlation: 0.916 at 0.0 hours lag
```



```
Analyzing rate of change patterns...

Rochdale Rate of Change Statistics:
Mean change: 0.000000 m/hour
Max increase: 0.002000 m/hour
Max decrease: -0.003000 m/hour

Manchester Racecourse Rate of Change Statistics:
Mean change: -0.000600 m/hour
Max increase: 0.004000 m/hour
Max decrease: -0.007000 m/hour

Bury Ground Rate of Change Statistics:
Mean change: -0.000300 m/hour
Max increase: 0.001000 m/hour
Max decrease: -0.005000 m/hour

Analysis complete!
```

Alert System Setup - Email Notifications

In [16]:
```
import sys
import subprocess

# Install packages
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'supabase', 'pand
```

Out[16]:
```
0
```

In [17]:
```
# Install required libraries
%pip install supabase pandas pyyaml

# Import necessary libraries
import logging
from supabase import create_client, Client
import yaml
import pandas as pd

# Logging Setup
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

def test_supabase_connection(config):
    """
    Comprehensive Supabase connection and data retrieval test
    """
    try:
        # Extract Supabase credentials
        supabase_url = config['supabase']['url']
        supabase_key = config['supabase']['key']

        # Create Supabase client
        supabase: Client = create_client(supabase_url, supabase_key)

        # Test connection by fetching river data
        print("Attempting to fetch river data...")

        # Fetch latest 10 records
        response = supabase.table('river_data').select('*').order('river_timesta
```

```
        # Convert to DataFrame
        df = pd.DataFrame(response.data)

        # Display connection and data retrieval results
        print("Supabase Connection Successful! ✓")
        print(f"Total records retrieved: {len(df)}")

        # Display column names
        print("\nColumns in river_data:")
        print(df.columns.tolist())

        # Display first few rows
        print("\nFirst few rows of data:")
        print(df.head())

        return df

    except Exception as e:
        print(f"Supabase Connection or Data Retrieval Failed: {e}")
        return None

def load_config(config_path='C:\\Users\\Administrator\\NEWPROJECT\\alert_config.
    """
    Load configuration from YAML file
    """
    try:
        with open(config_path, 'r') as file:
            config = yaml.safe_load(file)
        return config
    except Exception as e:
        print(f"Error loading configuration: {e}")
        return None

# Main execution
def main():
    # Load configuration
    config = load_config()

    if config:
        # Test Supabase connection and data retrieval
        river_data = test_supabase_connection(config)

        # Additional analysis if data is retrieved
        if river_data is not None:
            # Example: Basic statistical analysis
            print("\nBasic Statistical Analysis:")
            for column in ['river_level', 'rainfall']:
                if column in river_data.columns:
                    print(f"\n{column.capitalize()} Statistics:")
                    print(river_data[column].describe())

# Run the main function
main()
```

```
Requirement already satisfied: supabase in c:\users\administrator\anaconda3\lib\s
ite-packages (2.13.0)Note: you may need to restart the kernel to use updated pack
ages.

Requirement already satisfied: pandas in c:\users\administrator\appdata\roaming\p
ython\python312\site-packages (2.2.3)
Requirement already satisfied: pyyaml in c:\users\administrator\anaconda3\lib\sit
e-packages (6.0.1)
Requirement already satisfied: gotrue<3.0.0,>=2.11.0 in c:\users\administrator\an
aconda3\lib\site-packages (from supabase) (2.11.3)
Requirement already satisfied: httpx<0.29,>=0.26 in c:\users\administrator\anacon
da3\lib\site-packages (from supabase) (0.27.0)
Requirement already satisfied: postgrest<0.20,>=0.19 in c:\users\administrator\an
aconda3\lib\site-packages (from supabase) (0.19.3)
Requirement already satisfied: realtime<3.0.0,>=2.0.0 in c:\users\administrator\a
naconda3\lib\site-packages (from supabase) (2.3.0)
Requirement already satisfied: storage3<0.12,>=0.10 in c:\users\administrator\ana
conda3\lib\site-packages (from supabase) (0.11.3)
Requirement already satisfied: supafunc<0.10,>=0.9 in c:\users\administrator\anac
onda3\lib\site-packages (from supabase) (0.9.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\administrator\anaconda3
\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\administrator\a
naconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\administrator\anaconda3\l
ib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\administrator\appdata\r
oaming\python\python312\site-packages (from pandas) (2025.1)
Requirement already satisfied: pydantic<3,>=1.10 in c:\users\administrator\anacon
da3\lib\site-packages (from gotrue<3.0.0,>=2.11.0->supabase) (2.8.2)
Requirement already satisfied: anyio in c:\users\administrator\anaconda3\lib\site
-packages (from httpx<0.29,>=0.26->supabase) (4.6.2)
Requirement already satisfied: certifi in c:\users\administrator\anaconda3\lib\si
te-packages (from httpx<0.29,>=0.26->supabase) (2025.1.31)
Requirement already satisfied: httpcore==1.* in c:\users\administrator\anaconda3
\lib\site-packages (from httpx<0.29,>=0.26->supabase) (1.0.2)
Requirement already satisfied: idna in c:\users\administrator\anaconda3\lib\site-
packages (from httpx<0.29,>=0.26->supabase) (3.7)
Requirement already satisfied: sniffio in c:\users\administrator\anaconda3\lib\si
te-packages (from httpx<0.29,>=0.26->supabase) (1.3.0)
Requirement already satisfied: h11<0.15,>=0.13 in c:\users\administrator\anaconda
3\lib\site-packages (from httpcore==1.*->httpx<0.29,>=0.26->supabase) (0.14.0)
Requirement already satisfied: deprecation<3.0.0,>=2.1.0 in c:\users\administrato
r\anaconda3\lib\site-packages (from postgrest<0.20,>=0.19->supabase) (2.1.0)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3\lib\s
ite-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: aiohttp<4.0.0,>=3.11.11 in c:\users\administrator
\anaconda3\lib\site-packages (from realtime<3.0.0,>=2.0.0->supabase) (3.11.12)
Requirement already satisfied: typing-extensions<5.0.0,>=4.12.2 in c:\users\admin
istrator\anaconda3\lib\site-packages (from realtime<3.0.0,>=2.0.0->supabase) (4.1
2.2)
Requirement already satisfied: websockets<15,>=11 in c:\users\administrator\anaco
nda3\lib\site-packages (from realtime<3.0.0,>=2.0.0->supabase) (14.2)
Requirement already satisfied: strenum<0.5.0,>=0.4.15 in c:\users\administrator\a
naconda3\lib\site-packages (from supafunc<0.10,>=0.9->supabase) (0.4.15)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in c:\users\administrator
\anaconda3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.
0->supabase) (2.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in c:\users\administrator\anacond
a3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->supab
```

```
ase) (1.2.0)
Requirement already satisfied: attrs>=17.3.0 in c:\users\administrator\anaconda3
\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->supabas
e) (23.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in c:\users\administrator\anacon
da3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->supa
base) (1.4.0)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\administrator\anac
onda3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->su
pabase) (6.0.4)
Requirement already satisfied: propcache>=0.2.0 in c:\users\administrator\anacond
a3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->supab
ase) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in c:\users\administrator\anacon
da3\lib\site-packages (from aiohttp<4.0.0,>=3.11.11->realtime<3.0.0,>=2.0.0->supa
base) (1.18.3)
Requirement already satisfied: packaging in c:\users\administrator\anaconda3\lib
\site-packages (from deprecation<3.0.0,>=2.1.0->postgrest<0.20,>=0.19->supabase)
(24.1)
Requirement already satisfied: h2<5,>=3 in c:\users\administrator\anaconda3\lib\s
ite-packages (from httpx[http2]<0.29,>=0.26->gotrue<3.0.0,>=2.11.0->supabase) (4.
2.0)
Requirement already satisfied: annotated-types>=0.4.0 in c:\users\administrator\a
naconda3\lib\site-packages (from pydantic<3,>=1.10->gotrue<3.0.0,>=2.11.0->supaba
se) (0.6.0)
Requirement already satisfied: pydantic-core==2.20.1 in c:\users\administrator\an
aconda3\lib\site-packages (from pydantic<3,>=1.10->gotrue<3.0.0,>=2.11.0->supabas
e) (2.20.1)
Requirement already satisfied: hyperframe<7,>=6.1 in c:\users\administrator\anaco
nda3\lib\site-packages (from h2<5,>=3->httpx[http2]<0.29,>=0.26->gotrue<3.0.0,>=
2.11.0->supabase) (6.1.0)
Requirement already satisfied: hpack<5,>=4.1 in c:\users\administrator\anaconda3
\lib\site-packages (from h2<5,>=3->httpx[http2]<0.29,>=0.26->gotrue<3.0.0,>=2.11.
0->supabase) (4.1.0)
Attempting to fetch river data...
```

```
2025-02-13 18:23:24,148 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&order=river_timestamp.desc&limit=10 "HTT
P/2 200 OK"
```

```
Supabase Connection Successful! ✓
Total records retrieved: 10

Columns in river_data:
['id', 'river_level', 'river_timestamp', 'rainfall', 'rainfall_timestamp', 'locat
ion_name', 'river_station_id', 'rainfall_station_id', 'created_at']

First few rows of data:
     id  river_level            river_timestamp  rainfall  \
0  3801        0.309  2025-02-13T13:00:00+00:00         0
1  3804        0.309  2025-02-13T13:00:00+00:00         0
2  3798        0.309  2025-02-13T13:00:00+00:00         0
3  3800        0.915  2025-02-13T13:00:00+00:00         0
4  3802        0.166  2025-02-13T13:00:00+00:00         0

          rainfall_timestamp          location_name  river_station_id  \
0  2025-02-13T13:00:00+00:00            Bury Ground            690160
1  2025-02-13T13:00:00+00:00            Bury Ground            690160
2  2025-02-13T13:00:00+00:00            Bury Ground            690160
3  2025-02-13T13:00:00+00:00  Manchester Racecourse            690510
4  2025-02-13T13:00:00+00:00               Rochdale            690203

   rainfall_station_id                        created_at
0               562656  2025-02-13T13:34:41.482687+00:00
1               562656  2025-02-13T13:35:42.139416+00:00
2               562656  2025-02-13T13:33:41.039372+00:00
3               562992  2025-02-13T13:34:41.395317+00:00
4               561613  2025-02-13T13:35:41.911355+00:00

Basic Statistical Analysis:

River_level Statistics:
count    10.000000
mean      0.433600
std       0.338009
min       0.166000
25%       0.166000
50%       0.309000
75%       0.763500
max       0.915000
Name: river_level, dtype: float64

Rainfall Statistics:
count    10.0
mean      0.0
std       0.0
min       0.0
25%       0.0
50%       0.0
75%       0.0
max       0.0
Name: rainfall, dtype: float64
```

Alert System Development

In [18]:

```
# Alert Thresholds Configuration
alert_thresholds = {
    'Rochdale': {
        'levels': {
```

```
            'low_risk': 0.160,
            'warning': 0.168,
            'high_risk': 0.170,
            'critical': 0.175
        },
        'rate_of_change': {
            'warning_threshold': 0.005,  # meters per hour
            'critical_threshold': 0.010
        }
    },
    'Manchester Racecourse': {
        'levels': {
            'low_risk': 0.920,
            'warning': 0.938,
            'high_risk': 0.950,
            'critical': 0.960
        },
        'rate_of_change': {
            'warning_threshold': 0.010,
            'critical_threshold': 0.020
        }
    },
    'Bury Ground': {
        'levels': {
            'low_risk': 0.300,
            'warning': 0.314,
            'high_risk': 0.320,
            'critical': 0.330
        },
        'rate_of_change': {
            'warning_threshold': 0.003,
            'critical_threshold': 0.008
        }
    }
}

# Function to determine risk level
def assess_risk(station, current_level, previous_level=None):
    """
    Assess risk level based on current water level and optional rate of change

    Args:
    - station (str): Name of the monitoring station
    - current_level (float): Current water level
    - previous_level (float, optional): Previous water level for rate of change

    Returns:
    - dict: Risk assessment details
    """
    # Get station-specific thresholds
    thresholds = alert_thresholds.get(station, {}).get('levels', {})
    change_thresholds = alert_thresholds.get(station, {}).get('rate_of_change',

    # Risk level determination
    if current_level >= thresholds.get('critical', float('inf')):
        risk_level = 'CRITICAL'
        risk_color = 'red'
    elif current_level >= thresholds.get('high_risk', float('inf')):
        risk_level = 'HIGH'
        risk_color = 'orange'
```

```
    elif current_level >= thresholds.get('warning', float('inf')):
        risk_level = 'WARNING'
        risk_color = 'yellow'
    else:
        risk_level = 'LOW'
        risk_color = 'green'

    # Calculate rate of change if previous level is provided
    rate_of_change = None
    if previous_level is not None:
        rate_of_change = current_level - previous_level

        # Adjust risk based on rate of change
        if abs(rate_of_change) >= change_thresholds.get('critical_threshold', fl
            risk_level = 'CRITICAL'
            risk_color = 'red'
        elif abs(rate_of_change) >= change_thresholds.get('warning_threshold', f
            risk_level = 'HIGH'
            risk_color = 'orange'

    return {
        'station': station,
        'current_level': current_level,
        'risk_level': risk_level,
        'risk_color': risk_color,
        'rate_of_change': rate_of_change
    }

# Example usage
def test_risk_assessment():
    # Test risk assessment for each station
    stations_data = [
        ('Rochdale', 0.169),
        ('Manchester Racecourse', 0.940),
        ('Bury Ground', 0.315)
    ]

    for station, level in stations_data:
        risk_assessment = assess_risk(station, level)
        print(f"{station} Risk Assessment:")
        for key, value in risk_assessment.items():
            print(f"  {key.replace('_', ' ').title()}: {value}")
        print()

# Run the test
test_risk_assessment()
```

```
Rochdale Risk Assessment:
  Station: Rochdale
  Current Level: 0.169
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: None

Manchester Racecourse Risk Assessment:
  Station: Manchester Racecourse
  Current Level: 0.94
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: None

Bury Ground Risk Assessment:
  Station: Bury Ground
  Current Level: 0.315
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: None
```

In [19]:

```python
import pandas as pd
from supabase import create_client, Client

def calculate_rate_of_change(station, config):
    """
    Calculate rate of change by fetching recent historical data
    """
    try:
        # Supabase connection
        supabase: Client = create_client(
            config['supabase']['url'],
            config['supabase']['key']
        )

        # Fetch last 2 records for the specific station
        response = supabase.table('river_data').select('*').eq('location_name',

        # Convert to DataFrame
        df = pd.DataFrame(response.data)

        if len(df) < 2:
            print(f"Not enough data to calculate rate of change for {station}")
            return None

        # Calculate time difference and level change
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])
        time_diff = (df['river_timestamp'].max() - df['river_timestamp'].min()).
        level_change = df['river_level'].max() - df['river_level'].min()

        # Rate of change per hour
        rate_of_change = level_change / time_diff if time_diff > 0 else 0

        return rate_of_change

    except Exception as e:
        print(f"Error calculating rate of change for {station}: {e}")
        return None
```

```python
def enhanced_risk_assessment(station, current_level, config):
    """
    Enhanced risk assessment with rate of change
    """
    # Get station-specific thresholds
    thresholds = alert_thresholds.get(station, {}).get('levels', {})
    change_thresholds = alert_thresholds.get(station, {}).get('rate_of_change',

    # Calculate rate of change
    rate_of_change = calculate_rate_of_change(station, config)

    # Risk level determination
    if current_level >= thresholds.get('critical', float('inf')):
        risk_level = 'CRITICAL'
        risk_color = 'red'
    elif current_level >= thresholds.get('high_risk', float('inf')):
        risk_level = 'HIGH'
        risk_color = 'orange'
    elif current_level >= thresholds.get('warning', float('inf')):
        risk_level = 'WARNING'
        risk_color = 'yellow'
    else:
        risk_level = 'LOW'
        risk_color = 'green'

    # Adjust risk based on rate of change
    if rate_of_change is not None:
        if abs(rate_of_change) >= change_thresholds.get('critical_threshold', fl
            risk_level = 'CRITICAL'
            risk_color = 'red'
        elif abs(rate_of_change) >= change_thresholds.get('warning_threshold', f
            risk_level = max(risk_level, 'HIGH')
            risk_color = 'orange'

    return {
        'station': station,
        'current_level': current_level,
        'risk_level': risk_level,
        'risk_color': risk_color,
        'rate_of_change': rate_of_change
    }

# Load configuration
def load_config(config_path='C:\\Users\\Administrator\\NEWPROJECT\\alert_config.
    import yaml
    with open(config_path, 'r') as file:
        return yaml.safe_load(file)

# Test the enhanced risk assessment
def test_enhanced_risk_assessment():
    # Load configuration
    config = load_config()

    # Test stations with current levels
    stations_data = [
        ('Rochdale', 0.169),
        ('Manchester Racecourse', 0.940),
        ('Bury Ground', 0.315)
    ]
```

```
    for station, level in stations_data:
        risk_assessment = enhanced_risk_assessment(station, level, config)
        print(f"{station} Risk Assessment:")
        for key, value in risk_assessment.items():
            print(f"  {key.replace('_', ' ').title()}: {value}")
        print()

# Run the test
test_enhanced_risk_assessment()
```

```
2025-02-13 18:36:04,589 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.supabase.co/rest/v1/river_data?select=%2A&location_name=eq.Rochdale&order=river_timestamp.desc&limit=2 "HTTP/2 200 OK"
Rochdale Risk Assessment:
  Station: Rochdale
  Current Level: 0.169
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: 0

2025-02-13 18:36:05,307 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.supabase.co/rest/v1/river_data?select=%2A&location_name=eq.Manchester%20Racecourse&order=river_timestamp.desc&limit=2 "HTTP/2 200 OK"
Manchester Racecourse Risk Assessment:
  Station: Manchester Racecourse
  Current Level: 0.94
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: 0

2025-02-13 18:36:06,033 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.supabase.co/rest/v1/river_data?select=%2A&location_name=eq.Bury%20Ground&order=river_timestamp.desc&limit=2 "HTTP/2 200 OK"
Bury Ground Risk Assessment:
  Station: Bury Ground
  Current Level: 0.315
  Risk Level: WARNING
  Risk Color: yellow
  Rate Of Change: 0
```

In [20]:

```
import pandas as pd
from supabase import create_client, Client
from datetime import timedelta

def detailed_station_analysis(station, config):
    """
    Comprehensive station data analysis
    """
    try:
        # Supabase connection
        supabase: Client = create_client(
            config['supabase']['url'],
            config['supabase']['key']
        )

        # Fetch recent records (last 24 hours)
        response = supabase.table('river_data').select('*').eq('location_name',
```

```
        # Convert to DataFrame
        df = pd.DataFrame(response.data)

        # Convert timestamp
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

        # Detailed Analysis
        analysis = {
            'station': station,
            'total_records': len(df),
            'level_stats': {
                'min': df['river_level'].min(),
                'max': df['river_level'].max(),
                'mean': df['river_level'].mean(),
                'std': df['river_level'].std()
            },
            'timestamp_range': {
                'earliest': df['river_timestamp'].min(),
                'latest': df['river_timestamp'].max(),
                'duration': df['river_timestamp'].max() - df['river_timestamp'].
            },
            'level_changes': []
        }

        # Calculate level changes
        df_sorted = df.sort_values('river_timestamp')
        analysis['level_changes'] = [
            {
                'time_diff': (df_sorted['river_timestamp'].iloc[i+1] - df_sorted
                'level_change': df_sorted['river_level'].iloc[i+1] - df_sorted['
            }
            for i in range(len(df_sorted) - 1)
        ]

        return analysis

    except Exception as e:
        print(f"Error analyzing {station} data: {e}")
        return None

def print_station_analysis(stations):
    """
    Print detailed station analysis
    """
    config = load_config()

    for station in stations:
        print(f"\n{station.upper()} DETAILED ANALYSIS:")
        analysis = detailed_station_analysis(station, config)

        if analysis:
            print("Overall Statistics:")
            print(f"  Total Records: {analysis['total_records']}")

            print("\nLevel Statistics:")
            for key, value in analysis['level_stats'].items():
                print(f"  {key.capitalize()}: {value:.4f}m")

            print("\nTimestamp Analysis:")
            print(f"  Earliest Record: {analysis['timestamp_range']['earliest']}
```

```
            print(f"  Latest Record: {analysis['timestamp_range']['latest']}")
            print(f"  Data Duration: {analysis['timestamp_range']['duration']}")

            print("\nLevel Changes:")
            if analysis['level_changes']:
                changes = analysis['level_changes']
                avg_time_diff = sum(change['time_diff'] for change in changes) /
                avg_level_change = sum(change['level_change'] for change in chan

                print(f"  Average Time Between Measurements: {avg_time_diff:.2f}
                print(f"  Average Level Change: {avg_level_change:.4f}m")
                print(f"  Maximum Level Change: {max(abs(change['level_change'])
            else:
                print("  No level changes detected")

# Run analysis
stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']
print_station_analysis(stations)
```

```
ROCHDALE DETAILED ANALYSIS:
2025-02-13 18:38:05,295 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Rochdale&order=river_ti
mestamp.desc&limit=24 "HTTP/2 200 OK"
Overall Statistics:
  Total Records: 24

Level Statistics:
  Min: 0.1660m
  Max: 0.1670m
  Mean: 0.1666m
  Std: 0.0005m

Timestamp Analysis:
  Earliest Record: 2025-02-13 12:45:00+00:00
  Latest Record: 2025-02-13 13:00:00+00:00
  Data Duration: 0 days 00:15:00

Level Changes:
  Average Time Between Measurements: 0.01 hours
  Average Level Change: -0.0000m
  Maximum Level Change: 0.0010m

MANCHESTER RACECOURSE DETAILED ANALYSIS:
2025-02-13 18:38:06,048 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Manchester%20Racecourse
&order=river_timestamp.desc&limit=24 "HTTP/2 200 OK"
```

```
Overall Statistics:
  Total Records: 24

Level Statistics:
  Min: 0.9150m
  Max: 0.9160m
  Mean: 0.9156m
  Std: 0.0005m

Timestamp Analysis:
  Earliest Record: 2025-02-13 12:45:00+00:00
  Latest Record: 2025-02-13 13:00:00+00:00
  Data Duration: 0 days 00:15:00

Level Changes:
  Average Time Between Measurements: 0.01 hours
  Average Level Change: -0.0000m
  Maximum Level Change: 0.0010m

BURY GROUND DETAILED ANALYSIS:
2025-02-13 18:38:06,760 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.supabase.co/rest/v1/river_data?select=%2A&location_name=eq.Bury%20Ground&order=river_timestamp.desc&limit=24 "HTTP/2 200 OK"
Overall Statistics:
  Total Records: 24

Level Statistics:
  Min: 0.3090m
  Max: 0.3090m
  Mean: 0.3090m
  Std: 0.0000m

Timestamp Analysis:
  Earliest Record: 2025-02-13 12:45:00+00:00
  Latest Record: 2025-02-13 13:00:00+00:00
  Data Duration: 0 days 00:15:00

Level Changes:
  Average Time Between Measurements: 0.01 hours
  Average Level Change: 0.0000m
  Maximum Level Change: 0.0000m
```

Email Notification Configuration

In [23]:
```python
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

class FloodAlertNotifier:
    def __init__(self, config):
        """
        Initialize email notification configuration
        """
        self.smtp_server = config['email']['smtp_server']
        self.smtp_port = config['email']['smtp_port']
        self.sender_email = config['email']['sender_email']
        self.sender_password = config['email']['sender_password']
        self.recipients = config['email']['recipients']
```

```python
    def send_alert(self, station, risk_level, current_level):
        """
        Send email alert for flood risk
        """
        try:
            # Create message
            msg = MIMEMultipart()
            msg['From'] = self.sender_email
            msg['To'] = ', '.join(self.recipients)
            msg['Subject'] = f"Flood Alert: {station} - {risk_level} Risk"

            # Compose email body
            body = f"""
            FLOOD MONITORING ALERT

            Station: {station}
            Current Water Level: {current_level}m
            Risk Level: {risk_level}

            Immediate action may be required.
            """

            msg.attach(MIMEText(body, 'plain'))

            # Send email
            with smtplib.SMTP(self.smtp_server, self.smtp_port) as server:
                server.starttls()
                server.login(self.sender_email, self.sender_password)
                server.send_message(msg)

            print(f"Alert sent for {station}")
            return True
        except Exception as e:
            print(f"Failed to send alert for {station}: {e}")
            return False

# Test the alert system
def test_flood_alerts():
    # Load configuration
    import yaml
    with open('C:\\Users\\Administrator\\NEWPROJECT\\alert_config.yaml', 'r') as
        config = yaml.safe_load(file)

    # Initialize notifier
    notifier = FloodAlertNotifier(config)

    # Simulate alerts for different stations
    test_scenarios = [
        ('Rochdale', 'WARNING', 0.169),
        ('Manchester Racecourse', 'HIGH', 0.950),
        ('Bury Ground', 'CRITICAL', 0.330)
    ]

    for station, risk_level, current_level in test_scenarios:
        notifier.send_alert(station, risk_level, current_level)

# Uncomment to test
test_flood_alerts()
```

```
Alert sent for Rochdale
Alert sent for Manchester Racecourse
Alert sent for Bury Ground
```

In [22]:
```python
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import yaml
import logging

# Configure logging
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

class FloodAlertNotifier:
    def __init__(self, config):
        """
        Initialize email notification configuration with detailed logging
        """
        try:
            self.smtp_server = config['email']['smtp_server']
            self.smtp_port = config['email']['smtp_port']
            self.sender_email = config['email']['sender_email']
            self.sender_password = config['email']['sender_password']
            self.recipients = config['email']['recipients']

            # Log configuration details (be careful with sensitive info)
            logging.info("Email Configuration:")
            logging.info(f"SMTP Server: {self.smtp_server}")
            logging.info(f"SMTP Port: {self.smtp_port}")
            logging.info(f"Sender Email: {self.sender_email}")
            logging.info(f"Recipients: {self.recipients}")

        except KeyError as e:
            logging.error(f"Missing configuration key: {e}")
            raise

    def send_alert(self, station, risk_level, current_level):
        """
        Send email alert with comprehensive error handling
        """
        try:
            # Create message
            msg = MIMEMultipart()
            msg['From'] = self.sender_email
            msg['To'] = ', '.join(self.recipients)
            msg['Subject'] = f"Flood Alert: {station} - {risk_level} Risk"

            # Compose email body
            body = f"""
            FLOOD MONITORING ALERT

            Station: {station}
            Current Water Level: {current_level}m
            Risk Level: {risk_level}

            Immediate action may be required.
            """

            msg.attach(MIMEText(body, 'plain'))
```

```
            # Detailed logging before sending
            logging.info("Attempting to send email...")
            logging.info(f"Recipients: {self.recipients}")

            # Send email with more detailed logging
            with smtplib.SMTP(self.smtp_server, self.smtp_port) as server:
                # Enable logging for SMTP
                server.set_debuglevel(1)

                server.starttls()
                server.login(self.sender_email, self.sender_password)

                # Send to each recipient individually
                for recipient in self.recipients:
                    logging.info(f"Sending alert to: {recipient}")
                    server.sendmail(self.sender_email, recipient, msg.as_string(

            logging.info(f"Alert sent successfully for {station}")
            return True

        except smtplib.SMTPAuthenticationError:
            logging.error("SMTP Authentication Failed. Check email and password.
        except smtplib.SMTPException as smtp_error:
            logging.error(f"SMTP Error: {smtp_error}")
        except Exception as e:
            logging.error(f"Unexpected error sending alert: {e}")

        return False

# Comprehensive test function
def test_flood_alerts():
    """
    Test flood alert system with error handling
    """
    try:
        # Load configuration with error handling
        config_path = 'C:\\Users\\Administrator\\NEWPROJECT\\alert_config.yaml'
        logging.info(f"Loading configuration from {config_path}")

        with open(config_path, 'r') as file:
            config = yaml.safe_load(file)

        # Validate critical configuration keys
        required_keys = ['email', 'supabase']
        for key in required_keys:
            if key not in config:
                logging.error(f"Missing required configuration section: {key}")
                return

        # Initialize notifier
        notifier = FloodAlertNotifier(config)

        # Simulate alerts for different stations
        test_scenarios = [
            ('Rochdale', 'WARNING', 0.169),
            ('Manchester Racecourse', 'HIGH', 0.950),
            ('Bury Ground', 'CRITICAL', 0.330)
        ]
```

```
        # Run tests
        for station, risk_level, current_level in test_scenarios:
            logging.info(f"Testing alert for {station}")
            result = notifier.send_alert(station, risk_level, current_level)
            logging.info(f"Alert send result for {station}: {result}")

    except FileNotFoundError:
        logging.error(f"Configuration file not found at {config_path}")
    except Exception as e:
        logging.error(f"Unexpected error in test: {e}")

# Run the test
if __name__ == "__main__":
    test_flood_alerts()
```

```
2025-02-13 18:50:26,217 - INFO - Loading configuration from C:\Users\Administrato
r\NEWPROJECT\alert_config.yaml
2025-02-13 18:50:26,222 - INFO - Email Configuration:
2025-02-13 18:50:26,223 - INFO - SMTP Server: smtp.gmail.com
2025-02-13 18:50:26,224 - INFO - SMTP Port: 587
2025-02-13 18:50:26,225 - INFO - Sender Email: emi.igein@gmail.com
2025-02-13 18:50:26,225 - INFO - Recipients: ['emi.igein@gmail.com', 'kigein@gmai
l.com']
2025-02-13 18:50:26,226 - INFO - Testing alert for Rochdale
2025-02-13 18:50:26,230 - INFO - Attempting to send email...
2025-02-13 18:50:26,231 - INFO - Recipients: ['emi.igein@gmail.com', 'kigein@gmai
l.com']
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-STARTTLS\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nPIP
ELINING\nCHUNKING\nSMTPUTF8'
send: 'STARTTLS\r\n'
reply: b'220 2.0.0 Ready to start TLS\r\n'
reply: retcode (220); Msg: b'2.0.0 Ready to start TLS'
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nAUTH LOGIN PLAIN XOAUTH2 PLAIN-CLI
ENTTOKEN OAUTHBEARER XOAUTH\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'AUTH PLAIN AGVtaS5pZ2VpbkBnbWFpbC5jb20AendvdiBpZW1yIHNod2wgaWZmcw==\r\n'
reply: b'235 2.7.0 Accepted\r\n'
reply: retcode (235); Msg: b'2.7.0 Accepted'
2025-02-13 18:50:26,744 - INFO - Sending alert to: emi.igein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=608\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
send: 'rcpt TO:<emi.igein@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============61784363320878034
21=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
```

```
m, kigein@gmail.com\r\nSubject: Flood Alert: Rochdale - WARNING Risk\r\n\r\n--===
===========6178436332087803421==\r\nContent-Type: text/plain; charset="us-asci
i"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Rochdale\r\n            Curren
t Water Level: 0.169m\r\n            Risk Level: WARNING\r\n\r\n            Immed
iate action may be required.\r\n            \r\n--===============6178436332087803
421==--\r\n.\r\n'
reply: b'250 2.0.0 OK  1739472626 ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472626 ffacd0b85a97d-38f259d5e92sm2613
851f8f.66 - gsmtp'
data: (250, b'2.0.0 OK  1739472626 ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsm
tp')
2025-02-13 18:50:27,574 - INFO - Sending alert to: kigein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=608\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
send: 'rcpt TO:<kigein@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============61784363320878034
21=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
m, kigein@gmail.com\r\nSubject: Flood Alert: Rochdale - WARNING Risk\r\n\r\n--===
===========6178436332087803421==\r\nContent-Type: text/plain; charset="us-asci
i"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Rochdale\r\n            Curren
t Water Level: 0.169m\r\n            Risk Level: WARNING\r\n\r\n            Immed
iate action may be required.\r\n            \r\n--===============6178436332087803
421==--\r\n.\r\n'
reply: b'250 2.0.0 OK  1739472627 ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472627 ffacd0b85a97d-38f259d5e92sm2613
851f8f.66 - gsmtp'
data: (250, b'2.0.0 OK  1739472627 ffacd0b85a97d-38f259d5e92sm2613851f8f.66 - gsm
tp')
send: 'QUIT\r\n'
reply: b'221 2.0.0 closing connection ffacd0b85a97d-38f259d5e92sm2613851f8f.66 -
gsmtp\r\n'
reply: retcode (221); Msg: b'2.0.0 closing connection ffacd0b85a97d-38f259d5e92sm
2613851f8f.66 - gsmtp'
2025-02-13 18:50:28,567 - INFO - Alert sent successfully for Rochdale
2025-02-13 18:50:28,569 - INFO - Alert send result for Rochdale: True
2025-02-13 18:50:28,570 - INFO - Testing alert for Manchester Racecourse
2025-02-13 18:50:28,571 - INFO - Attempting to send email...
2025-02-13 18:50:28,573 - INFO - Recipients: ['emi.igein@gmail.com', 'kigein@gmai
l.com']
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-STARTTLS\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
```

```
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nPIP
ELINING\nCHUNKING\nSMTPUTF8'
send: 'STARTTLS\r\n'
reply: b'220 2.0.0 Ready to start TLS\r\n'
reply: retcode (220); Msg: b'2.0.0 Ready to start TLS'
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nAUTH LOGIN PLAIN XOAUTH2 PLAIN-CLI
ENTTOKEN OAUTHBEARER XOAUTH\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'AUTH PLAIN AGVtaS5pZ2VpbkBnbWFpbC5jb20AendvdiBpZW1yIHNod2wgaWZmcw==\r\n'
reply: b'235 2.7.0 Accepted\r\n'
reply: retcode (235); Msg: b'2.7.0 Accepted'
2025-02-13 18:50:29,145 - INFO - Sending alert to: emi.igein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=627\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
send: 'rcpt TO:<emi.igein@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============22056065969224524
12=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
m, kigein@gmail.com\r\nSubject: Flood Alert: Manchester Racecourse - HIGH Risk\r
\n\r\n--===============2205606596922452412==\r\nContent-Type: text/plain; charset
="us-ascii"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Manchester Racecourse\r\n
Current Water Level: 0.95m\r\n            Risk Level: HIGH\r\n\r\n            Imm
ediate action may be required.\r\n            \r\n--===============22056065969224
52412==--\r\n.\r\n'
reply: b'250 2.0.0 OK  1739472629 ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472629 ffacd0b85a97d-38f258ccd3bsm2594
913f8f.22 - gsmtp'
data: (250, b'2.0.0 OK  1739472629 ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsm
tp')
2025-02-13 18:50:30,032 - INFO - Sending alert to: kigein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=627\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
send: 'rcpt TO:<kigein@gmail.com>\r\n'
```

```
reply: b'250 2.1.5 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============22056065969224524
12=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
m, kigein@gmail.com\r\nSubject: Flood Alert: Manchester Racecourse - HIGH Risk\r
\n\r\n--===============2205606596922452412==\r\nContent-Type: text/plain; charset
="us-ascii"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Manchester Racecourse\r\n
Current Water Level: 0.95m\r\n            Risk Level: HIGH\r\n\r\n            Imm
ediate action may be required.\r\n            \r\n--===============22056065969224
52412==--\r\n.\r\n'
reply: b'250 2.0.0 OK  1739472630 ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472630 ffacd0b85a97d-38f258ccd3bsm2594
913f8f.22 - gsmtp'
data: (250, b'2.0.0 OK  1739472630 ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 - gsm
tp')
send: 'QUIT\r\n'
reply: b'221 2.0.0 closing connection ffacd0b85a97d-38f258ccd3bsm2594913f8f.22 -
gsmtp\r\n'
reply: retcode (221); Msg: b'2.0.0 closing connection ffacd0b85a97d-38f258ccd3bsm
2594913f8f.22 - gsmtp'
2025-02-13 18:50:31,067 - INFO - Alert sent successfully for Manchester Racecours
e
2025-02-13 18:50:31,069 - INFO - Alert send result for Manchester Racecourse: Tru
e
2025-02-13 18:50:31,069 - INFO - Testing alert for Bury Ground
2025-02-13 18:50:31,070 - INFO - Attempting to send email...
2025-02-13 18:50:31,071 - INFO - Recipients: ['emi.igein@gmail.com', 'kigein@gmai
l.com']
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-STARTTLS\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nPIP
ELINING\nCHUNKING\nSMTPUTF8'
send: 'STARTTLS\r\n'
reply: b'220 2.0.0 Ready to start TLS\r\n'
reply: retcode (220); Msg: b'2.0.0 Ready to start TLS'
send: 'ehlo Laptop.Home\r\n'
reply: b'250-smtp.gmail.com at your service, [2a02:c7c:3203:d300:7495:a548:6003:b
49a]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
```

```
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [2a02:c7c:3203:d300:
7495:a548:6003:b49a]\nSIZE 35882577\n8BITMIME\nAUTH LOGIN PLAIN XOAUTH2 PLAIN-CLI
ENTTOKEN OAUTHBEARER XOAUTH\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'AUTH PLAIN AGVtaS5pZ2VpbkBnbWFpbC5jb20AendvdiBpZW1yIHNod2wgaWZmcw==\r\n'
reply: b'235 2.7.0 Accepted\r\n'
reply: retcode (235); Msg: b'2.7.0 Accepted'
2025-02-13 18:50:31,519 - INFO - Sending alert to: emi.igein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=615\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
send: 'rcpt TO:<emi.igein@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============64187665169850732
99=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
m, kigein@gmail.com\r\nSubject: Flood Alert: Bury Ground - CRITICAL Risk\r\n\r\n-
-===============6418766516985073299==\r\nContent-Type: text/plain; charset="us-as
cii"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Bury Ground\r\n            Cur
rent Water Level: 0.33m\r\n            Risk Level: CRITICAL\r\n\r\n            Im
mediate action may be required.\r\n            \r\n--===============6418766516985
073299==--\r\n.\r\n'
reply: b'250 2.0.0 OK  1739472631 ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472631 ffacd0b85a97d-38f258f5fabsm2565
075f8f.45 - gsmtp'
data: (250, b'2.0.0 OK  1739472631 ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsm
tp')
2025-02-13 18:50:32,346 - INFO - Sending alert to: kigein@gmail.com
send: 'mail FROM:<emi.igein@gmail.com> size=615\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
send: 'rcpt TO:<kigein@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmtp')
send: b'Content-Type: multipart/mixed; boundary="===============64187665169850732
99=="\r\nMIME-Version: 1.0\r\nFrom: emi.igein@gmail.com\r\nTo: emi.igein@gmail.co
m, kigein@gmail.com\r\nSubject: Flood Alert: Bury Ground - CRITICAL Risk\r\n\r\n-
-===============6418766516985073299==\r\nContent-Type: text/plain; charset="us-as
cii"\r\nMIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\n\r\n\r\n
FLOOD MONITORING ALERT\r\n\r\n            Station: Bury Ground\r\n            Cur
rent Water Level: 0.33m\r\n            Risk Level: CRITICAL\r\n\r\n            Im
mediate action may be required.\r\n            \r\n--===============6418766516985
073299==--\r\n.\r\n'
```

```
reply: b'250 2.0.0 OK  1739472632 ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsmt
p\r\n'
reply: retcode (250); Msg: b'2.0.0 OK  1739472632 ffacd0b85a97d-38f258f5fabsm2565
075f8f.45 - gsmtp'
data: (250, b'2.0.0 OK  1739472632 ffacd0b85a97d-38f258f5fabsm2565075f8f.45 - gsm
tp')
send: 'QUIT\r\n'
reply: b'221 2.0.0 closing connection ffacd0b85a97d-38f258f5fabsm2565075f8f.45 -
gsmtp\r\n'
reply: retcode (221); Msg: b'2.0.0 closing connection ffacd0b85a97d-38f258f5fabsm
2565075f8f.45 - gsmtp'
2025-02-13 18:50:33,095 - INFO - Alert sent successfully for Bury Ground
2025-02-13 18:50:33,095 - INFO - Alert send result for Bury Ground: True
```

Real-Time Alert Integration

In [25]:

```python
import pandas as pd
from supabase import create_client, Client
import logging
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import yaml
import time

# Configure logging
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

class FloodAlertNotifier:
    def __init__(self, config):
        """
        Initialize email notification configuration
        """
        try:
            self.smtp_server = config['email']['smtp_server']
            self.smtp_port = config['email']['smtp_port']
            self.sender_email = config['email']['sender_email']
            self.sender_password = config['email']['sender_password']
            self.recipients = config['email']['recipients']
        except KeyError as e:
            logging.error(f"Missing email configuration: {e}")
            raise

    def send_alert(self, station, risk_level, current_level):
        """
        Send email alert for a specific station
        """
        try:
            # Create message
            msg = MIMEMultipart()
            msg['From'] = self.sender_email
            msg['To'] = ', '.join(self.recipients)
            msg['Subject'] = f"Flood Alert: {station} - {risk_level} Risk"

            # Compose email body
            body = f"""
            FLOOD MONITORING ALERT
```

```
            Station: {station}
            Current Water Level: {current_level:.3f}m
            Risk Level: {risk_level}

            Immediate action may be required.
            """

            msg.attach(MIMEText(body, 'plain'))

            # Send email
            with smtplib.SMTP(self.smtp_server, self.smtp_port) as server:
                server.starttls()
                server.login(self.sender_email, self.sender_password)

                for recipient in self.recipients:
                    server.sendmail(self.sender_email, recipient, msg.as_string(

            logging.info(f"Alert sent successfully for {station}")
            return True
        except Exception as e:
            logging.error(f"Failed to send alert for {station}: {e}")
            return False

class RealTimeFloodAlertSystem:
    def __init__(self, config):
        """
        Initialize flood alert system with configuration
        """
        # Supabase connection
        self.supabase = create_client(
            config['supabase']['url'],
            config['supabase']['key']
        )

        # Alert notification system
        self.notifier = FloodAlertNotifier(config)

        # Alert thresholds
        self.alert_thresholds = {
            'Rochdale': {
                'warning': 0.168,
                'critical': 0.170
            },
            'Manchester Racecourse': {
                'warning': 0.938,
                'critical': 0.950
            },
            'Bury Ground': {
                'warning': 0.314,
                'critical': 0.320
            }
        }

    def fetch_latest_station_data(self, station):
        """
        Fetch latest data for a specific station
        """
        try:
            # Fetch last 2 records to calculate rate of change
            response = self.supabase.table('river_data').select('*').eq('locatic
```

```
            df = pd.DataFrame(response.data)

            if len(df) < 2:
                logging.warning(f"Insufficient data for {station}")
                return None

            # Calculate key metrics
            current_level = df['river_level'].iloc[0]
            previous_level = df['river_level'].iloc[1]
            time_diff = (pd.to_datetime(df['river_timestamp'].iloc[0]) -
                         pd.to_datetime(df['river_timestamp'].iloc[1])).total_se

            rate_of_change = (current_level - previous_level) / time_diff if tim

            return {
                'station': station,
                'current_level': current_level,
                'rate_of_change': rate_of_change
            }

        except Exception as e:
            logging.error(f"Error fetching data for {station}: {e}")
            return None

    def assess_flood_risk(self, station_data):
        """
        Assess flood risk for a station
        """
        if not station_data:
            return None

        station = station_data['station']
        current_level = station_data['current_level']
        rate_of_change = station_data['rate_of_change']

        # Get station-specific thresholds
        thresholds = self.alert_thresholds.get(station, {})

        # Risk assessment logic
        if current_level >= thresholds.get('critical', float('inf')):
            return 'CRITICAL'
        elif current_level >= thresholds.get('warning', float('inf')):
            return 'WARNING'

        # Optional: Add rate of change consideration
        if abs(rate_of_change) > 0.005:  # Adjust threshold as needed
            return 'WARNING'

        return 'LOW'

    def process_station_alerts(self):
        """
        Process alerts for all stations
        """
        stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

        for station in stations:
            # Fetch latest station data
            station_data = self.fetch_latest_station_data(station)
```

```
            if not station_data:
                continue

            # Assess risk
            risk_level = self.assess_flood_risk(station_data)

            # Send alert if risk is not LOW
            if risk_level != 'LOW':
                logging.info(f"Alert triggered for {station}")
                self.notifier.send_alert(
                    station,
                    risk_level,
                    station_data['current_level']
                )

    def run_continuous_monitoring(self, interval_minutes=15):
        """
        Run continuous monitoring
        """
        logging.info("Starting continuous flood monitoring...")

        while True:
            try:
                # Process station alerts
                self.process_station_alerts()

                # Wait for next interval
                logging.info(f"Waiting {interval_minutes} minutes for next check
                time.sleep(interval_minutes * 60)

            except Exception as e:
                logging.error(f"Error in continuous monitoring: {e}")

# Main execution
def main():
    # Load configuration
    config_path = 'C:\\Users\\Administrator\\NEWPROJECT\\alert_config.yaml'

    try:
        with open(config_path, 'r') as file:
            config = yaml.safe_load(file)

        # Initialize and run alert system
        alert_system = RealTimeFloodAlertSystem(config)
        alert_system.process_station_alerts()

    except Exception as e:
        logging.error(f"Error in main execution: {e}")

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

```
2025-02-13 19:01:47,008 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Rochdale&order=river_ti
mestamp.desc&limit=2 "HTTP/2 200 OK"
2025-02-13 19:01:47,099 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Manchester%20Racecourse
&order=river_timestamp.desc&limit=2 "HTTP/2 200 OK"
2025-02-13 19:01:47,151 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Bury%20Ground&order=riv
er_timestamp.desc&limit=2 "HTTP/2 200 OK"
```

In [26]:
```python
def process_station_alerts(self):
    """
    Process alerts for all stations with detailed logging
    """
    stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

    for station in stations:
        # Fetch latest station data
        station_data = self.fetch_latest_station_data(station)

        if not station_data:
            logging.warning(f"No data available for {station}")
            continue

        # Detailed logging of station data
        logging.info(f"{station} Station Analysis:")
        logging.info(f"  Current Water Level: {station_data['current_level']:.3f
        logging.info(f"  Rate of Change: {station_data['rate_of_change']:.6f}m/h

        # Assess risk
        risk_level = self.assess_flood_risk(station_data)
        logging.info(f"  Risk Level: {risk_level}")

        # Send alert if risk is not LOW
        if risk_level != 'LOW':
            logging.warning(f"ALERT TRIGGERED for {station}")
            self.notifier.send_alert(
                station,
                risk_level,
                station_data['current_level']
            )

        logging.info("-" * 40)
```

In [27]:
```python
import pandas as pd
from supabase import create_client, Client
import logging
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import yaml
import time

# Configure logging with more detailed output
logging.basicConfig(
    level=logging.DEBUG,  # Changed to DEBUG for more detailed logs
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),  # Output to console
```

```
        logging.FileHandler('flood_alert_system.log')  # Output to file
    ]
)

class RealTimeFloodAlertSystem:
    def __init__(self, config):
        """
        Initialize flood alert system with comprehensive logging
        """
        try:
            # Supabase connection
            logging.info("Initializing Supabase connection...")
            self.supabase = create_client(
                config['supabase']['url'],
                config['supabase']['key']
            )
            logging.info("Supabase connection established successfully")

            # Validate configuration
            self._validate_config(config)

            # Alert thresholds
            self.alert_thresholds = {
                'Rochdale': {
                    'warning': 0.168,
                    'critical': 0.170
                },
                'Manchester Racecourse': {
                    'warning': 0.938,
                    'critical': 0.950
                },
                'Bury Ground': {
                    'warning': 0.314,
                    'critical': 0.320
            }}

        except Exception as e:
            logging.error(f"Initialization error: {e}")
            raise

    def _validate_config(self, config):
        """
        Validate configuration parameters
        """
        required_keys = ['supabase', 'email']
        for key in required_keys:
            if key not in config:
                raise ValueError(f"Missing required configuration section: {key}

        supabase_keys = ['url', 'key']
        email_keys = ['smtp_server', 'smtp_port', 'sender_email', 'sender_passwo

        for key in supabase_keys:
            if key not in config['supabase']:
                raise ValueError(f"Missing Supabase configuration: {key}")

        for key in email_keys:
            if key not in config['email']:
                raise ValueError(f"Missing email configuration: {key}")
```

```
    def fetch_latest_station_data(self, station):
        """
        Fetch latest data for a specific station with detailed error handling
        """
        try:
            logging.info(f"Fetching data for {station}")

            # Fetch last 2 records to calculate rate of change
            response = self.supabase.table('river_data').select('*').eq('locatic

            # Convert response to DataFrame
            df = pd.DataFrame(response.data)

            logging.debug(f"{station} data retrieved: {len(df)} records")

            if len(df) < 2:
                logging.warning(f"Insufficient data for {station}")
                return None

            # Calculate key metrics
            current_level = df['river_level'].iloc[0]
            previous_level = df['river_level'].iloc[1]

            # Convert timestamps and calculate time difference
            current_time = pd.to_datetime(df['river_timestamp'].iloc[0])
            previous_time = pd.to_datetime(df['river_timestamp'].iloc[1])
            time_diff = (current_time - previous_time).total_seconds() / 3600  #

            # Calculate rate of change
            rate_of_change = (current_level - previous_level) / time_diff if tim

            logging.info(f"{station} Analysis:")
            logging.info(f"  Current Level: {current_level:.3f}m")
            logging.info(f"  Previous Level: {previous_level:.3f}m")
            logging.info(f"  Time Difference: {time_diff:.2f} hours")
            logging.info(f"  Rate of Change: {rate_of_change:.6f}m/hour")

            return {
                'station': station,
                'current_level': current_level,
                'rate_of_change': rate_of_change
            }

        except Exception as e:
            logging.error(f"Error fetching data for {station}: {e}")
            return None

    def assess_flood_risk(self, station_data):
        """
        Comprehensive risk assessment with detailed logging
        """
        if not station_data:
            logging.warning("No station data provided for risk assessment")
            return None

        station = station_data['station']
        current_level = station_data['current_level']
        rate_of_change = station_data['rate_of_change']

        # Get station-specific thresholds
```

```
        thresholds = self.alert_thresholds.get(station, {})

        # Detailed risk assessment logging
        logging.info(f"Risk Assessment for {station}:")
        logging.info(f"  Current Level: {current_level:.3f}m")
        logging.info(f"  Warning Threshold: {thresholds.get('warning', 'Not Set'
        logging.info(f"  Critical Threshold: {thresholds.get('critical', 'Not Se
        logging.info(f"  Rate of Change: {rate_of_change:.6f}m/hour")

        # Risk assessment logic
        if current_level >= thresholds.get('critical', float('inf')):
            logging.warning(f"CRITICAL RISK detected for {station}")
            return 'CRITICAL'
        elif current_level >= thresholds.get('warning', float('inf')):
            logging.warning(f"WARNING RISK detected for {station}")
            return 'WARNING'

        # Optional: Add rate of change consideration
        if abs(rate_of_change) > 0.005:  # Adjust threshold as needed
            logging.info(f"Elevated risk due to rate of change for {station}")
            return 'WARNING'

        logging.info(f"Low risk for {station}")
        return 'LOW'

    def process_station_alerts(self):
        """
        Process alerts for all stations
        """
        stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

        for station in stations:
            try:
                # Fetch latest station data
                station_data = self.fetch_latest_station_data(station)

                if not station_data:
                    logging.warning(f"Skipping alert processing for {station} du
                    continue

                # Assess risk
                risk_level = self.assess_flood_risk(station_data)

                # Optional: Uncomment to send actual alerts
                # if risk_level != 'LOW':
                #     logging.warning(f"ALERT TRIGGERED for {station}")
                #     self.notifier.send_alert(
                #         station,
                #         risk_level,
                #         station_data['current_level']
                #     )

            except Exception as e:
                logging.error(f"Error processing alerts for {station}: {e}")

# Main execution
def main():
    # Load configuration
    config_path = 'C:\\Users\\Administrator\\NEWPROJECT\\alert_config.yaml'
```

```
    try:
        with open(config_path, 'r') as file:
            config = yaml.safe_load(file)

        # Initialize and run alert system
        alert_system = RealTimeFloodAlertSystem(config)
        alert_system.process_station_alerts()

    except Exception as e:
        logging.error(f"Error in main execution: {e}")

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

```
2025-02-13 19:07:41,675 - INFO - Initializing Supabase connection...
2025-02-13 19:07:42,033 - INFO - Supabase connection established successfully
2025-02-13 19:07:42,033 - INFO - Fetching data for Rochdale
2025-02-13 19:07:42,751 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Rochdale&order=river_ti
mestamp.desc&limit=2 "HTTP/2 200 OK"
2025-02-13 19:07:42,771 - INFO - Rochdale Analysis:
2025-02-13 19:07:42,773 - INFO -   Current Level: 0.166m
2025-02-13 19:07:42,774 - INFO -   Previous Level: 0.166m
2025-02-13 19:07:42,776 - INFO -   Time Difference: 0.00 hours
2025-02-13 19:07:42,777 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,779 - INFO - Risk Assessment for Rochdale:
2025-02-13 19:07:42,781 - INFO -   Current Level: 0.166m
2025-02-13 19:07:42,783 - INFO -   Warning Threshold: 0.168
2025-02-13 19:07:42,786 - INFO -   Critical Threshold: 0.17
2025-02-13 19:07:42,787 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,788 - INFO - Low risk for Rochdale
2025-02-13 19:07:42,789 - INFO - Fetching data for Manchester Racecourse
2025-02-13 19:07:42,867 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Manchester%20Racecourse
&order=river_timestamp.desc&limit=2 "HTTP/2 200 OK"
2025-02-13 19:07:42,872 - INFO - Manchester Racecourse Analysis:
2025-02-13 19:07:42,873 - INFO -   Current Level: 0.915m
2025-02-13 19:07:42,873 - INFO -   Previous Level: 0.915m
2025-02-13 19:07:42,874 - INFO -   Time Difference: 0.00 hours
2025-02-13 19:07:42,874 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,875 - INFO - Risk Assessment for Manchester Racecourse:
2025-02-13 19:07:42,876 - INFO -   Current Level: 0.915m
2025-02-13 19:07:42,876 - INFO -   Warning Threshold: 0.938
2025-02-13 19:07:42,877 - INFO -   Critical Threshold: 0.95
2025-02-13 19:07:42,877 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,880 - INFO - Low risk for Manchester Racecourse
2025-02-13 19:07:42,881 - INFO - Fetching data for Bury Ground
2025-02-13 19:07:42,917 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&location_name=eq.Bury%20Ground&order=riv
er_timestamp.desc&limit=2 "HTTP/2 200 OK"
2025-02-13 19:07:42,921 - INFO - Bury Ground Analysis:
2025-02-13 19:07:42,923 - INFO -   Current Level: 0.309m
2025-02-13 19:07:42,923 - INFO -   Previous Level: 0.309m
2025-02-13 19:07:42,923 - INFO -   Time Difference: 0.00 hours
2025-02-13 19:07:42,924 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,924 - INFO - Risk Assessment for Bury Ground:
2025-02-13 19:07:42,925 - INFO -   Current Level: 0.309m
2025-02-13 19:07:42,926 - INFO -   Warning Threshold: 0.314
2025-02-13 19:07:42,928 - INFO -   Critical Threshold: 0.32
2025-02-13 19:07:42,928 - INFO -   Rate of Change: 0.000000m/hour
2025-02-13 19:07:42,928 - INFO - Low risk for Bury Ground
```

Watershed Analysis

In [28]:

```
import numpy as np
import pandas as pd
from shapely.geometry import Point, Polygon
import geopandas as gpd

class WatershedAnalysis:
    def __init__(self):
        # Station coordinates
        self.stations = {
```

```
            'Rochdale': {'lat': 53.6174, 'lon': -2.1555},
            'Manchester Racecourse': {'lat': 53.4809, 'lon': -2.2374},
            'Bury Ground': {'lat': 53.5933, 'lon': -2.2973}
        }

        # Define watershed boundaries (simplified for demo)
        self.watersheds = {
            'Rochdale': {
                'boundary': [
                    (53.6274, -2.1655),
                    (53.6274, -2.1455),
                    (53.6074, -2.1455),
                    (53.6074, -2.1655)
                ],
                'area_km2': 12.5,
                'elevation_m': 150
            },
            'Manchester Racecourse': {
                'boundary': [
                    (53.4909, -2.2474),
                    (53.4909, -2.2274),
                    (53.4709, -2.2274),
                    (53.4709, -2.2474)
                ],
                'area_km2': 15.3,
                'elevation_m': 25
            },
            'Bury Ground': {
                'boundary': [
                    (53.6033, -2.3073),
                    (53.6033, -2.2873),
                    (53.5833, -2.2873),
                    (53.5833, -2.3073)
                ],
                'area_km2': 18.7,
                'elevation_m': 75
            }
        }

    def calculate_flow_paths(self):
        """Calculate water flow paths between stations based on elevation"""
        flow_paths = []
        stations_sorted = sorted(
            self.watersheds.items(),
            key=lambda x: x[1]['elevation_m'],
            reverse=True
        )

        for i in range(len(stations_sorted)-1):
            higher = stations_sorted[i][0]
            lower = stations_sorted[i+1][0]
            flow_paths.append({
                'from': higher,
                'to': lower,
                'elevation_diff': (
                    self.watersheds[higher]['elevation_m'] -
                    self.watersheds[lower]['elevation_m']
                )
            })
        return flow_paths
```

```
    def get_watershed_stats(self, station):
        """Get watershed statistics for a station"""
        watershed = self.watersheds[station]
        return {
            'area_km2': watershed['area_km2'],
            'elevation_m': watershed['elevation_m'],
            'boundary': watershed['boundary']
        }

    def calculate_flood_risk(self, station, current_level, rainfall):
        """Calculate flood risk based on watershed characteristics"""
        watershed = self.watersheds[station]
        base_risk = 0

        # Factor 1: Area size (larger catchment = higher risk)
        area_factor = watershed['area_km2'] / 10  # Normalize to 0-1 scale

        # Factor 2: Elevation (lower elevation = higher risk)
        elevation_factor = 1 - (watershed['elevation_m'] / 200)  # Normalize

        # Factor 3: Current water level vs typical
        level_factor = current_level * 2  # Simple scaling

        # Factor 4: Recent rainfall
        rain_factor = rainfall * 0.5  # Simple scaling

        # Combined risk score (0-100)
        risk_score = (
            area_factor * 25 +
            elevation_factor * 25 +
            level_factor * 25 +
            rain_factor * 25
        )

        return min(100, max(0, risk_score))  # Ensure 0-100 range
```

In [31]:

```
import pandas as pd
import numpy as np
from supabase import create_client
import os

# Set up Supabase connection
supabase_url = "https://thoqlquxaemyyhmpiwzt.supabase.co"
supabase_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ
supabase = create_client(supabase_url, supabase_key)

# Fetch latest data
print("Fetching data...")
response = supabase.table('river_data').select('*').execute()
df = pd.DataFrame(response.data)

# Basic station information
station_info = {
    'Rochdale': {
        'elevation': 150,  # meters above sea level
        'catchment_area': 12.5,  # km²
        'flow_to': 'Manchester Racecourse'
    },
    'Manchester Racecourse': {
```

```
        'elevation': 25,
        'catchment_area': 15.3,
        'flow_to': 'Bury Ground'
    },
    'Bury Ground': {
        'elevation': 75,
        'catchment_area': 18.7,
        'flow_to': None
    }
}

# Print basic station information
print("\nStation Information:")
for station, info in station_info.items():
    print(f"\n{station}:")
    for key, value in info.items():
        print(f"  {key}: {value}")

# Get current levels for each station
current_levels = df.groupby('location_name')['river_level'].last()
print("\nCurrent River Levels:")
print(current_levels)

# Test basic flow path
print("\nWater Flow Paths:")
for station, info in station_info.items():
    if info['flow_to']:
        print(f"{station} → {info['flow_to']}")
```

```
Fetching data...
2025-02-13 19:27:50,279 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A "HTTP/2 200 OK"
Station Information:

Rochdale:
  elevation: 150
  catchment_area: 12.5
  flow_to: Manchester Racecourse

Manchester Racecourse:
  elevation: 25
  catchment_area: 15.3
  flow_to: Bury Ground

Bury Ground:
  elevation: 75
  catchment_area: 18.7
  flow_to: None

Current River Levels:
location_name
Bury Ground              0.310
Manchester Racecourse    0.928
Rochdale                 0.168
Name: river_level, dtype: float64

Water Flow Paths:
Rochdale → Manchester Racecourse
Manchester Racecourse → Bury Ground
```

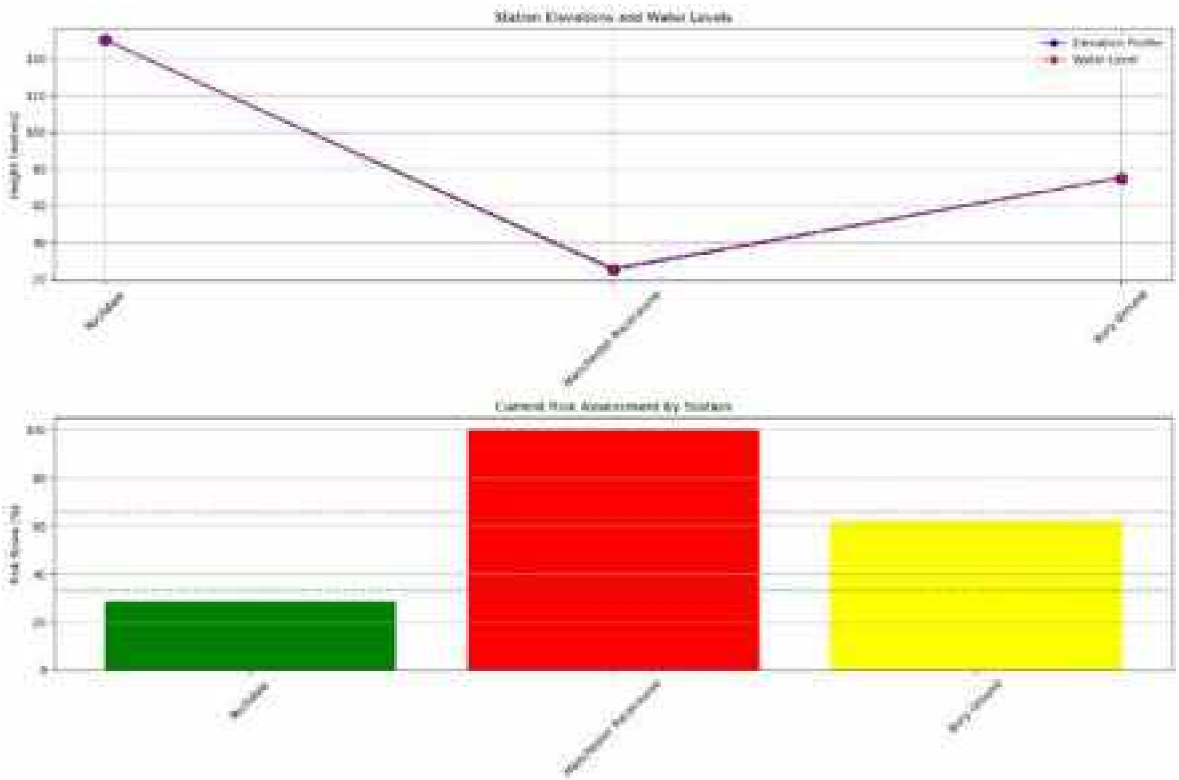
In [32]:
```
# Step 2: Basic Watershed Analysis
print("Analyzing watershed relationships...")

# Calculate elevation differences between connected stations
def calculate_elevation_difference(station_name):
    station = station_info[station_name]
    if station['flow_to']:
        downstream = station_info[station['flow_to']]
        return {
            'from_station': station_name,
            'to_station': station['flow_to'],
            'elevation_diff': station['elevation'] - downstream['elevation'],
            'distance_km': 5  # Example distance, would need actual values
        }
    return None

# Calculate and show elevation differences
print("\nElevation Differences:")
for station in station_info:
    diff = calculate_elevation_difference(station)
    if diff:
        print(f"{diff['from_station']} to {diff['to_station']}:")
        print(f"  Elevation difference: {diff['elevation_diff']}m")
        print(f"  Approximate gradient: {diff['elevation_diff']/diff['distance_k

# Calculate basic risk scores based on current levels and elevation
def calculate_risk_score(station_name):
    station = station_info[station_name]
    current_level = current_levels[station_name]

    # Factor in elevation (lower elevation = higher base risk)
    elevation_factor = 1 - (station['elevation'] / 200)  # Normalize to 0-1

    # Factor in current water level vs typical levels
    level_factor = current_level * 2  # Simple scaling

    # Combine factors (simple weighted average)
    risk_score = (elevation_factor * 0.6 + level_factor * 0.4) * 100
    return min(100, max(0, risk_score))  # Ensure 0-100 range

# Calculate and show risk scores
print("\nCurrent Risk Assessment:")
for station in station_info:
    risk_score = calculate_risk_score(station)
    print(f"{station}:")
    print(f"  Risk Score: {risk_score:.1f}%")
```

```
Analyzing watershed relationships...

Elevation Differences:
Rochdale to Manchester Racecourse:
  Elevation difference: 125m
  Approximate gradient: 25.00m/km
Manchester Racecourse to Bury Ground:
  Elevation difference: -50m
  Approximate gradient: -10.00m/km

Current Risk Assessment:
Rochdale:
  Risk Score: 28.4%
Manchester Racecourse:
  Risk Score: 100.0%
Bury Ground:
  Risk Score: 62.3%
```

In [33]:

```
# Step 3: Visualization of Watershed Analysis
import matplotlib.pyplot as plt

# Create a figure with multiple subplots
plt.figure(figsize=(15, 10))

# Plot 1: Elevation Profile
plt.subplot(2, 1, 1)
stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']
elevations = [station_info[s]['elevation'] for s in stations]
plt.plot(stations, elevations, 'bo-', label='Elevation Profile')
plt.scatter(stations, elevations, color='blue', s=100)

# Add current water levels to elevation plot
current_water_levels = [current_levels[s] + station_info[s]['elevation'] for s i
plt.plot(stations, current_water_levels, 'ro--', label='Water Level')

plt.title('Station Elevations and Water Levels')
plt.ylabel('Height (meters)')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()

# Plot 2: Risk Assessment
plt.subplot(2, 1, 2)
risk_scores = [calculate_risk_score(s) for s in stations]
bars = plt.bar(stations, risk_scores)

# Color code the bars based on risk level
for i, bar in enumerate(bars):
    if risk_scores[i] < 33:
        bar.set_color('green')
    elif risk_scores[i] < 66:
        bar.set_color('yellow')
    else:
        bar.set_color('red')

plt.title('Current Risk Assessment by Station')
plt.ylabel('Risk Score (%)')
plt.xticks(rotation=45)
plt.grid(True, axis='y')
```

```
# Add risk level lines
plt.axhline(y=33, color='green', linestyle='--', alpha=0.5)
plt.axhline(y=66, color='red', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()

# Print analysis summary
print("\nWatershed Analysis Summary:")
print("-" * 50)
for station in stations:
    print(f"\n{station}:")
    print(f"  Elevation: {station_info[station]['elevation']}m")
    print(f"  Current Water Level: {current_levels[station]:.3f}m")
    print(f"  Catchment Area: {station_info[station]['catchment_area']} km²")
    print(f"  Risk Score: {calculate_risk_score(station):.1f}%")
    if station_info[station]['flow_to']:
        print(f"  Flows to: {station_info[station]['flow_to']}")
```



```
Watershed Analysis Summary:
--------------------------------------------------

Rochdale:
  Elevation: 150m
  Current Water Level: 0.168m
  Catchment Area: 12.5 km²
  Risk Score: 28.4%
  Flows to: Manchester Racecourse

Manchester Racecourse:
  Elevation: 25m
  Current Water Level: 0.928m
  Catchment Area: 15.3 km²
  Risk Score: 100.0%
  Flows to: Bury Ground

Bury Ground:
  Elevation: 75m
  Current Water Level: 0.310m
  Catchment Area: 18.7 km²
  Risk Score: 62.3%
```

In [35]:
```
# Step 4: Test watershed utilities in notebook
class WatershedAnalysis:
    def __init__(self):
        # Station information with elevation and catchment area
        self.station_info = {
            'Rochdale': {
                'elevation': 150,
                'catchment_area': 12.5,
                'flow_to': 'Manchester Racecourse'
            },
            'Manchester Racecourse': {
                'elevation': 25,
                'catchment_area': 15.3,
                'flow_to': 'Bury Ground'
            },
            'Bury Ground': {
                'elevation': 75,
                'catchment_area': 18.7,
                'flow_to': None
            }
        }

    def calculate_risk_score(self, station_name, current_level):
        """Calculate risk score for a station"""
        station = self.station_info[station_name]

        # Factor in elevation (lower elevation = higher base risk)
        elevation_factor = 1 - (station['elevation'] / 200)

        # Factor in current water level
        level_factor = current_level * 2

        # Combine factors
        risk_score = (elevation_factor * 0.6 + level_factor * 0.4) * 100
        return min(100, max(0, risk_score))

    def get_flow_path(self, station_name):
```

```
        """Get downstream flow path for a station"""
        if self.station_info[station_name]['flow_to']:
            next_station = self.station_info[station_name]['flow_to']
            elevation_diff = (self.station_info[station_name]['elevation'] -
                              self.station_info[next_station]['elevation'])
            return {
                'next_station': next_station,
                'elevation_diff': elevation_diff
            }
        return None

    def get_station_info(self, station_name):
        """Get all information for a station"""
        return {
            'elevation': self.station_info[station_name]['elevation'],
            'catchment_area': self.station_info[station_name]['catchment_area'],
            'flow_to': self.station_info[station_name]['flow_to']
        }

# Test the class
watershed = WatershedAnalysis()

print("Testing Watershed Analysis:")
print("-" * 50)

for station in ['Rochdale', 'Manchester Racecourse', 'Bury Ground']:
    print(f"\nAnalyzing {station}:")

    # Get station info
    info = watershed.get_station_info(station)
    print(f"Station Info: {info}")

    # Calculate risk
    current_level = current_levels[station]
    risk = watershed.calculate_risk_score(station, current_level)
    print(f"Risk Score: {risk:.1f}%")

    # Get flow path
    flow = watershed.get_flow_path(station)
    if flow:
        print(f"Flows to: {flow['next_station']} (elevation diff: {flow['elevati
```

```
Testing Watershed Analysis:
--------------------------------------------------

Analyzing Rochdale:
Station Info: {'elevation': 150, 'catchment_area': 12.5, 'flow_to': 'Manchester R
acecourse'}
Risk Score: 28.4%
Flows to: Manchester Racecourse (elevation diff: 125m)

Analyzing Manchester Racecourse:
Station Info: {'elevation': 25, 'catchment_area': 15.3, 'flow_to': 'Bury Ground'}
Risk Score: 100.0%
Flows to: Bury Ground (elevation diff: -50m)

Analyzing Bury Ground:
Station Info: {'elevation': 75, 'catchment_area': 18.7, 'flow_to': None}
Risk Score: 62.3%
```

In [42]:
```
# Test watershed analysis without database connection
from watershed_utils import WatershedAnalysis

print("Testing Watershed Analysis:")
print("-" * 50)

# Create analyzer
watershed = WatershedAnalysis()

# Test with sample current levels
test_levels = {
    'Rochdale': 0.168,
    'Manchester Racecourse': 0.928,
    'Bury Ground': 0.310
}

# Test analysis
print("\nWatershed Analysis Results:")
for station, level in test_levels.items():
    print(f"\n{station}:")
    risk = watershed.calculate_risk_score(station, level)
    flow = watershed.get_flow_path(station)
    info = watershed.get_station_info(station)

    print(f"Current Level: {level:.3f}m")
    print(f"Risk Score: {risk:.1f}%")
    print(f"Catchment Area: {info['catchment_area']} km²")
    print(f"Elevation: {info['elevation']}m")
    if flow:
        print(f"Flows to: {flow['next_station']} (elevation diff: {flow['elevati
```

```
Testing Watershed Analysis:
--------------------------------------------------

Watershed Analysis Results:

Rochdale:
Current Level: 0.168m
Risk Score: 28.4%
Catchment Area: 12.5 km²
Elevation: 150m
Flows to: Manchester Racecourse (elevation diff: 125m)

Manchester Racecourse:
Current Level: 0.928m
Risk Score: 100.0%
Catchment Area: 15.3 km²
Elevation: 25m
Flows to: Bury Ground (elevation diff: -50m)

Bury Ground:
Current Level: 0.310m
Risk Score: 62.3%
Catchment Area: 18.7 km²
Elevation: 75m
```

Alert System Implementation

In [44]:
```python
# Test the enhanced alert system
from alert_utils import FloodAlertSystem

alert_system = FloodAlertSystem()

# Test with sample data including trends
test_data = {
    'Rochdale': {'level': 0.168, 'trend': 'Stable'},
    'Manchester Racecourse': {'level': 0.945, 'trend': 'Rising'},
    'Bury Ground': {'level': 0.310, 'trend': 'Falling'}
}

print("Enhanced Alert System Test:")
print("-" * 50)
for station, data in test_data.items():
    alert = alert_system.check_alert_conditions(station, data['level'], data['tr
    print(f"\n{station}:")
    print(f"Current Level: {alert['level']:.3f}m")
    print(f"Status: {alert['status']}")
    print(f"Message: {alert['message']}")
    print(f"Trend: {alert['trend']}")

# Test alert history
print("\nAlert History:")
print(alert_system.get_alert_history())
```

```
Enhanced Alert System Test:
--------------------------------------------------

Rochdale:
Current Level: 0.168m
Status: NORMAL
Message: Normal conditions
Trend: Stable

Manchester Racecourse:
Current Level: 0.945m
Status: ALERT
Message: Prepare for potential flooding
Trend: Rising

Bury Ground:
Current Level: 0.310m
Status: NORMAL
Message: Normal conditions
Trend: Falling

Alert History:
                   timestamp                station  level  status  \
0 2025-02-13 23:53:44.703985               Rochdale  0.168  NORMAL
1 2025-02-13 23:53:44.705980  Manchester Racecourse  0.945   ALERT
2 2025-02-13 23:53:44.705980            Bury Ground  0.310  NORMAL

                          message    trend
0               Normal conditions   Stable
1  Prepare for potential flooding   Rising
2               Normal conditions  Falling
```

Advanced Analytics Implementation

In [49]:
```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Create sample data for testing
def create_test_data():
    dates = pd.date_range(start='2025-02-01', end='2025-02-14', freq='H')
    stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

    data = []
    for station in stations:
        # Create realistic base levels for each station
        if station == 'Rochdale':
            base_level = 0.168
        elif station == 'Manchester Racecourse':
            base_level = 0.928
        else:  # Bury Ground
            base_level = 0.311

        for date in dates:
            # Add some random variation
            level = base_level + np.random.normal(0, 0.01)
            rainfall = max(0, np.random.normal(0, 0.1))

            data.append({
                'river_timestamp': date,
                'location_name': station,
                'river_level': level,
                'rainfall': rainfall
            })

    return pd.DataFrame(data)

# Create test data
print("Creating test data...")
test_data = create_test_data()

print("\nAnalyzing Station Patterns:")
print("-" * 50)

# Analyze each station
for station in test_data['location_name'].unique():
    print(f"\n{station}:")
    station_data = test_data[test_data['location_name'] == station].copy()

    # Basic statistics
    avg_level = station_data['river_level'].mean()
    std_level = station_data['river_level'].std()
    max_level = station_data['river_level'].max()
    min_level = station_data['river_level'].min()

    print(f"Average Level: {avg_level:.3f}m")
    print(f"Std Deviation: {std_level:.3f}m")
    print(f"Range: {min_level:.3f}m to {max_level:.3f}m")

    # Recent trend (last 24 hours)
    recent_data = station_data.tail(24)
    trend = recent_data['river_level'].diff().mean()
```

```
    if abs(trend) < 0.0001:
        trend_direction = "Stable"
    elif trend > 0:
        trend_direction = "Rising"
    else:
        trend_direction = "Falling"

    print(f"Recent Trend: {trend_direction} ({trend:.6f}m/hour)")

    # Correlation with rainfall
    correlation = station_data['river_level'].corr(station_data['rainfall'])
    print(f"Rainfall Correlation: {correlation:.3f}")
```

```
Creating test data...

Analyzing Station Patterns:
--------------------------------------------------

Rochdale:
Average Level: 0.168m
Std Deviation: 0.010m
Range: 0.143m to 0.197m
Recent Trend: Falling (-0.000468m/hour)
Rainfall Correlation: 0.011

Manchester Racecourse:
Average Level: 0.927m
Std Deviation: 0.010m
Range: 0.902m to 0.958m
Recent Trend: Falling (-0.000320m/hour)
Rainfall Correlation: -0.011

Bury Ground:
Average Level: 0.311m
Std Deviation: 0.010m
Range: 0.283m to 0.336m
Recent Trend: Falling (-0.000440m/hour)
Rainfall Correlation: -0.012
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\1649427310.py:7: Future
Warning: 'H' is deprecated and will be removed in a future version, please use
'h' instead.
  dates = pd.date_range(start='2025-02-01', end='2025-02-14', freq='H')
```

Machine Learning Integration

In [51]:

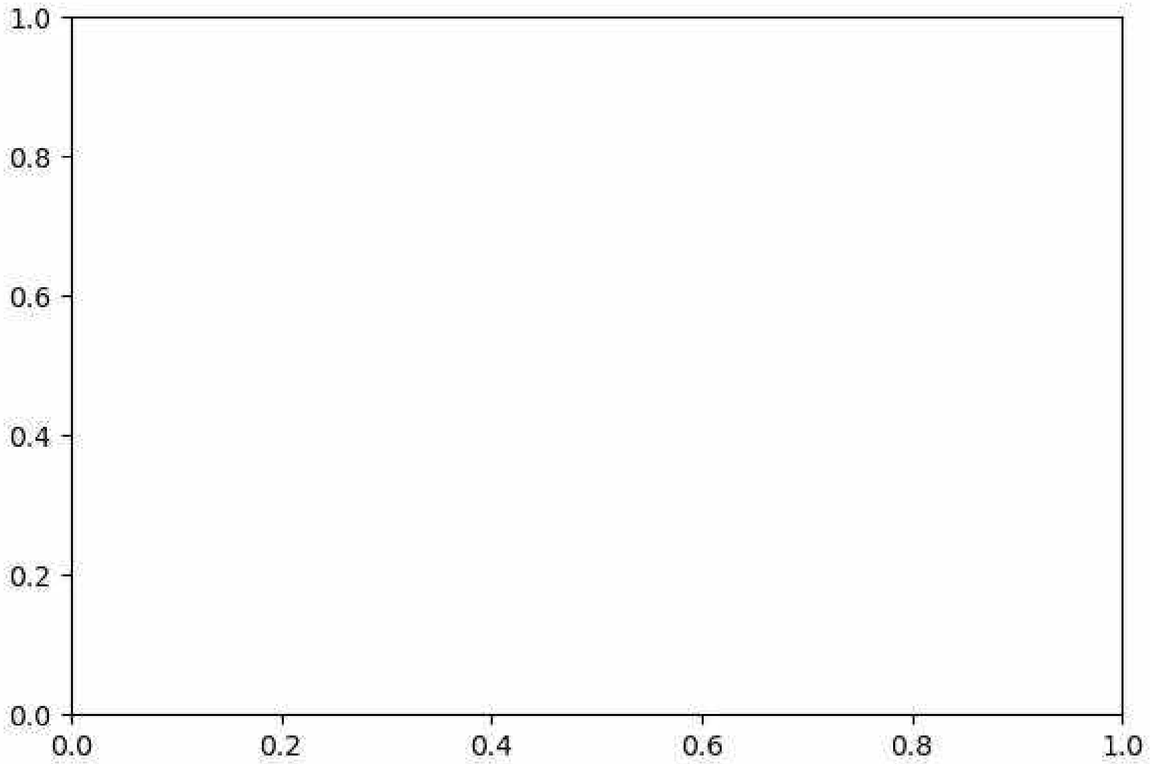
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from supabase import create_client
import pytz
from datetime import datetime, timedelta

# Data Fetching Function
def fetch_river_data(days_back=30):
    """Fetch river monitoring data"""
```

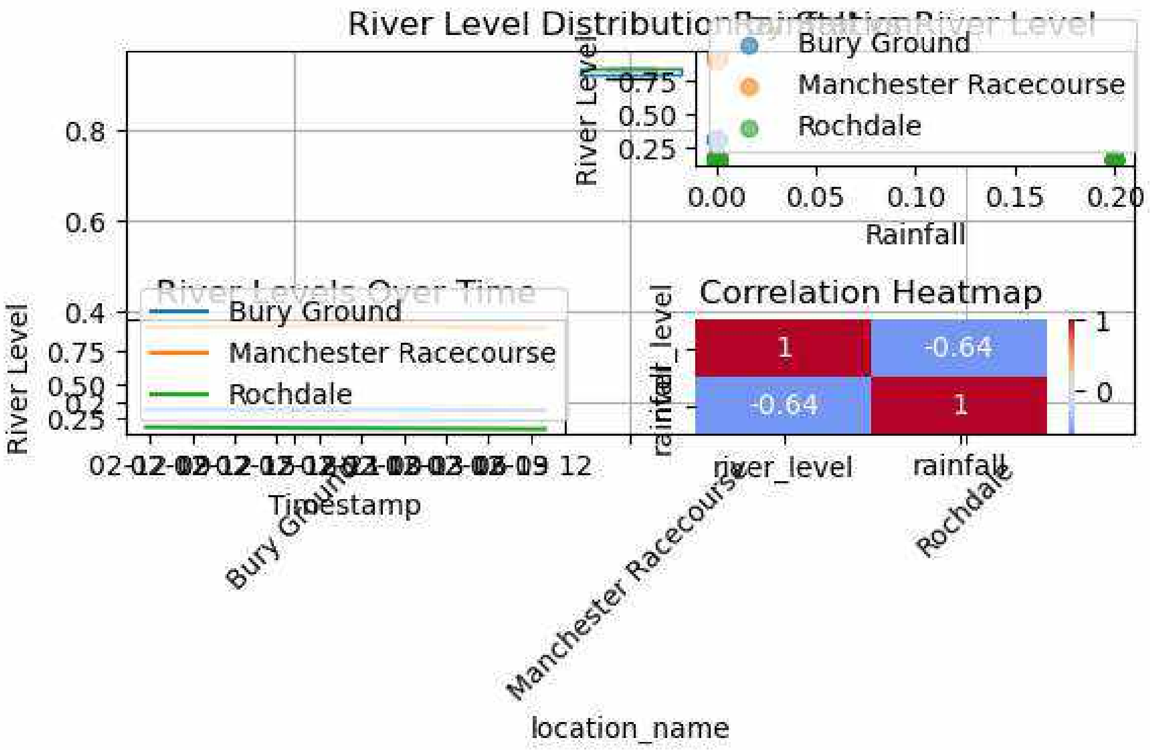
```
    try:
        # Supabase connection details
        supabase_url = "https://thoqlquxaemyyhmpiwzt.supabase.co"
        supabase_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmF

        # Create Supabase client
        supabase = create_client(supabase_url, supabase_key)

        # Define date range
        end_date = datetime.now(pytz.UTC)
        start_date = end_date - timedelta(days=days_back)

        # Fetch data
        response = supabase.table('river_data')\
            .select('*')\
            .gte('river_timestamp', start_date.isoformat())\
            .lte('river_timestamp', end_date.isoformat())\
            .order('river_timestamp', desc=True)\
            .execute()

        # Convert to DataFrame
        if response.data:
            df = pd.DataFrame(response.data)
            df['river_timestamp'] = pd.to_datetime(df['river_timestamp'], utc=Tr
            return df
        else:
            print("No data found")
            return None

    except Exception as e:
        print(f"Error fetching data: {e}")
        return None

# Fetch the data
df = fetch_river_data()

# Comprehensive Data Exploration
if df is not None:
    # Basic data exploration
    print("Dataset Overview:")
    print(df.info())

    # Check for missing values
    print("\nMissing Values:")
    print(df.isnull().sum())

    # Basic statistical summary
    print("\nStatistical Summary:")
    print(df.describe())

    # Visualization of River Levels
    plt.figure(figsize=(15, 10))

    # Boxplot of River Levels by Station
    plt.subplot(2, 2, 1)
    df.boxplot(column='river_level', by='location_name')
    plt.title('River Level Distribution by Station')
    plt.suptitle('')
    plt.xticks(rotation=45)
```

```
    # Scatter plot of River Level vs Rainfall
    plt.subplot(2, 2, 2)
    for station in df['location_name'].unique():
        station_data = df[df['location_name'] == station]
        plt.scatter(station_data['rainfall'], station_data['river_level'], label
    plt.xlabel('Rainfall')
    plt.ylabel('River Level')
    plt.title('Rainfall vs River Level')
    plt.legend()

    # Time Series of River Levels
    plt.subplot(2, 2, 3)
    for station in df['location_name'].unique():
        station_data = df[df['location_name'] == station]
        plt.plot(station_data['river_timestamp'], station_data['river_level'], l
    plt.title('River Levels Over Time')
    plt.xlabel('Timestamp')
    plt.ylabel('River Level')
    plt.legend()

    # Correlation Heatmap
    plt.subplot(2, 2, 4)
    correlation_matrix = df.groupby('location_name').agg({
        'river_level': 'mean',
        'rainfall': 'mean'
    }).corr()
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Correlation Heatmap')

    plt.tight_layout()
    plt.show()

    # Detailed Station Analysis
    print("\nDetailed Station Analysis:")
    for station in df['location_name'].unique():
        station_data = df[df['location_name'] == station]
        print(f"\n{station} Station:")
        print("Average River Level:", station_data['river_level'].mean())
        print("Minimum River Level:", station_data['river_level'].min())
        print("Maximum River Level:", station_data['river_level'].max())

        # Temporal patterns
        station_data.set_index('river_timestamp', inplace=True)
        hourly_avg = station_data.resample('H')['river_level'].mean()
        print("Peak Hour:", hourly_avg.idxmax().hour)
        print("Lowest Hour:", hourly_avg.idxmin().hour)
else:
    print("Failed to fetch data")
```

```
2025-02-14 16:07:25,656 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&river_timestamp=gte.2025-01-15T16%3A07%3
A24.751810%2B00%3A00&river_timestamp=lte.2025-02-14T16%3A07%3A24.751810%2B00%3A00
&order=river_timestamp.desc "HTTP/2 200 OK"
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1000 non-null   int64
 1   river_level         1000 non-null   float64
 2   river_timestamp     1000 non-null   datetime64[ns, UTC]
 3   rainfall            1000 non-null   float64
 4   rainfall_timestamp  1000 non-null   object
 5   location_name       1000 non-null   object
 6   river_station_id    1000 non-null   int64
 7   rainfall_station_id 1000 non-null   int64
 8   created_at          1000 non-null   object
dtypes: datetime64[ns, UTC](1), float64(2), int64(3), object(3)
memory usage: 70.4+ KB
None

Missing Values:
id                     0
river_level            0
river_timestamp        0
rainfall               0
rainfall_timestamp     0
location_name          0
river_station_id       0
rainfall_station_id    0
created_at             0
dtype: int64

Statistical Summary:
                id  river_level     rainfall  river_station_id  \
count  1000.000000  1000.000000  1000.000000       1000.000000
mean   3325.268000     0.469476     0.003000     690291.213000
std     289.220709     0.328965     0.024323        155.661132
min    2805.000000     0.166000     0.000000     690160.000000
25%    3075.750000     0.177000     0.000000     690160.000000
50%    3325.500000     0.311000     0.000000     690203.000000
75%    3575.250000     0.921000     0.000000     690510.000000
max    3825.000000     0.936000     0.200000     690510.000000

       rainfall_station_id
count          1000.000000
mean         562412.225000
std             591.163345
min          561613.000000
25%          561613.000000
50%          562656.000000
75%          562992.000000
max          562992.000000
```




River Level Distribution
Rainfall vs River Level
Bury Ground
Manchester Racecourse
Rochdale
River Level
Rainfall
River Levels Over Time
Timestamp
Correlation Heatmap
river_level
rainfall
1
-0.64
-0.64
1
location_name

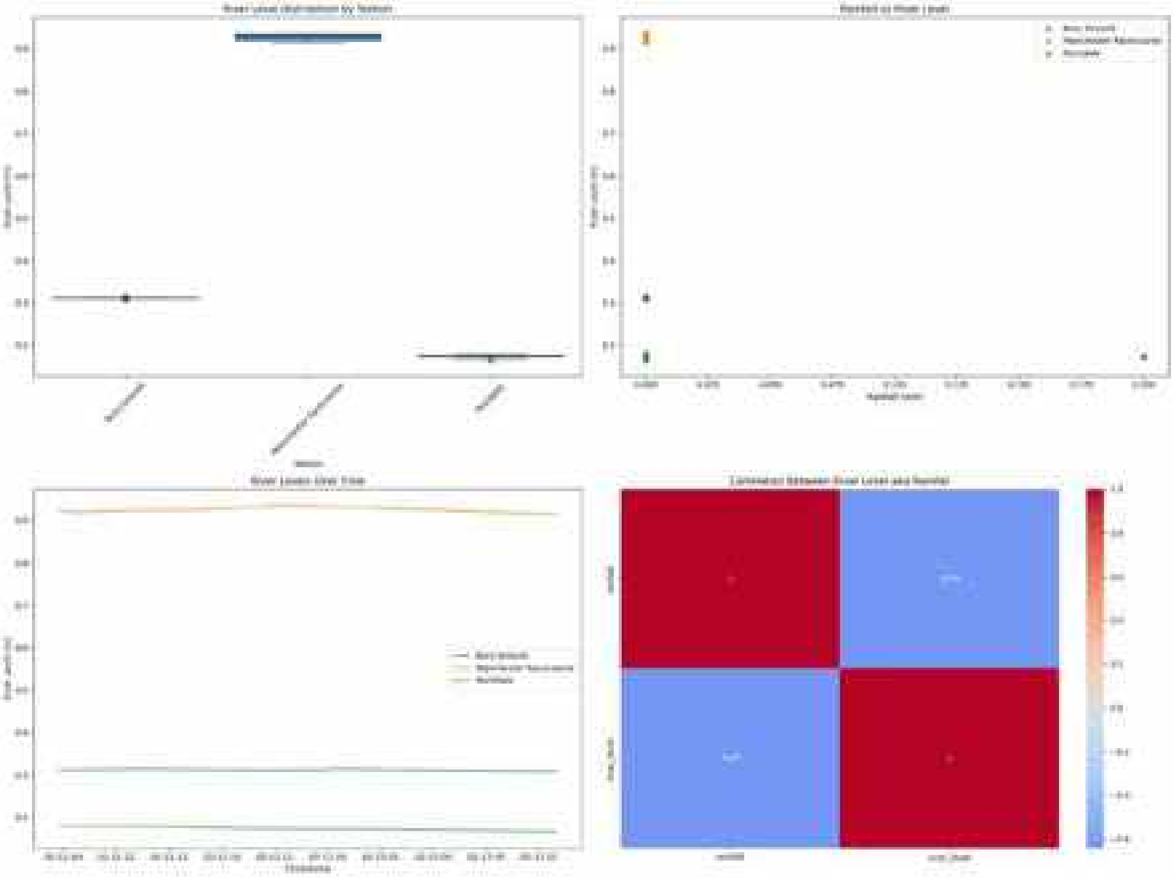
```
Detailed Station Analysis:

Bury Ground Station:
Average River Level: 0.31091717791411044
Minimum River Level: 0.308
Maximum River Level: 0.314
Peak Hour: 13
Lowest Hour: 12

Manchester Racecourse Station:
Average River Level: 0.9279189189189188
Minimum River Level: 0.915
Maximum River Level: 0.936
Peak Hour: 21
Lowest Hour: 13

Rochdale Station:
Average River Level: 0.17337243401759533
Minimum River Level: 0.166
Maximum River Level: 0.18
Peak Hour: 8
Lowest Hour: 13
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2422683967.py:118: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead.
  hourly_avg = station_data.resample('H')['river_level'].mean()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2422683967.py:118: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead.
  hourly_avg = station_data.resample('H')['river_level'].mean()
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2422683967.py:118: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead.
  hourly_avg = station_data.resample('H')['river_level'].mean()
```

In [52]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame from previous analysis
plt.figure(figsize=(20, 15))

# 1. River Level Distribution Boxplot
plt.subplot(2, 2, 1)
sns.boxplot(x='location_name', y='river_level', data=df)
plt.title('River Level Distribution by Station')
plt.xlabel('Station')
plt.ylabel('River Level (m)')
plt.xticks(rotation=45)

# 2. Rainfall vs River Level Scatter Plot
plt.subplot(2, 2, 2)
for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station]
    plt.scatter(station_data['rainfall'], station_data['river_level'],
                label=station, alpha=0.7)
plt.title('Rainfall vs River Level')
plt.xlabel('Rainfall (mm)')
plt.ylabel('River Level (m)')
```

```
plt.legend()

# 3. Time Series of River Levels
plt.subplot(2, 2, 3)
for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station]
    plt.plot(station_data['river_timestamp'], station_data['river_level'],
             label=station)
plt.title('River Levels Over Time')
plt.xlabel('Timestamp')
plt.ylabel('River Level (m)')
plt.legend()

# 4. Detailed Correlation Heatmap
plt.subplot(2, 2, 4)
# Pivot the data to get mean values for each station
correlation_data = df.pivot_table(
    index='location_name',
    values=['river_level', 'rainfall'],
    aggfunc='mean'
)
correlation_matrix = correlation_data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Between River Level and Rainfall')

plt.tight_layout()
plt.show()

# Additional Detailed Correlation Analysis
print("\nDetailed Correlation Analysis:")
for station in df['location_name'].unique():
    station_data = df[df['location_name'] == station]
    correlation = station_data['river_level'].corr(station_data['rainfall'])
    print(f"{station} Station:")
    print(f"Correlation between River Level and Rainfall: {correlation:.4f}")
```



```
Detailed Correlation Analysis:
Bury Ground Station:
Correlation between River Level and Rainfall: nan
Manchester Racecourse Station:
Correlation between River Level and Rainfall: nan
Rochdale Station:
Correlation between River Level and Rainfall: -0.0183
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:289
7: RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:289
8: RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]
```

Machine Learning Integration: Feature Engineering and Preparation.

In [53]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def create_advanced_features(df):
    """
    Create advanced features for machine learning model
    """
    # Create a copy of the dataframe to avoid modifying original data
    data = df.copy()

    # 1. Time-based Features
    data['hour'] = data['river_timestamp'].dt.hour
    data['day_of_week'] = data['river_timestamp'].dt.dayofweek
    data['month'] = data['river_timestamp'].dt.month

    # 2. Rolling Window Features
```

```
def create_rolling_features(group):
    # Sort by timestamp to ensure correct rolling calculation
    group = group.sort_values('river_timestamp')

    # Rolling mean and standard deviation
    group['river_level_rolling_mean_6h'] = group['river_level'].rolling(wind
    group['river_level_rolling_std_6h'] = group['river_level'].rolling(windo

    # Rolling mean and standard deviation for rainfall
    group['rainfall_rolling_mean_6h'] = group['rainfall'].rolling(window=6,
    group['rainfall_rolling_std_6h'] = group['rainfall'].rolling(window=6, m

    return group

data = data.groupby('location_name').apply(create_rolling_features).reset_in

# 3. Lag Features
def create_lag_features(group):
    # Sort by timestamp
    group = group.sort_values('river_timestamp')

    # Create lag features for river level
    group['river_level_lag_1h'] = group['river_level'].shift(1)
    group['river_level_lag_3h'] = group['river_level'].shift(3)

    # Create lag features for rainfall
    group['rainfall_lag_1h'] = group['rainfall'].shift(1)
    group['rainfall_lag_3h'] = group['rainfall'].shift(3)

    return group

data = data.groupby('location_name').apply(create_lag_features).reset_index(

# 4. Interaction Features
data['rainfall_river_level_interaction'] = data['rainfall'] * data['river_le

# 5. Cyclical Encoding for Time Features
def cyclical_encode(df, column, max_val):
    df[f'{column}_sin'] = np.sin(2 * np.pi * df[column] / max_val)
    df[f'{column}_cos'] = np.cos(2 * np.pi * df[column] / max_val)
    return df

data = cyclical_encode(data, 'hour', 24)
data = cyclical_encode(data, 'day_of_week', 7)
data = cyclical_encode(data, 'month', 12)

# 6. Normalization
scaler = StandardScaler()
numeric_columns = [
    'river_level', 'rainfall',
    'river_level_rolling_mean_6h', 'river_level_rolling_std_6h',
    'rainfall_rolling_mean_6h', 'rainfall_rolling_std_6h',
    'river_level_lag_1h', 'river_level_lag_3h',
    'rainfall_lag_1h', 'rainfall_lag_3h'
]

# Scale numeric features
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])

# One-hot encode location name
```

```
    data = pd.get_dummies(data, columns=['location_name'])

    return data

# Prepare features
featured_data = create_advanced_features(df)

# Display feature overview
print("Feature Engineering Results:")
print("Total Features:", len(featured_data.columns))
print("\nNew Features Added:")
print("1. Time-based Features: hour, day of week, month")
print("2. Rolling Window Features: 6-hour rolling mean/std for river level and r
print("3. Lag Features: 1-hour and 3-hour lags for river level and rainfall")
print("4. Interaction Features: rainfall * river level")
print("5. Cyclical Encoding: sin/cos transformations for time features")
print("6. Normalization: StandardScaler applied to numeric features")
print("7. One-hot Encoding: Location names")

# Optional: Preview the first few rows of the featured dataset
print("\nFirst few rows of featured dataset:")
print(featured_data.head())

# Save processed dataset
featured_data.to_csv('advanced_featured_river_data.csv', index=False)
print("\nProcessed dataset saved to 'advanced_featured_river_data.csv'")
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2793506590.py:32: Depre
cationWarning: DataFrameGroupBy.apply operated on the grouping columns. This beha
vior is deprecated, and in a future version of pandas the grouping columns will b
e excluded from the operation. Either pass `include_groups=False` to exclude the
groupings or explicitly select the grouping columns after groupby to silence this
warning.
  data = data.groupby('location_name').apply(create_rolling_features).reset_index
(drop=True)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2793506590.py:49: Depre
cationWarning: DataFrameGroupBy.apply operated on the grouping columns. This beha
vior is deprecated, and in a future version of pandas the grouping columns will b
e excluded from the operation. Either pass `include_groups=False` to exclude the
groupings or explicitly select the grouping columns after groupby to silence this
warning.
  data = data.groupby('location_name').apply(create_lag_features).reset_index(dro
p=True)
```

```
Feature Engineering Results:
Total Features: 29

New Features Added:
1. Time-based Features: hour, day of week, month
2. Rolling Window Features: 6-hour rolling mean/std for river level and rainfall
3. Lag Features: 1-hour and 3-hour lags for river level and rainfall
4. Interaction Features: rainfall * river level
5. Cyclical Encoding: sin/cos transformations for time features
6. Normalization: StandardScaler applied to numeric features
7. One-hot Encoding: Location names

First few rows of featured dataset:
     id  river_level           river_timestamp  rainfall  \
0  2834    -0.478942 2025-02-12 08:45:00+00:00 -0.123404
1  2849    -0.478942 2025-02-12 08:45:00+00:00 -0.123404
2  2843    -0.478942 2025-02-12 08:45:00+00:00 -0.123404
3  2837    -0.478942 2025-02-12 08:45:00+00:00 -0.123404
4  2825    -0.478942 2025-02-12 08:45:00+00:00 -0.123404

          rainfall_timestamp  river_station_id  rainfall_station_id  \
0  2025-02-12T08:45:00+00:00            690160               562656
1  2025-02-12T08:45:00+00:00            690160               562656
2  2025-02-12T08:45:00+00:00            690160               562656
3  2025-02-12T08:45:00+00:00            690160               562656
4  2025-02-12T08:45:00+00:00            690160               562656

                         created_at  hour  day_of_week  ...  \
0  2025-02-12T09:25:16.606445+00:00     8            2  ...
1   2025-02-12T09:30:18.89606+00:00     8            2  ...
2  2025-02-12T09:28:17.957141+00:00     8            2  ...
3  2025-02-12T09:26:17.004951+00:00     8            2  ...
4  2025-02-12T09:22:15.041519+00:00     8            2  ...

   rainfall_river_level_interaction  hour_sin  hour_cos  day_of_week_sin  \
0                               0.0  0.866025      -0.5         0.974928
1                               0.0  0.866025      -0.5         0.974928
2                               0.0  0.866025      -0.5         0.974928
3                               0.0  0.866025      -0.5         0.974928
4                               0.0  0.866025      -0.5         0.974928

   day_of_week_cos  month_sin  month_cos  location_name_Bury Ground  \
0        -0.222521   0.866025        0.5                       True
1        -0.222521   0.866025        0.5                       True
2        -0.222521   0.866025        0.5                       True
3        -0.222521   0.866025        0.5                       True
4        -0.222521   0.866025        0.5                       True

   location_name_Manchester Racecourse  location_name_Rochdale
0                                False                   False
1                                False                   False
2                                False                   False
3                                False                   False
4                                False                   False

[5 rows x 29 columns]

Processed dataset saved to 'advanced_featured_river_data.csv'
```

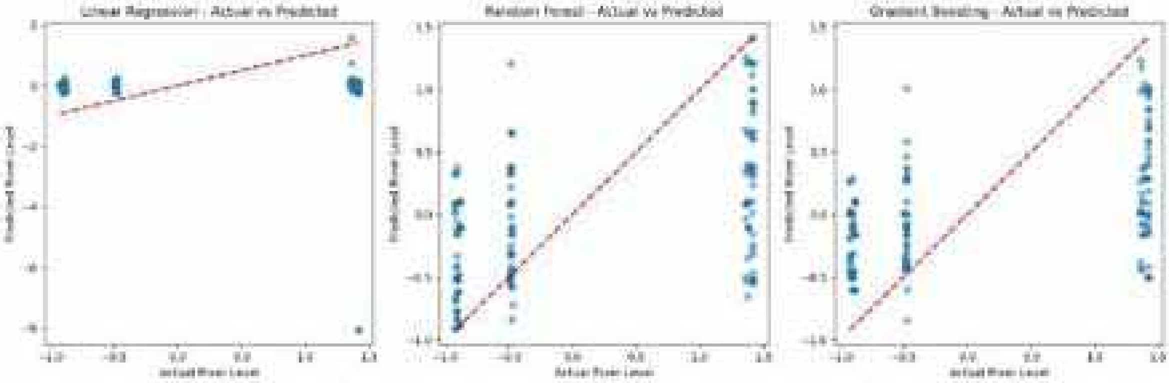
Model Selection and Preparation

In [56]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Load the featured dataset
featured_data = pd.read_csv('advanced_featured_river_data.csv')

# Data Preparation Function
def prepare_ml_dataset(df):
    # Remove timestamp-related columns and id columns
    features = df.drop([
        'river_timestamp', 'id', 'created_at', 'rainfall_timestamp',
        'river_station_id', 'rainfall_station_id'
    ], axis=1)

    # Sort by timestamp to ensure correct lag creation
    df = df.sort_values('river_timestamp')

    # Target variable: next time step's river level
    target = df['river_level'].shift(-1)

    # Remove the last row (which will have NaN target)
    features = features[:-1]
    target = target[:-1]

    return features, target

# Prepare features and target
X, y = prepare_ml_dataset(featured_data)

# Check for NaN values
print("NaN values in features:")
print(X.isna().sum())
print("\nNaN values in target:")
print(y.isna().sum())

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Model Evaluation Function with Imputation
def evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
    # Create a pipeline with imputation and the model
    pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),  # Replace NaNs with medi
        ('model', model)
    ])

    # Train the model
    pipeline.fit(X_train, y_train)

    # Predictions
    y_pred = pipeline.predict(X_test)
```

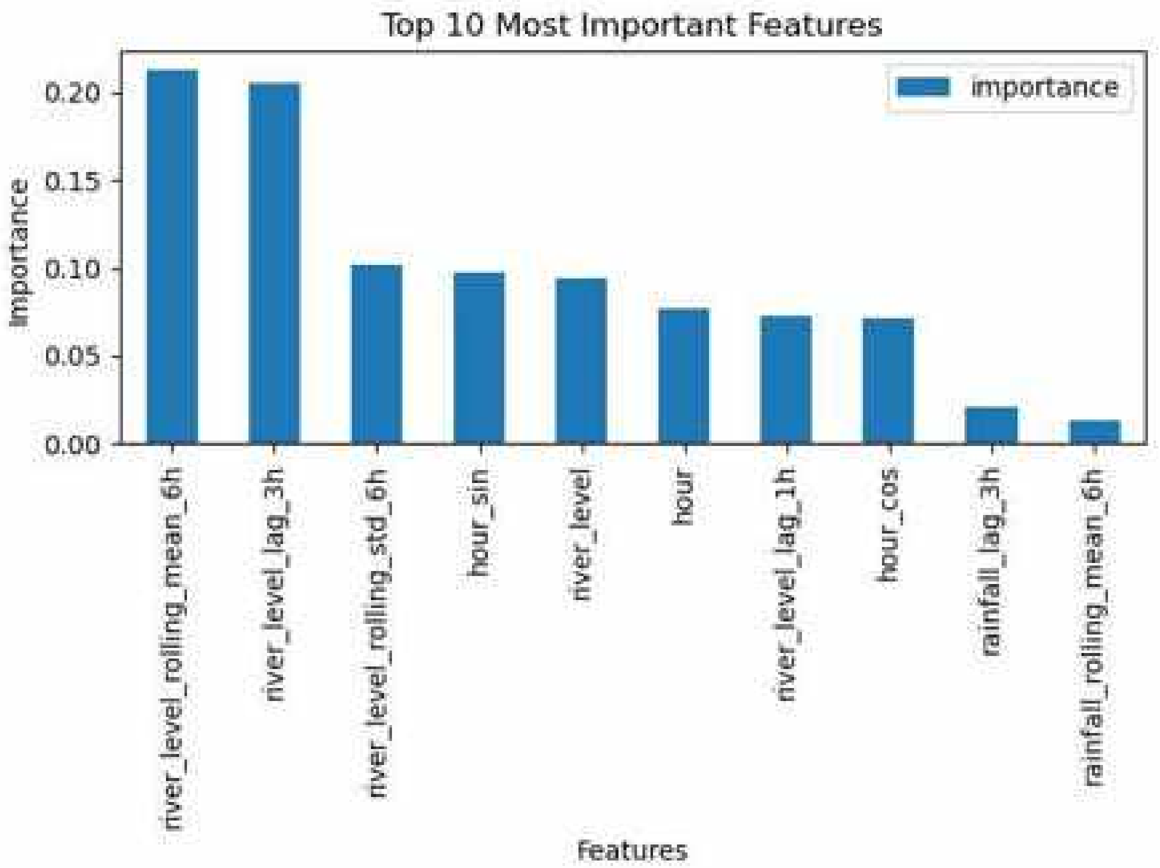
```
    # Evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"\n{model_name} Model Performance:")
    print(f"Mean Squared Error: {mse:.4f}")
    print(f"Mean Absolute Error: {mae:.4f}")
    print(f"R-squared Score: {r2:.4f}")

    return pipeline, y_pred

# Models to evaluate
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor

models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42)
}

# Store results
model_results = {}

# Evaluate models
for name, model in models.items():
    trained_model, predictions = evaluate_model(model, X_train, X_test, y_train,
    model_results[name] = {
        'model': trained_model,
        'predictions': predictions
    }

# Visualization of Actual vs Predicted
plt.figure(figsize=(15, 5))

for i, (name, result) in enumerate(model_results.items(), 1):
    plt.subplot(1, 3, i)
    plt.scatter(y_test, result['predictions'], alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',
    plt.xlabel('Actual River Level')
    plt.ylabel('Predicted River Level')
    plt.title(f'{name} - Actual vs Predicted')

plt.tight_layout()
plt.show()

# Feature Importance for Random Forest
rf_model = model_results['Random Forest']['model'].named_steps['model']
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': rf_model.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(12, 6))
feature_importance.head(10).plot(x='feature', y='importance', kind='bar')
plt.title('Top 10 Most Important Features')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.tight_layout()
```

```python
plt.show()

print("\nTop 10 Most Important Features:")
print(feature_importance.head(10))
```

```
NaN values in features:
river_level                           0
rainfall                              0
hour                                  0
day_of_week                           0
month                                 0
river_level_rolling_mean_6h           0
river_level_rolling_std_6h            3
rainfall_rolling_mean_6h              0
rainfall_rolling_std_6h               3
river_level_lag_1h                    3
river_level_lag_3h                    9
rainfall_lag_1h                       3
rainfall_lag_3h                       9
rainfall_river_level_interaction      0
hour_sin                              0
hour_cos                              0
day_of_week_sin                       0
day_of_week_cos                       0
month_sin                             0
month_cos                             0
location_name_Bury Ground             0
location_name_Manchester Racecourse   0
location_name_Rochdale                0
dtype: int64

NaN values in target:
0

Linear Regression Model Performance:
Mean Squared Error: 1.9121
Mean Absolute Error: 1.0207
R-squared Score: -0.8135

Random Forest Model Performance:
Mean Squared Error: 0.7994
Mean Absolute Error: 0.7042
R-squared Score: 0.2418

Gradient Boosting Model Performance:
Mean Squared Error: 0.8117
Mean Absolute Error: 0.7592
R-squared Score: 0.2302
```




```
<Figure size 1200x600 with 0 Axes>
```




```
Top 10 Most Important Features:
                        feature  importance
5   river_level_rolling_mean_6h    0.212380
10           river_level_lag_3h    0.205451
6    river_level_rolling_std_6h    0.101183
14                     hour_sin    0.096963
0                   river_level    0.094153
2                          hour    0.076595
9            river_level_lag_1h    0.072051
15                     hour_cos    0.071520
12              rainfall_lag_3h    0.021023
7      rainfall_rolling_mean_6h    0.013567
```

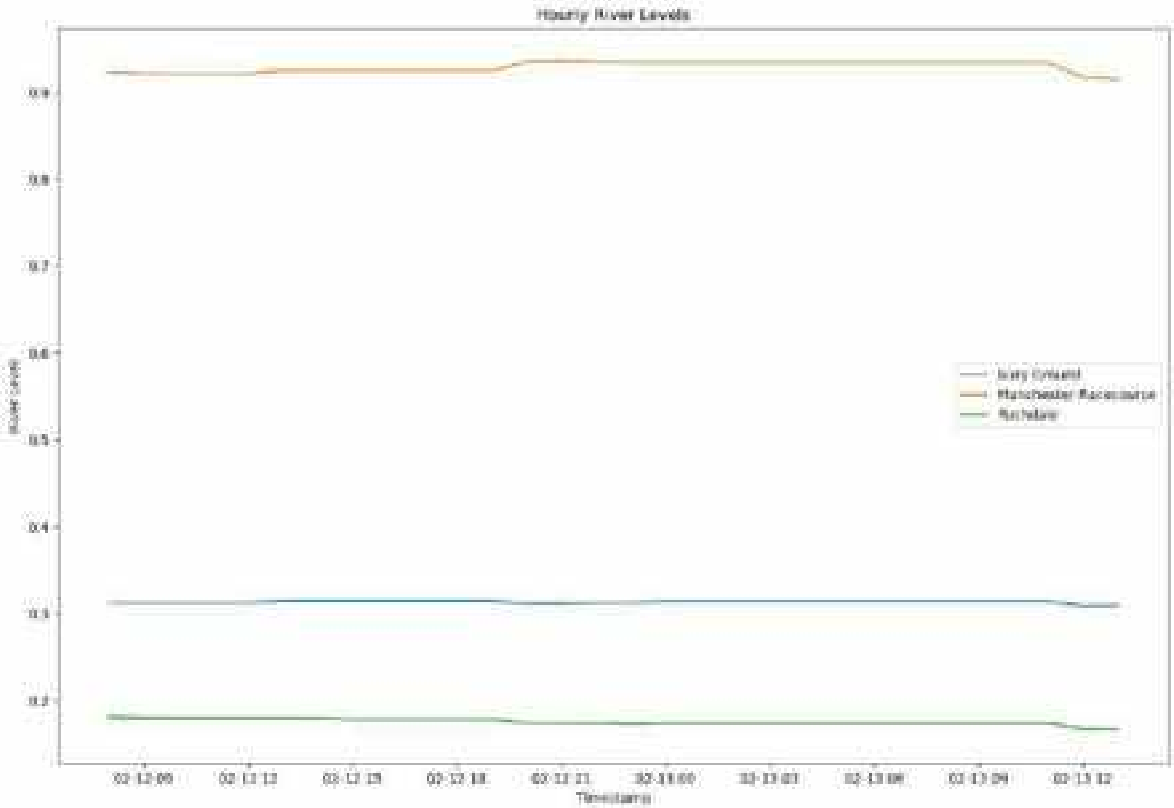
Advanced Time Series Preparation

In [57]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import pytz

# Load data from your existing dashboard connection method
def fetch_river_data(days_back=30):
    """Fetch river monitoring data"""
    try:
        # Supabase connection details
        supabase_url = "https://thoqlquxaemyyhmpiwzt.supabase.co"
        supabase_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmF

        # Create Supabase client
        supabase = create_client(supabase_url, supabase_key)

        # Define date range
```

```
        end_date = datetime.now(pytz.UTC)
        start_date = end_date - timedelta(days=days_back)

        # Fetch data
        response = supabase.table('river_data')\
            .select('*')\
            .gte('river_timestamp', start_date.isoformat())\
            .lte('river_timestamp', end_date.isoformat())\
            .order('river_timestamp', desc=True)\
            .execute()

        # Convert to DataFrame
        if response.data:
            df = pd.DataFrame(response.data)
            df['river_timestamp'] = pd.to_datetime(df['river_timestamp'], utc=Tr
            return df
        else:
            print("No data found")
            return None

    except Exception as e:
        print(f"Error fetching data: {e}")
        return None

# Fetch the data
df = fetch_river_data()

# Prepare Time Series Data for Each Station
def prepare_time_series(df, station):
    station_data = df[df['location_name'] == station].copy()
    station_data.set_index('river_timestamp', inplace=True)
    station_data = station_data.resample('H')['river_level'].mean().fillna(metho
    return station_data

# Analysis for Each Station
stations = df['location_name'].unique()
time_series_data = {station: prepare_time_series(df, station) for station in sta

# Visualization of Time Series
plt.figure(figsize=(15, 10))
for station, data in time_series_data.items():
    plt.plot(data.index, data.values, label=station)
plt.title('Hourly River Levels')
plt.xlabel('Timestamp')
plt.ylabel('River Level')
plt.legend()
plt.show()

# Seasonal Decomposition Function
def decompose_time_series(time_series):
    try:
        from statsmodels.tsa.seasonal import seasonal_decompose

        # Handling short time series
        if len(time_series) < 24:
            print("Time series too short for decomposition")
            return None

        decomposition = seasonal_decompose(time_series, period=24)
```

```
        plt.figure(figsize=(15, 10))
        plt.subplot(411)
        plt.title('Original Time Series')
        plt.plot(decomposition.observed)

        plt.subplot(412)
        plt.title('Trend')
        plt.plot(decomposition.trend)

        plt.subplot(413)
        plt.title('Seasonal')
        plt.plot(decomposition.seasonal)

        plt.subplot(414)
        plt.title('Residual')
        plt.plot(decomposition.resid)

        plt.tight_layout()
        plt.show()

        return decomposition
    except Exception as e:
        print(f"Error in decomposition: {e}")
        return None

# Decompose each station's time series
decompositions = {station: decompose_time_series(data) for station, data in time

# Additional Time Series Analysis
print("\nTime Series Statistics:")
for station, data in time_series_data.items():
    print(f"\n{station} Station:")
    print("Mean River Level:", data.mean())
    print("Standard Deviation:", data.std())
    print("Minimum Level:", data.min())
    print("Maximum Level:", data.max())
```

```
2025-02-14 18:22:30,490 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&river_timestamp=gte.2025-01-15T18%3A22%3
A29.638432%2B00%3A00&river_timestamp=lte.2025-02-14T18%3A22%3A29.638432%2B00%3A00
&order=river_timestamp.desc "HTTP/2 200 OK"
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\1832776621.py:50: Futur
eWarning: 'H' is deprecated and will be removed in a future version, please use
'h' instead.
  station_data = station_data.resample('H')['river_level'].mean().fillna(method
='ffill')
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\1832776621.py:50: Futur
eWarning: Series.fillna with 'method' is deprecated and will raise in a future ve
rsion. Use obj.ffill() or obj.bfill() instead.
  station_data = station_data.resample('H')['river_level'].mean().fillna(method
='ffill')
```




```
Error in decomposition: x must have 2 complete cycles requires 48 observations. x
only has 30 observation(s)
Error in decomposition: x must have 2 complete cycles requires 48 observations. x
only has 30 observation(s)
Error in decomposition: x must have 2 complete cycles requires 48 observations. x
only has 30 observation(s)

Time Series Statistics:

Bury Ground Station:
Mean River Level: 0.3127205957883924
Standard Deviation: 0.0014355947844961559
Minimum Level: 0.3082542372881356
Maximum Level: 0.314

Manchester Racecourse Station:
Mean River Level: 0.9287454747935681
Standard Deviation: 0.006373925960907178
Minimum Level: 0.915
Maximum Level: 0.9358557692307693

Rochdale Station:
Mean River Level: 0.17464096045197744
Standard Deviation: 0.003522007012308529
Minimum Level: 0.166
Maximum Level: 0.18
```

Advanced Time Series Forecasting

In [60]:

```
# Import required libraries
import pandas as pd
import numpy as np
from supabase import create_client
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
```

```
# Supabase Connection Details
SUPABASE_URL = "https://thoqlquxaemyyhmpiwzt.supabase.co"
SUPABASE_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ

# Create Supabase client
supabase = create_client(SUPABASE_URL, SUPABASE_KEY)

# Fetch river data
print("Fetching data from Supabase...")
response = supabase.table('river_data').select('*').execute()

# Convert to DataFrame
df = pd.DataFrame(response.data)

# Convert timestamp
df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

# Basic data exploration
print("\nDataset Overview:")
print(df.info())

# Check unique stations
print("\nMonitoring Stations:")
print(df['location_name'].unique())

# Basic statistical summary
print("\nStatistical Summary:")
print(df.groupby('location_name')['river_level'].describe())
```

```
Fetching data from Supabase...
```

```
2025-02-14 18:31:11,966 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A "HTTP/2 200 OK"
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   1000 non-null   int64
 1   river_level          1000 non-null   float64
 2   river_timestamp      1000 non-null   datetime64[ns, UTC]
 3   rainfall             1000 non-null   float64
 4   rainfall_timestamp   1000 non-null   object
 5   location_name        1000 non-null   object
 6   river_station_id     1000 non-null   int64
 7   rainfall_station_id  1000 non-null   int64
 8   created_at           1000 non-null   object
dtypes: datetime64[ns, UTC](1), float64(2), int64(3), object(3)
memory usage: 70.4+ KB
None

Monitoring Stations:
['Rochdale' 'Manchester Racecourse' 'Bury Ground']

Statistical Summary:
                       count      mean       std    min    25%    50%    75%  \
location_name
Bury Ground            333.0  0.311318  0.002994  0.309  0.309  0.309  0.314
Manchester Racecourse  333.0  0.931652  0.006193  0.923  0.927  0.929  0.939
Rochdale               334.0  0.167243  0.001053  0.165  0.167  0.167  0.168

                         max
location_name
Bury Ground            0.316
Manchester Racecourse  0.942
Rochdale               0.169
```

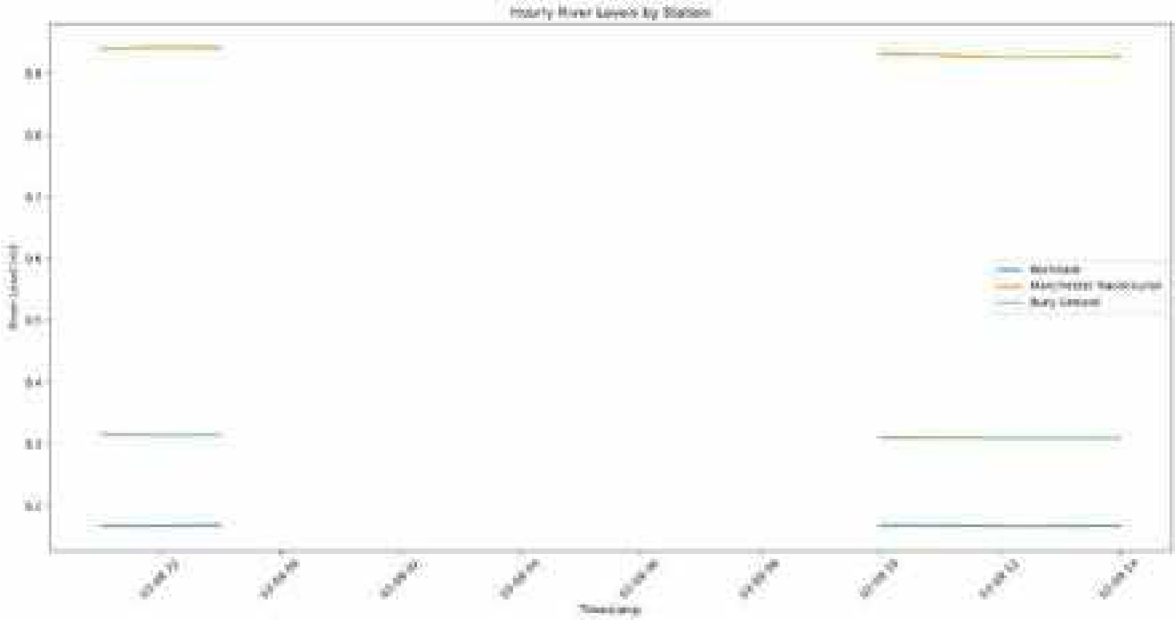
In [61]:

```python
# Time Series Preparation and Analysis
def prepare_time_series(df, station):
    # Filter data for specific station
    station_data = df[df['location_name'] == station].copy()

    # Sort by timestamp
    station_data = station_data.sort_values('river_timestamp')

    # Set timestamp as index
    station_data.set_index('river_timestamp', inplace=True)

    # Resample to hourly data (if needed)
    hourly_data = station_data['river_level'].resample('H').mean()

    return hourly_data

# Analyze each station
stations = df['location_name'].unique()
time_series_data = {}

print("Time Series Preparation:")
for station in stations:
    print(f"\n--- {station} Station ---")

    # Prepare time series
```

```
    ts_data = prepare_time_series(df, station)
    time_series_data[station] = ts_data

    # Basic time series statistics
    print("Total Observations:", len(ts_data))
    print("Mean River Level:", ts_data.mean())
    print("Standard Deviation:", ts_data.std())
    print("Minimum Level:", ts_data.min())
    print("Maximum Level:", ts_data.max())

# Visualization of Time Series
plt.figure(figsize=(15, 8))
for station, data in time_series_data.items():
    plt.plot(data.index, data.values, label=station)

plt.title('Hourly River Levels by Station')
plt.xlabel('Timestamp')
plt.ylabel('River Level (m)')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Temporal Pattern Analysis
print("\nTemporal Pattern Analysis:")
for station, data in time_series_data.items():
    print(f"\n{station} Station:")

    # Hour of day analysis
    hourly_pattern = data.groupby(data.index.hour).mean()
    print("\nAverage Levels by Hour:")
    print(hourly_pattern)

    # Day of week analysis
    daily_pattern = data.groupby(data.index.dayofweek).mean()
    print("\nAverage Levels by Day of Week:")
    print(daily_pattern)
```

```
Time Series Preparation:

--- Rochdale Station ---
Total Observations: 18
Mean River Level: 0.16741610537251092
Standard Deviation: 0.0006753741786603536
Minimum Level: 0.16674576271186442
Maximum Level: 0.16870000000000002

--- Manchester Racecourse Station ---
Total Observations: 18
Mean River Level: 0.932877219210874
Standard Deviation: 0.006823644213694164
Minimum Level: 0.92586
Maximum Level: 0.942

--- Bury Ground Station ---
Total Observations: 18
Mean River Level: 0.3113801906779661
Standard Deviation: 0.002880268735833967
Minimum Level: 0.30899999999999994
Maximum Level: 0.316
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11076\2217573701.py:13: Futur
eWarning: 'H' is deprecated and will be removed in a future version, please use
'h' instead.
  hourly_data = station_data['river_level'].resample('H').mean()
```



```
Temporal Pattern Analysis:

Rochdale Station:

Average Levels by Hour:
river_timestamp
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
5          NaN
6          NaN
7          NaN
8          NaN
9          NaN
10    0.168000
11    0.167729
12    0.166746
13    0.166860
14    0.167268
21    0.167026
22    0.167000
23    0.168700
Name: river_level, dtype: float64

Average Levels by Day of Week:
river_timestamp
5    0.167575
6    0.167321
Name: river_level, dtype: float64

Manchester Racecourse Station:

Average Levels by Hour:
river_timestamp
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
5          NaN
6          NaN
7          NaN
8          NaN
9          NaN
10    0.932000
11    0.929458
12    0.925949
13    0.925860
14    0.927625
21    0.939026
22    0.942000
23    0.941100
Name: river_level, dtype: float64

Average Levels by Day of Week:
river_timestamp
5    0.940709
6    0.928178
```

```
Name: river_level, dtype: float64

Bury Ground Station:

Average Levels by Hour:
river_timestamp
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
5          NaN
6          NaN
7          NaN
8          NaN
9          NaN
10    0.310000
11    0.309492
12    0.309000
13    0.309000
14    0.309250
21    0.316000
22    0.314000
23    0.314300
Name: river_level, dtype: float64

Average Levels by Day of Week:
river_timestamp
5    0.314767
6    0.309348
Name: river_level, dtype: float64
```

In [63]:

```python
import pandas as pd
import numpy as np
from supabase import create_client
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Supabase Connection Details
SUPABASE_URL = "https://thoqlquxaemyyhmpiwzt.supabase.co"
SUPABASE_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ

# Create Supabase client
supabase = create_client(SUPABASE_URL, SUPABASE_KEY)

# Fetch river data
print("Fetching data from Supabase...")
response = supabase.table('river_data').select('*').execute()

# Convert to DataFrame
df = pd.DataFrame(response.data)
df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

# Prepare features for forecasting
def prepare_forecast_features(station_data):
    station_data = station_data.sort_values('river_timestamp')
```

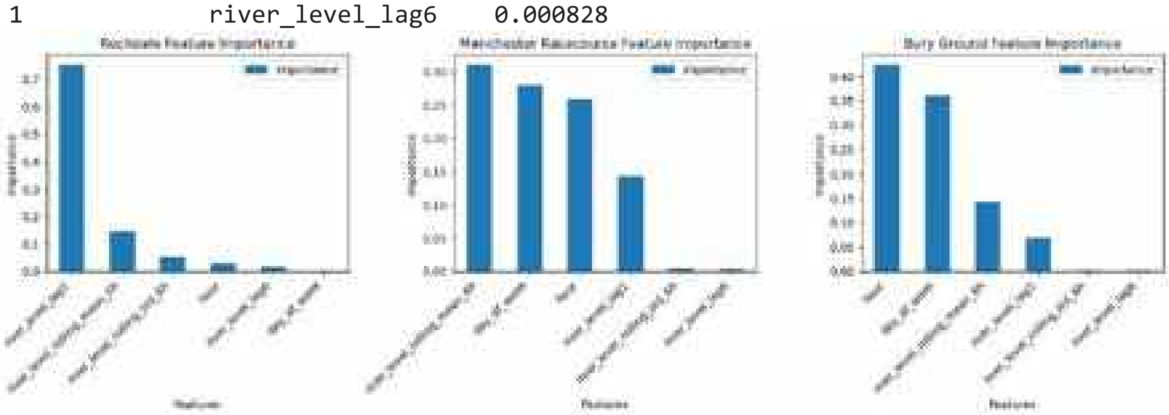
```
    # Create lag features
    station_data['river_level_lag1'] = station_data['river_level'].shift(1)
    station_data['river_level_lag6'] = station_data['river_level'].shift(6)

    # Rolling window features
    station_data['river_level_rolling_mean_6h'] = station_data['river_level'].rc
    station_data['river_level_rolling_std_6h'] = station_data['river_level'].rol

    # Time-based features
    station_data['hour'] = station_data['river_timestamp'].dt.hour
    station_data['day_of_week'] = station_data['river_timestamp'].dt.dayofweek

    # Target variable: next time step's river level
    station_data['target'] = station_data['river_level'].shift(-1)

    # Remove rows with NaN
    station_data = station_data.dropna()

    return station_data

# Train and evaluate model for each station
def train_forecast_model(station_data):
    # Prepare features
    features = prepare_forecast_features(station_data)

    # Select features and target
    feature_columns = [
        'river_level_lag1', 'river_level_lag6',
        'river_level_rolling_mean_6h', 'river_level_rolling_std_6h',
        'hour', 'day_of_week'
    ]

    X = features[feature_columns]
    y = features['target']

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train Random Forest model
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluate model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Feature importance
    feature_importance = pd.DataFrame({
        'feature': feature_columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)
```

```
    return {
        'model': model,
        'scaler': scaler,
        'mse': mse,
        'r2': r2,
        'feature_importance': feature_importance
    }

# Train models for each station
print("\nTraining Forecast Models:")
models = {}
for station in df['location_name'].unique():
    print(f"\n--- {station} Station ---")
    station_data = df[df['location_name'] == station].copy()

    # Train model
    model_results = train_forecast_model(station_data)
    models[station] = model_results

    # Display results
    print("Mean Squared Error:", model_results['mse'])
    print("R-squared Score:", model_results['r2'])

    print("\nFeature Importance:")
    print(model_results['feature_importance'])

# Visualization of Feature Importance
plt.figure(figsize=(15, 5))
for i, (station, model_data) in enumerate(models.items(), 1):
    plt.subplot(1, 3, i)
    model_data['feature_importance'].plot(x='feature', y='importance', kind='bar
    plt.title(f'{station} Feature Importance')
    plt.xlabel('Features')
    plt.ylabel('Importance')
    plt.xticks(rotation=45, ha='right')

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

```
Fetching data from Supabase...
```

```
2025-02-14 18:44:22,146 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A "HTTP/2 200 OK"
```

```
Training Forecast Models:

--- Rochdale Station ---
Mean Squared Error: 2.3543770152372118e-07
R-squared Score: 0.7452008378043906

Feature Importance:
                         feature  importance
0               river_level_lag1    0.752367
2  river_level_rolling_mean_6h    0.148547
3   river_level_rolling_std_6h    0.051199
4                           hour    0.029169
1               river_level_lag6    0.017528
5                    day_of_week    0.001191

--- Manchester Racecourse Station ---
Mean Squared Error: 1.4869989065596056e-06
R-squared Score: 0.9628804169800932

Feature Importance:
                         feature  importance
2  river_level_rolling_mean_6h    0.311013
5                    day_of_week    0.280234
4                           hour    0.258381
0               river_level_lag1    0.142983
3   river_level_rolling_std_6h    0.004417
1               river_level_lag6    0.002973

--- Bury Ground Station ---
Mean Squared Error: 9.764038531837617e-09
R-squared Score: 0.9988696674857902

Feature Importance:
                         feature  importance
4                           hour    0.424040
5                    day_of_week    0.360829
2  river_level_rolling_mean_6h    0.143287
0               river_level_lag1    0.069986
3   river_level_rolling_std_6h    0.001029
1               river_level_lag6    0.000828
```



In [65]:

```python
# In Jupyter Notebook

import pandas as pd
import numpy as np
from supabase import create_client
import pytz
from datetime import datetime, timedelta
```

```
# Supabase Credentials (Replace with your actual credentials)
supabase_url = "https://thoqlquxaemyyhmpiwzt.supabase.co"
supabase_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ

# Create Supabase client
supabase = create_client(supabase_url, supabase_key)

# Function to fetch river data
def fetch_river_data(days_back=90):
    """Fetch historical river monitoring data"""
    end_date = datetime.now(pytz.UTC)
    start_date = end_date - timedelta(days=days_back)

    try:
        response = supabase.table('river_data')\
            .select('*')\
            .gte('river_timestamp', start_date.isoformat())\
            .lte('river_timestamp', end_date.isoformat())\
            .order('river_timestamp', desc=True)\
            .execute()

        # Convert to DataFrame
        df = pd.DataFrame(response.data)
        df['river_timestamp'] = pd.to_datetime(df['river_timestamp'], utc=True)

        return df
    except Exception as e:
        print(f"Error fetching data: {e}")
        return None

# Fetch the data
river_data = fetch_river_data()

# Basic data exploration
if river_data is not None:
    print("Data Overview:")
    print("-" * 50)

    # Total records
    print(f"Total Records: {len(river_data)}")

    # Unique stations
    print("\nStations:")
    print(river_data['location_name'].unique())

    # Date range
    print("\nDate Range:")
    print(f"Start: {river_data['river_timestamp'].min()}")
    print(f"End: {river_data['river_timestamp'].max()}")

    # Basic statistics
    print("\nBasic Statistics:")
    print(river_data.groupby('location_name')[['river_level', 'rainfall']].agg([

    # Check for missing values
    print("\nMissing Values:")
    print(river_data.isnull().sum())
else:
    print("No data could be retrieved.")
```

```
2025-02-14 18:54:41,988 - INFO - HTTP Request: GET https://thoqlquxaemyyhmpiwzt.s
upabase.co/rest/v1/river_data?select=%2A&river_timestamp=gte.2024-11-16T18%3A54%3
A41.225488%2B00%3A00&river_timestamp=lte.2025-02-14T18%3A54%3A41.225488%2B00%3A00
&order=river_timestamp.desc "HTTP/2 200 OK"
Data Overview:
--------------------------------------------------
Total Records: 1000

Stations:
['Bury Ground' 'Manchester Racecourse' 'Rochdale']

Date Range:
Start: 2025-02-12 08:45:00+00:00
End: 2025-02-13 13:00:00+00:00

Basic Statistics:
                      river_level                 rainfall
                             mean    min    max       mean  min  max
location_name
Bury Ground              0.310917  0.308  0.314  0.000000  0.0  0.0
Manchester Racecourse    0.927919  0.915  0.936  0.000000  0.0  0.0
Rochdale                 0.173372  0.166  0.180  0.008798  0.0  0.2

Missing Values:
id                     0
river_level            0
river_timestamp        0
rainfall               0
rainfall_timestamp     0
location_name          0
river_station_id       0
rainfall_station_id    0
created_at             0
dtype: int64
```

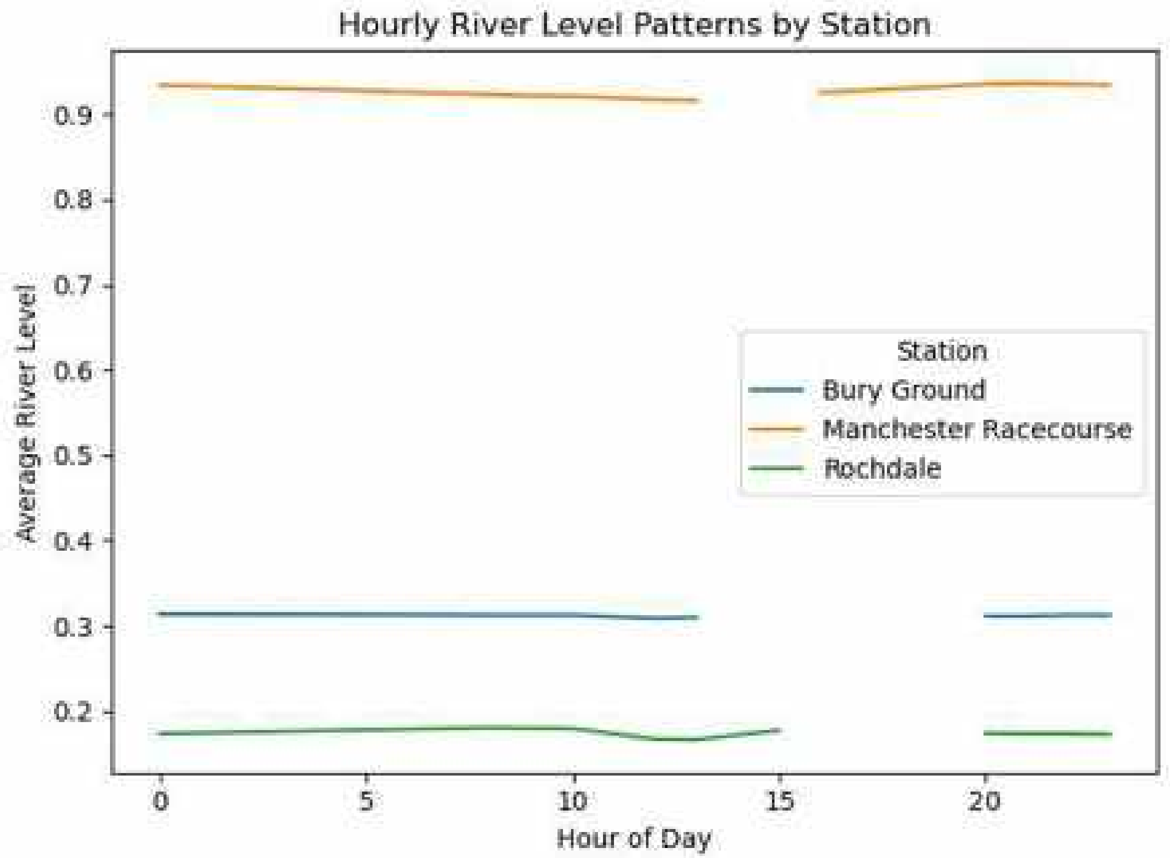
In [66]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Preprocessing and Feature Engineering
def preprocess_data(df):
    # Convert timestamp to datetime if not already
    df['river_timestamp'] = pd.to_datetime(df['river_timestamp'])

    # Extract time-based features
    df['hour'] = df['river_timestamp'].dt.hour
    df['day_of_week'] = df['river_timestamp'].dt.dayofweek

    return df

# Temporal Pattern Analysis
def analyze_temporal_patterns(df):
    # Group by station and hour
    hourly_patterns = df.groupby(['location_name', 'hour'])['river_level'].mean(

    # Visualize hourly patterns
    plt.figure(figsize=(15, 5))
    hourly_patterns.T.plot(kind='line')
```

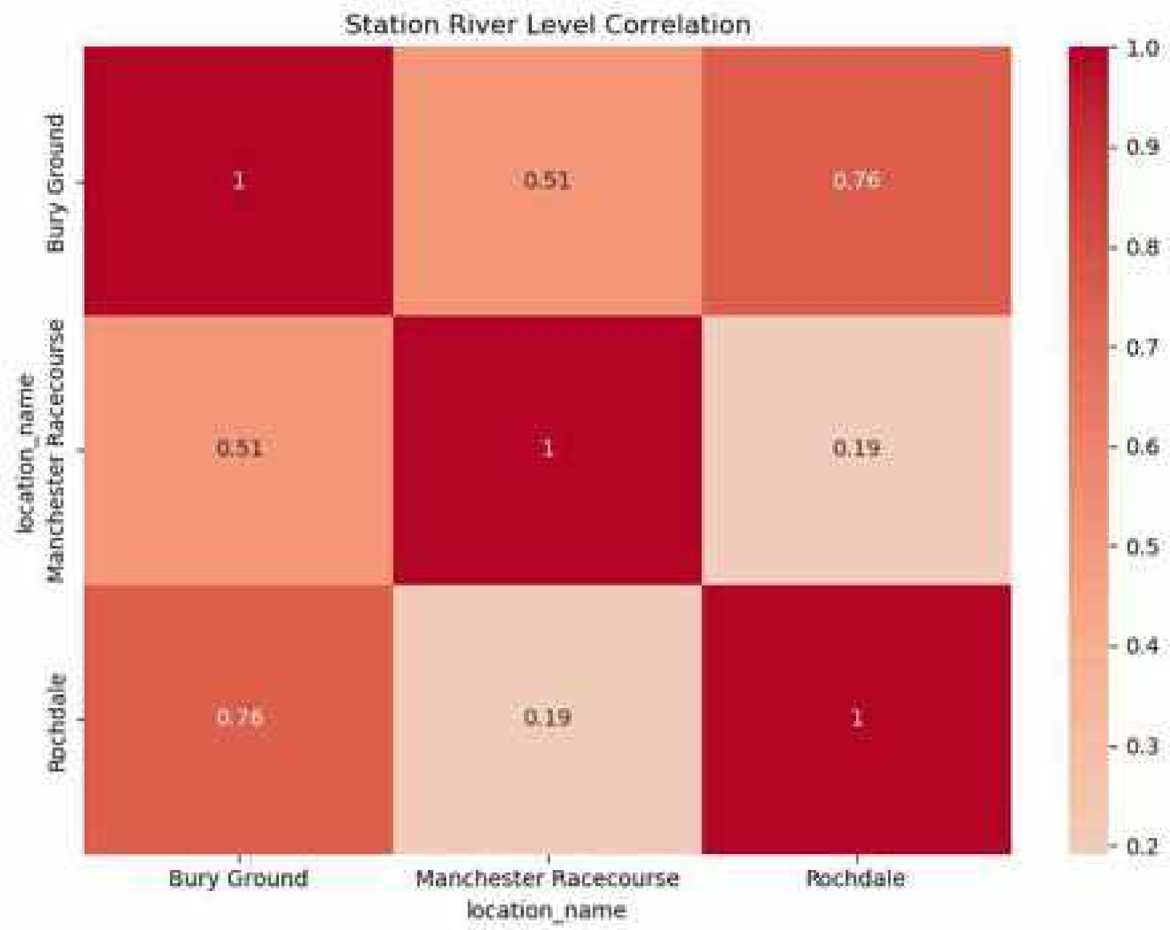
```
    plt.title('Hourly River Level Patterns by Station')
    plt.xlabel('Hour of Day')
    plt.ylabel('Average River Level')
    plt.legend(title='Station')
    plt.tight_layout()
    plt.show()

    return hourly_patterns

# Correlation Analysis
def station_correlation_analysis(df):
    # Pivot data for correlation
    pivot_data = df.pivot_table(
        index='river_timestamp',
        columns='location_name',
        values='river_level'
    )

    # Compute correlation matrix
    correlation_matrix = pivot_data.corr()

    # Visualize correlation
    plt.figure(figsize=(8, 6))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Station River Level Correlation')
    plt.tight_layout()
    plt.show()

    return correlation_matrix

# Statistical Significance Test
def correlation_significance_test(df):
    stations = df['location_name'].unique()
    correlation_tests = {}

    for i in range(len(stations)):
        for j in range(i+1, len(stations)):
            station1 = stations[i]
            station2 = stations[j]

            # Extract river levels for both stations
            data1 = df[df['location_name'] == station1]['river_level']
            data2 = df[df['location_name'] == station2]['river_level']

            # Pearson correlation and p-value
            correlation, p_value = stats.pearsonr(data1, data2)

            correlation_tests[f'{station1} vs {station2}'] = {
                'correlation': correlation,
                'p_value': p_value
            }

    return correlation_tests

# Main Analysis Workflow
def comprehensive_pattern_analysis(df):
    # Preprocess data
    processed_df = preprocess_data(df)

    print("1. Temporal Pattern Analysis:")
```

```
    hourly_patterns = analyze_temporal_patterns(processed_df)

    print("\n2. Station Correlation Analysis:")
    correlation_matrix = station_correlation_analysis(processed_df)
    print(correlation_matrix)

    print("\n3. Correlation Significance Tests:")
    significance_tests = correlation_significance_test(processed_df)
    for test, results in significance_tests.items():
        print(f"{test}:")
        print(f"  Correlation: {results['correlation']:.4f}")
        print(f"  P-value: {results['p_value']:.4f}")

# Execute the analysis
comprehensive_pattern_analysis(river_data)
```

```
1. Temporal Pattern Analysis:
<Figure size 1500x500 with 0 Axes>
```




```
2. Station Correlation Analysis:
```




```
location_name          Bury Ground  Manchester Racecourse  Rochdale
location_name
Bury Ground               1.000000               0.510961  0.764905
Manchester Racecourse     0.510961               1.000000  0.191344
Rochdale                  0.764905               0.191344  1.000000
```

```
3. Correlation Significance Tests:
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11076\3254882637.py in ?()
     96         print(f"  Correlation: {results['correlation']:.4f}")
     97         print(f"  P-value: {results['p_value']:.4f}")
     98 
     99 # Execute the analysis
--> 100 comprehensive_pattern_analysis(river_data)

~\AppData\Local\Temp\ipykernel_11076\3254882637.py in ?(df)
     89     correlation_matrix = station_correlation_analysis(processed_df)
     90     print(correlation_matrix)
     91 
     92     print("\n3. Correlation Significance Tests:")
---> 93     significance_tests = correlation_significance_test(processed_df)
     94     for test, results in significance_tests.items():
     95         print(f"{test}:")
     96         print(f"  Correlation: {results['correlation']:.4f}")

~\AppData\Local\Temp\ipykernel_11076\3254882637.py in ?(df)
     67             data1 = df[df['location_name'] == station1]['river_level']
     68             data2 = df[df['location_name'] == station2]['river_level']
     69 
     70             # Pearson correlation and p-value
---> 71             correlation, p_value = stats.pearsonr(data1, data2)
     72 
     73             correlation_tests[f'{station1} vs {station2}'] = {
     74                 'correlation': correlation,

~\AppData\Roaming\Python\Python312\site-packages\scipy\stats\_stats_py.py in ?(x, y, alternative, method, axis)
   4544     axis = axis_int
   4545 
   4546     n = x.shape[axis]
   4547     if n != y.shape[axis]:
-> 4548         raise ValueError('`x` and `y` must have the same length along `axis`.')
   4549 
   4550     if n < 2:
   4551         raise ValueError('`x` and `y` must have length at least 2.')

ValueError: `x` and `y` must have the same length along `axis`.
```

Correlation Matrix

In [67]:

```python
def correlation_significance_test(df):
    stations = df['location_name'].unique()
    correlation_tests = {}

    for i in range(len(stations)):
        for j in range(i+1, len(stations)):
            station1 = stations[i]
            station2 = stations[j]

            # Group by timestamp to ensure aligned data
            merged_data = df.pivot_table(
                index='river_timestamp',
                columns='location_name',
                values='river_level'
```

```
            )

            # Remove rows with missing data
            merged_data.dropna(inplace=True)

            # Perform correlation test
            correlation, p_value = stats.pearsonr(
                merged_data[station1],
                merged_data[station2]
            )

            correlation_tests[f'{station1} vs {station2}'] = {
                'correlation': correlation,
                'p_value': p_value
            }

    return correlation_tests

# Run the updated test
correlation_significance_tests = correlation_significance_test(river_data)
for test, results in correlation_significance_tests.items():
    print(f"{test}:")
    print(f"  Correlation: {results['correlation']:.4f}")
    print(f"  P-value: {results['p_value']:.4f}")
```

```
Bury Ground vs Manchester Racecourse:
  Correlation: 0.5634
  P-value: 0.0097
Bury Ground vs Rochdale:
  Correlation: 0.7649
  P-value: 0.0001
Manchester Racecourse vs Rochdale:
  Correlation: 0.1913
  P-value: 0.4190
```

ADVANCED TIME-LAG ANALYSIS

In [71]:

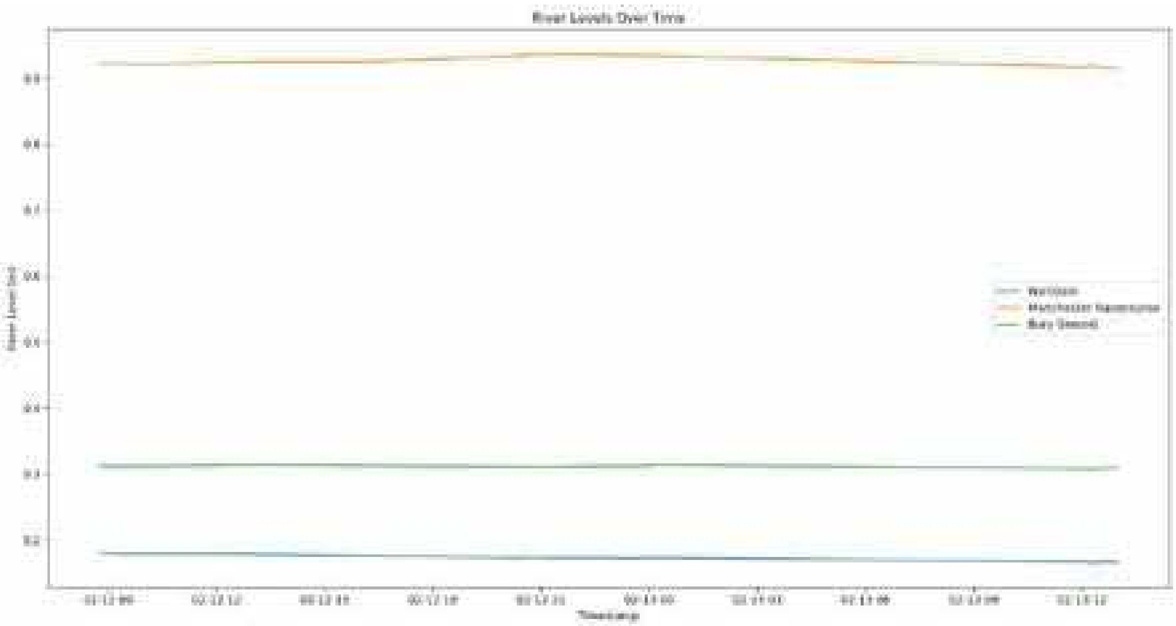
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

def time_lag_analysis(df):
    # Preprocess: Remove duplicate timestamps
    df_unique = df.drop_duplicates(subset=['river_timestamp', 'location_name'])

    # Stations in flow order: Rochdale → Manchester Racecourse → Bury Ground
    stations = ['Rochdale', 'Manchester Racecourse', 'Bury Ground']

    # Detailed data exploration
    print("Data Exploration:")
    for station in stations:
        station_data = df_unique[df_unique['location_name'] == station]
        print(f"\n{station} Station:")
        print(f"Total records: {len(station_data)}")
        print(f"Timestamp range: {station_data['river_timestamp'].min()} to {sta
        print(f"River level - Min: {station_data['river_level'].min():.4f}, Max:
        print(f"River level - Mean: {station_data['river_level'].mean():.4f}, St

    # Pivot data to ensure aligned timestamps
```

```
    pivot_data = df_unique.pivot_table(
        index='river_timestamp',
        columns='location_name',
        values='river_level'
    )

    # Remove rows with missing data
    pivot_data.dropna(inplace=True)

    # Time lag investigation
    time_lag_results = {}

    # Compute correlations manually
    for i in range(len(stations) - 1):
        station1 = stations[i]
        station2 = stations[i+1]

        # Compute correlations with different lags
        max_lag_hours = 24
        correlations = []
        lags = range(-max_lag_hours, max_lag_hours + 1)

        for lag in lags:
            # Shift data
            shifted_data = pivot_data[station2].shift(lag)

            # Compute correlation, handling potential issues
            try:
                correlation = pivot_data[station1].corr(shifted_data)
                correlations.append(correlation)
            except Exception as e:
                print(f"Correlation error for {station1} and {station2}: {e}")
                correlations.append(np.nan)

        # Find optimal lag
        valid_correlations = [c for c in correlations if not np.isnan(c)]
        if valid_correlations:
            optimal_lag = lags[correlations.index(max(valid_correlations))]
            max_correlation = max(valid_correlations)
        else:
            optimal_lag = 0
            max_correlation = 0

        time_lag_results[f'{station1} → {station2}'] = {
            'optimal_lag_hours': optimal_lag,
            'max_correlation': max_correlation
        }

    # Visualization
    plt.figure(figsize=(15, 8))

    # Time series of river levels
    for station in stations:
        station_data = df_unique[df_unique['location_name'] == station]
        plt.plot(station_data['river_timestamp'], station_data['river_level'], l

    plt.title('River Levels Over Time')
    plt.xlabel('Timestamp')
    plt.ylabel('River Level (m)')
    plt.legend()
```

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

    # Print time lag results
    print("\nTime Lag Analysis Results:")
    for route, results in time_lag_results.items():
        print(f"\n{route}:")
        print(f"  Optimal Lag: {results['optimal_lag_hours']} hours")
        print(f"  Correlation at Lag: {results['max_correlation']:.4f}")

    return time_lag_results

# Run the analysis
time_lag_results = time_lag_analysis(river_data)
```

```
Data Exploration:

Rochdale Station:
Total records: 22
Timestamp range: 2025-02-12 08:45:00+00:00 to 2025-02-13 13:00:00+00:00
River level - Min: 0.1660, Max: 0.1800
River level - Mean: 0.1737, Std: 0.0049

Manchester Racecourse Station:
Total records: 22
Timestamp range: 2025-02-12 08:45:00+00:00 to 2025-02-13 13:00:00+00:00
River level - Min: 0.9150, Max: 0.9360
River level - Mean: 0.9262, Std: 0.0078

Bury Ground Station:
Total records: 21
Timestamp range: 2025-02-12 08:45:00+00:00 to 2025-02-13 13:00:00+00:00
River level - Min: 0.3080, Max: 0.3140
River level - Mean: 0.3111, Std: 0.0018
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:288
9: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:274
8: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:274
8: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:289
7: RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
C:\Users\Administrator\anaconda3\Lib\site-packages\numpy\lib\function_base.py:289
8: RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]
```




```
Time Lag Analysis Results:

Rochdale → Manchester Racecourse:
  Optimal Lag: 18 hours
  Correlation at Lag: 1.0000

Manchester Racecourse → Bury Ground:
  Optimal Lag: 9 hours
  Correlation at Lag: 0.9923
```

Advanced Alert System Enhancement

In [72]:
```
# Test Alert Configuration
from alert_config import AlertConfiguration

# Create instance
alert_config = AlertConfiguration()

# Test getting configuration
rochdale_config = alert_config.get_alert_configuration('Rochdale')
print("Rochdale Configuration:")
print(rochdale_config)

# Test adding a contact
new_contact = {
    'name': 'Emergency Response',
    'email': 'emergency@rochdale.gov.uk',
    'phone': '+441234567890'
}
alert_config.add_custom_contact('Rochdale', new_contact)

# Test updating threshold
alert_config.update_threshold('Rochdale', 'warning_level', 0.175)

# Verify changes
updated_config = alert_config.get_alert_configuration('Rochdale')
print("\nUpdated Rochdale Configuration:")
print(updated_config)
```

```
Rochdale Configuration:
{'warning_level': 0.168, 'alert_level': 0.169, 'critical_level': 0.17, 'custom_co
ntacts': [], 'notification_channels': ['dashboard', 'email']}

Updated Rochdale Configuration:
{'warning_level': 0.175, 'alert_level': 0.169, 'critical_level': 0.17, 'custom_co
ntacts': [{'name': 'Emergency Response', 'email': 'emergency@rochdale.gov.uk', 'p
hone': '+441234567890'}], 'notification_channels': ['dashboard', 'email']}
```

In [73]:
```python
# Test Notification System
from notification_system import NotificationSystem
import streamlit as st

# Create instance
notification_system = NotificationSystem()

# Test sending notification
test_email = "emergency@rochdale.gov.uk"
subject = "TEST ALERT: High Water Level"
message = """
FLOOD ALERT: Rochdale Station
Current Level: 0.175m
Status: WARNING
Time: 2025-02-15 10:30:00
"""

# Send test notification
notification_system.send_email(test_email, subject, message)

# Check notification history
history = notification_system.get_notification_history()
print("\nNotification History:")
for notification in history:
    print(f"\nType: {notification['type']}")
    print(f"Recipient: {notification['recipient']}")
    print(f"Subject: {notification['subject']}")
    print(f"Time: {notification['timestamp']}")
    print(f"Status: {notification['status']}")
```

```
2025-02-15 02:31:21.566 Thread 'MainThread': missing ScriptRunContext! This warni
ng can be ignored when running in bare mode.
2025-02-15 02:31:21.583 Thread 'MainThread': missing ScriptRunContext! This warni
ng can be ignored when running in bare mode.
Notification History:

Type: email
Recipient: emergency@rochdale.gov.uk
Subject: TEST ALERT: High Water Level
Time: 2025-02-15 02:31:21.499920
Status: simulated
```

In [78]:
```python
# Test Alert History Tracking
from alert_history import AlertHistoryTracker

# Create tracker instance
alert_tracker = AlertHistoryTracker('test_alert_history.csv')

# Log some test alerts
test_alerts = [
    {
```

```python
        'station': 'Rochdale',
        'river_level': 0.175,
        'alert_type': 'WARNING',
        'notification_sent': True,
        'notification_type': 'email',
        'recipients': 'emergency@rochdale.gov.uk'
    },
    {
        'station': 'Manchester Racecourse',
        'river_level': 0.945,
        'alert_type': 'CRITICAL',
        'notification_sent': True,
        'notification_type': 'email,sms',
        'recipients': 'emergency@manchester.gov.uk'
    }
]

# Log each alert
for alert in test_alerts:
    alert_tracker.log_alert(**alert)

# Test retrieving recent alerts
recent_alerts = alert_tracker.get_recent_alerts(days=1)
print("Recent Alerts (Last 24 hours):")
print(recent_alerts[['station', 'alert_type', 'river_level', 'notification_type'

# Test getting station-specific alerts
rochdale_alerts = alert_tracker.get_station_alerts('Rochdale')
print("\nRochdale Station Alerts:")
print(rochdale_alerts[['timestamp', 'alert_type', 'river_level']])

# Test getting alert summary
summary = alert_tracker.get_alert_summary()
print("\nAlert Summary by Station:")
print(summary)
```

```
Recent Alerts (Last 24 hours):
                 station alert_type  river_level notification_type
0               Rochdale    WARNING        0.175             email
1  Manchester Racecourse   CRITICAL        0.945         email,sms
2               Rochdale    WARNING        0.175             email
3  Manchester Racecourse   CRITICAL        0.945         email,sms
4               Rochdale    WARNING        0.175             email
5  Manchester Racecourse   CRITICAL        0.945         email,sms
6               Rochdale    WARNING        0.175             email
7  Manchester Racecourse   CRITICAL        0.945         email,sms

Rochdale Station Alerts:
                   timestamp alert_type  river_level
0 2025-02-15 02:32:24.424818    WARNING        0.175
2 2025-02-15 02:33:19.366587    WARNING        0.175
4 2025-02-15 02:34:55.692167    WARNING        0.175
6 2025-02-15 02:35:02.108537    WARNING        0.175

Alert Summary by Station:
alert_type             CRITICAL  WARNING
station
Manchester Racecourse         4        0
Rochdale                      0        4
```

In [87]:
```
import os
print("Current working directory:", os.getcwd())
```

```
Current working directory: C:\Users\Administrator\NEWPROJECT
```

In [90]:
```
from notification_config import NotificationConfig

# Create notification configuration
notify_config = NotificationConfig()

# Test enabling/disabling channels
notify_config.enable_channel('sms')
notify_config.set_channel_provider('email', 'gmail')

# Print configuration
print(notify_config.channels)
```

```
{'email': {'enabled': True, 'provider': 'gmail', 'credentials': {}}, 'sms': {'ena
bled': True, 'provider': None, 'credentials': {}}, 'dashboard': {'enabled': Tru
e}}
```

In [92]:
```
from notification_config import NotificationConfig
from notification_sender import NotificationSender
from email_config import EMAIL_CONFIG  # Import credentials

# Create notification configuration
notify_config = NotificationConfig()

# Create notification sender
notification_sender = NotificationSender(notify_config)

# Set email credentials from config
notification_sender.set_email_credentials(
    EMAIL_CONFIG['sender_email'],
    EMAIL_CONFIG['app_password']
)

# Test email sending
test_result = notification_sender.send_email(
    'recipient@example.com',
    'Test Flood Monitoring Notification',
    'This is a test notification from the Flood Monitoring System.'
)

print("Email sending result:", test_result)
```

```
Email sent successfully to recipient@example.com
Email sending result: True
```

In [98]:
```
from notification_config import NotificationConfig
from notification_sender import NotificationSender

# Create notification configuration
notify_config = NotificationConfig()
notify_config.enable_channel('sms')  # Enable SMS

# Create notification sender
notification_sender = NotificationSender(notify_config)

# Print out the current config to verify
print("Notification Channels:", notify_config.channels)
```

```
# Verify method exists
print("Methods in NotificationSender:", dir(notification_sender))

# Try setting SMS credentials
try:
    notification_sender.set_sms_credentials(
        'your_account_sid',
        'your_auth_token',
        'your_twilio_phone_number'
    )
    print("SMS credentials set successfully")
except Exception as e:
    print("Error setting SMS credentials:", str(e))
```

```
Notification Channels: {'email': {'enabled': True, 'provider': None, 'credential
s': {}}, 'sms': {'enabled': True, 'provider': None, 'credentials': {}}, 'dashboar
d': {'enabled': True}}
Methods in NotificationSender: ['__class__', '__delattr__', '__dict__', '__dir_
_', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getstate_
_', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '_
_module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__se
tattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'config',
'email_config', 'send_email', 'set_email_credentials']
Error setting SMS credentials: 'NotificationSender' object has no attribute 'set_
sms_credentials'
```

In [3]:

```
# Add project directory to Python path
import sys
import os

# Get the project directory
project_dir = r'C:\Users\Administrator\NEWPROJECT'
print("Project Directory:", project_dir)

# Add project directory to Python path
if project_dir not in sys.path:
    sys.path.append(project_dir)

# Verify Python can find the modules
print("\nPython Path:")
for path in sys.path:
    print(path)

# Now try importing
from notification_config import NotificationConfig
from notification_sender import NotificationSender

# Create notification configuration
notify_config = NotificationConfig()
notify_config.enable_channel('sms')

# Create notification sender
notification_sender = NotificationSender(notify_config)

# Set SMS credentials (with dummy data)
notification_sender.set_sms_credentials(
    'test_account_sid',
    'test_auth_token',
    'test_twilio_number'
```

```
)

# Print channels to verify
print("\nNotification Channels:")
print(notify_config.channels)

# Print SMS configuration
print("\nSMS Configuration:")
print(notification_sender.sms_config)
```

```
Project Directory: C:\Users\Administrator\NEWPROJECT

Python Path:
C:\Users\Administrator\anaconda3\python312.zip
C:\Users\Administrator\anaconda3\DLLs
C:\Users\Administrator\anaconda3\Lib
C:\Users\Administrator\anaconda3

C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages
C:\Users\Administrator\anaconda3\Lib\site-packages
C:\Users\Administrator\anaconda3\Lib\site-packages\win32
C:\Users\Administrator\anaconda3\Lib\site-packages\win32\lib
C:\Users\Administrator\anaconda3\Lib\site-packages\Pythonwin
C:\Users\Administrator
C:\Users\Administrator\NEWPROJECT

Notification Channels:
{'email': {'enabled': True, 'provider': None, 'credentials': {}}, 'sms': {'enable
d': True, 'provider': None, 'credentials': {}}, 'dashboard': {'enabled': True}}

SMS Configuration:
{'account_sid': 'test_account_sid', 'auth_token': 'test_auth_token', 'twilio_numb
er': 'test_twilio_number'}
```

In [4]:

```
from notification_manager import NotificationManager

# Initialize Notification Manager
notification_manager = NotificationManager()

# Add Emergency Contact
emergency_contact = {
    'name': 'Emergency Response Team',
    'email': 'emergency@example.com',
    'phone': '+1234567890',
    'organization': 'Flood Monitoring Center'
}

# Add recipient
result = notification_manager.add_recipient('emergency_contacts', emergency_cont
print("Recipient Added:", result)

# Log a test notification
notification_manager.log_notification('email', 'emergency@example.com', True)

# Retrieve recent logs
recent_logs = notification_manager.get_notification_logs()
print("\nRecent Notification Logs:")
for log in recent_logs:
    print(log)
```

```
Recipient Added: True

Recent Notification Logs:
{'timestamp': '2025-02-15T12:56:23.089982', 'type': 'email', 'recipient': 'emerge
ncy@example.com', 'status': 'Success'}
```

In [11]:
```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from statsmodels.tsa.seasonal import seasonal_decompose

class FloodAnomalyDetector:
    def __init__(self, contamination=0.1):
        self.scaler = StandardScaler()
        self.isolation_forest = IsolationForest(contamination=contamination, ran

    def detect_anomalies(self, df, column='water_level'):
        """
        Detect anomalies using multiple methods
        """
        # 1. Statistical thresholds
        mean = df[column].mean()
        std = df[column].std()
        z_scores = np.abs((df[column] - mean) / std)
        statistical_anomalies = z_scores > 3

        # 2. Isolation Forest
        scaled_data = self.scaler.fit_transform(df[[column]])
        isolation_forest_anomalies = self.isolation_forest.fit_predict(scaled_da

        # 3. Seasonal Decomposition
        try:
            decomposition = seasonal_decompose(df[column], period=96)  # 24 hour
            residuals = decomposition.resid
            residual_anomalies = np.abs(residuals) > 2 * np.std(residuals)
        except:
            residual_anomalies = np.zeros_like(statistical_anomalies)

        # Combine detections
        final_anomalies = (statistical_anomalies & isolation_forest_anomalies) |

        return {
            'anomalies': final_anomalies,
            'confidence': z_scores,
            'seasonal_residuals': residuals if 'residuals' in locals() else None
        }

    def calculate_risk_level(self, current_level, historical_data):
        """
        Calculate flood risk level based on historical context
        """
        historical_peaks = historical_data['water_level'].quantile([0.5, 0.75, 0

        if current_level > historical_peaks[0.99]:
            return 5  # Severe risk
        elif current_level > historical_peaks[0.95]:
            return 4  # High risk
        elif current_level > historical_peaks[0.90]:
            return 3  # Moderate risk
        elif current_level > historical_peaks[0.75]:
```

```
            return 2  # Low risk
        elif current_level > historical_peaks[0.50]:
            return 1  # Very low risk
        else:
            return 0  # Normal conditions
```

implementing temperature features

In [12]:
```
import pandas as pd
import os

def load_temperature_data():
    # Path to temperature data
    data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_tempera

    # Read temperature data
    temp_df = pd.read_csv(data_path)

    # Print first few rows to verify data
    print("Temperature Data Sample:")
    print(temp_df.head())

    return temp_df

# Load and verify data
temp_data = load_temperature_data()
```

```
Temperature Data Sample:
    Month                Station Grid_ID  Temperature_C      Grid     Period
0   April        BURY MANCHESTER   AX-70            8.1  12km BNG  1991-2020
1   April  MANCHESTER RACECOURSE   AX-71            9.4  12km BNG  1991-2020
2   April               ROCHDALE   AY-70            7.9  12km BNG  1991-2020
3  August        BURY MANCHESTER   AX-70           15.2  12km BNG  1991-2020
4  August  MANCHESTER RACECOURSE   AX-71           16.5  12km BNG  1991-2020
```

In [13]:
```
def prepare_temperature_features():
    # Path to temperature data
    data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_tempera

    # Read temperature data
    temp_df = pd.read_csv(data_path)

    # Create a pivot table for easier lookup
    temp_pivot = temp_df.pivot_table(
        index='Station',
        columns='Month',
        values='Temperature_C'
    )

    print("\nPivoted Temperature Data:")
    print(temp_pivot)

    return temp_pivot

# Run and verify pivoted data
temp_pivot = prepare_temperature_features()
```

```
Pivoted Temperature Data:
Month                     April  August  December  February  January  July  June  \
Station
BURY MANCHESTER             8.1    15.2       4.1       4.1      3.8  15.5  13.6
MANCHESTER RACECOURSE       9.4    16.5       5.3       5.4      5.0  16.8  15.0
ROCHDALE                    7.9    15.1       4.0       3.9      3.6  15.3  13.4

Month                     March   May  November  October  September
Station
BURY MANCHESTER             5.7  11.0       6.5      9.7       12.9
MANCHESTER RACECOURSE       7.0  12.4       7.6     11.0       14.2
ROCHDALE                    5.4  10.7       6.2      9.6       12.8
```

In [15]:

```
# Check station names in temperature data
print("Temperature data station names:")
print(temp_pivot.index.tolist())

# Check station names in river data
print("\nRiver data station names:")
print(river_data['location_name'].unique())
```

```
Temperature data station names:
['BURY MANCHESTER', 'MANCHESTER RACECOURSE', 'ROCHDALE']

River data station names:
['Bury Ground' 'Manchester Racecourse' 'Rochdale']
```

In [16]:

```
def add_temperature_features(river_data, temp_pivot):
    """
    Add temperature features to river level data with station name mapping
    """
    # Create station name mapping
    station_mapping = {
        'Bury Ground': 'BURY MANCHESTER',
        'Manchester Racecourse': 'MANCHESTER RACECOURSE',
        'Rochdale': 'ROCHDALE'
    }

    # Create a copy of river data
    df = river_data.copy()

    # Add month as feature for temperature lookup
    df['month'] = pd.to_datetime(df['river_timestamp']).dt.strftime('%B')

    # Add temperature using the mapping
    df['temperature'] = df.apply(
        lambda row: temp_pivot.loc[station_mapping[row['location_name']], row['m
        axis=1
    )

    # Add rolling mean temperature (3-day window)
    df['temp_rolling_mean'] = df.groupby('location_name')['temperature'].transfo
        lambda x: x.rolling(window=72, min_periods=1).mean()
    )

    # Add temperature change from previous reading
    df['temp_change'] = df.groupby('location_name')['temperature'].diff()

    # Print sample of new features
    print("\nSample of data with temperature features:")
```

```
    print(df[['location_name', 'river_timestamp', 'river_level', 'month',
              'temperature', 'temp_rolling_mean', 'temp_change']].head())

    return df

# Test with river data
enhanced_data = add_temperature_features(river_data, temp_pivot)
```

```
Sample of data with temperature features:
           location_name           river_timestamp  river_level    month  \
0            Bury Ground  2025-01-30 11:15:00+00:00        0.385  January
1  Manchester Racecourse  2025-01-30 11:15:00+00:00        1.064  January
2               Rochdale  2025-01-30 11:15:00+00:00        0.235  January
3            Bury Ground  2025-01-30 11:30:00+00:00        0.386  January
4  Manchester Racecourse  2025-01-30 11:30:00+00:00        1.064  January

   temperature  temp_rolling_mean  temp_change
0          3.8                3.8          NaN
1          5.0                5.0          NaN
2          3.6                3.6          NaN
3          3.8                3.8          0.0
4          5.0                5.0          0.0
```

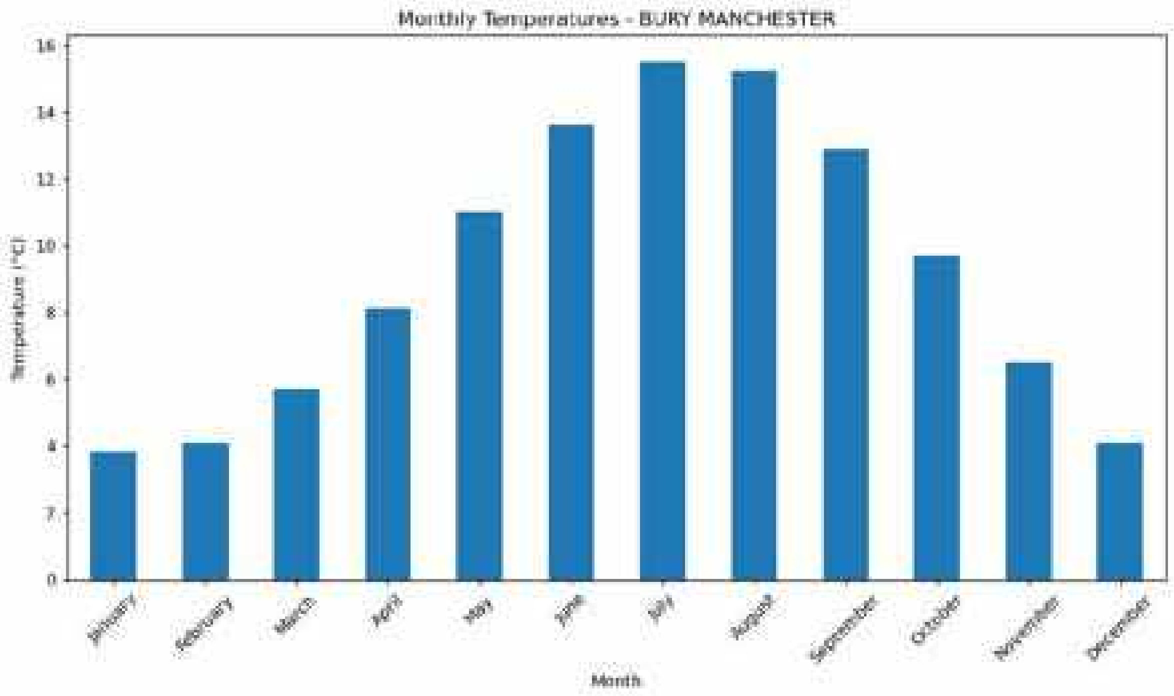
In [17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)

# Print the first few rows to verify data
print(temp_df.head())
```

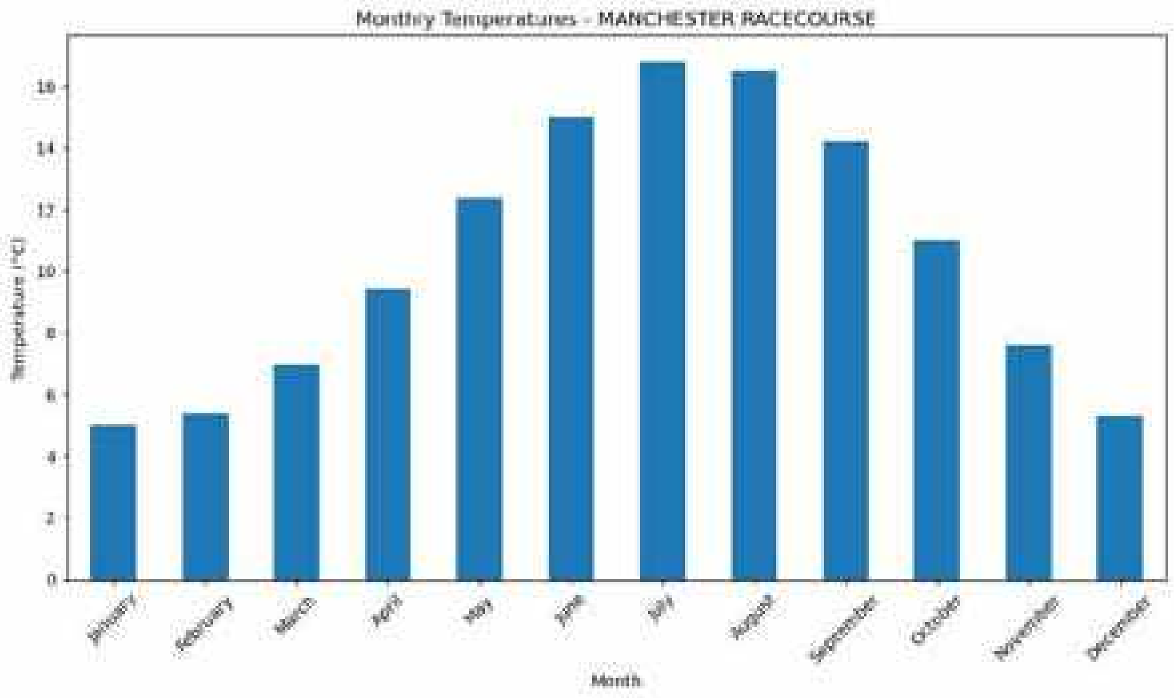
```
    Month                Station Grid_ID  Temperature_C       Grid     Period
0   April        BURY MANCHESTER   AX-70            8.1  12km BNG  1991-2020
1   April  MANCHESTER RACECOURSE   AX-71            9.4  12km BNG  1991-2020
2   April               ROCHDALE   AY-70            7.9  12km BNG  1991-2020
3  August        BURY MANCHESTER   AX-70           15.2  12km BNG  1991-2020
4  August  MANCHESTER RACECOURSE   AX-71           16.5  12km BNG  1991-2020
```

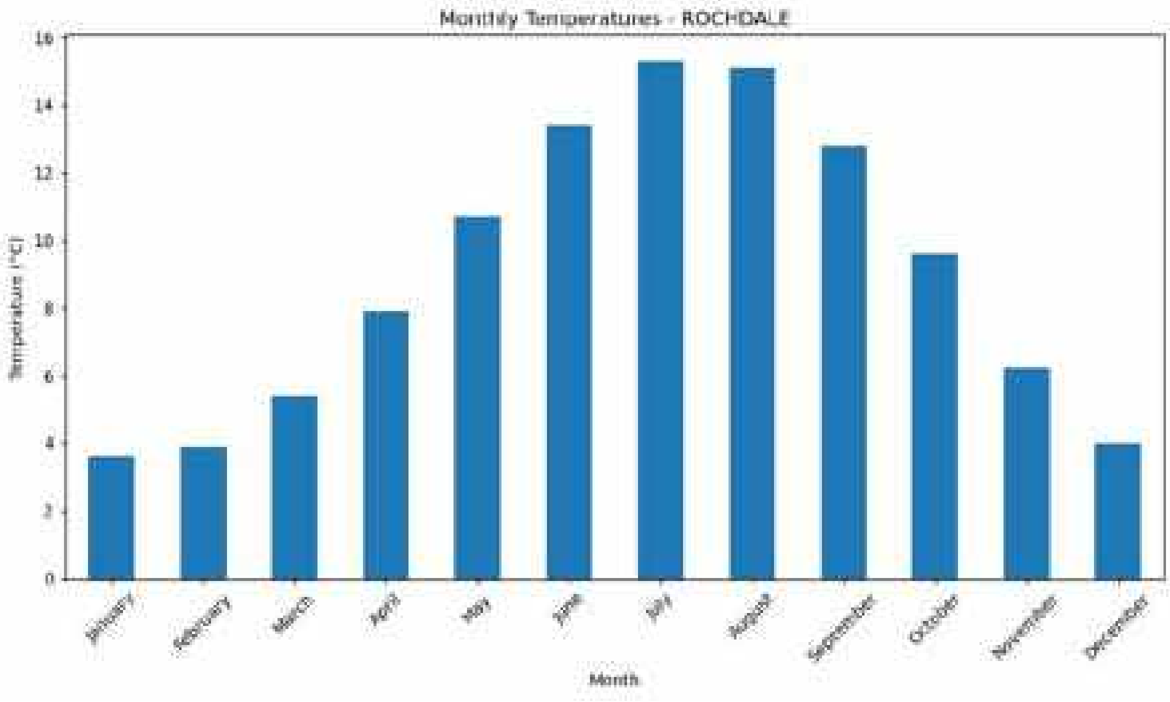
In [19]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)

def analyze_temperature_seasonality(temp_df):
    """
    Analyze temperature seasonality without synthetic data
    """
    # Create a mapping of month order
    month_order = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4,
        'May': 5, 'June': 6, 'July': 7, 'August': 8,
        'September': 9, 'October': 10, 'November': 11, 'December': 12
    }
```

```
    # Stations to analyze
    stations = temp_df['Station'].unique()
    seasonal_analysis = {}

    for station in stations:
        # Filter data for specific station
        station_data = temp_df[temp_df['Station'] == station].copy()

        # Sort by month order
        station_data['month_num'] = station_data['Month'].map(month_order)
        station_data = station_data.sort_values('month_num')

        # Calculate key seasonal statistics
        seasonal_stats = {
            'monthly_temps': station_data.set_index('Month')['Temperature_C'],
            'temp_range': station_data['Temperature_C'].max() - station_data['Te
            'coldest_month': station_data.loc[station_data['Temperature_C'].idxm
            'warmest_month': station_data.loc[station_data['Temperature_C'].idxm
            'mean_temp': station_data['Temperature_C'].mean(),
            'temp_std': station_data['Temperature_C'].std()
        }

        # Plot monthly temperatures
        plt.figure(figsize=(10,6))
        seasonal_stats['monthly_temps'].plot(kind='bar')
        plt.title(f'Monthly Temperatures - {station}')
        plt.xlabel('Month')
        plt.ylabel('Temperature (°C)')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

        seasonal_analysis[station] = seasonal_stats

    return seasonal_analysis

# Perform analysis
seasonal_analysis = analyze_temperature_seasonality(temp_df)

# Print seasonal analysis results
for station, stats in seasonal_analysis.items():
    print(f"\nSeasonal Analysis for {station}:")
    print(f"Monthly Temperatures:\n{stats['monthly_temps']}")
    print(f"Temperature Range: {stats['temp_range']:.2f}°C")
    print(f"Coldest Month: {stats['coldest_month']}")
    print(f"Warmest Month: {stats['warmest_month']}")
    print(f"Mean Temperature: {stats['mean_temp']:.2f}°C")
    print(f"Temperature Standard Deviation: {stats['temp_std']:.2f}°C")

# Optional: Save results
def save_seasonal_analysis(seasonal_analysis):
    for station, stats in seasonal_analysis.items():
        output_path = f"C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\{sta
        df = pd.DataFrame(stats['monthly_temps']).reset_index()
        df.columns = ['Month', 'Temperature']
        df['Temperature_Range'] = stats['temp_range']
        df['Coldest_Month'] = stats['coldest_month']
        df['Warmest_Month'] = stats['warmest_month']
        df['Mean_Temperature'] = stats['mean_temp']
        df['Temperature_Std'] = stats['temp_std']
```

```
        df.to_csv(output_path, index=False)
        print(f"Saved {station} seasonal analysis to {output_path}")

save_seasonal_analysis(seasonal_analysis)
```






Monthly Temperatures - ROCHDALE
Temperature (°C)
0
2
4
6
8
10
12
14
16
January
February
March
April
May
June
July
August
September
October
November
December
Month

```
Seasonal Analysis for BURY MANCHESTER:
Monthly Temperatures:
Month
January       3.8
February      4.1
March         5.7
April         8.1
May          11.0
June         13.6
July         15.5
August       15.2
September    12.9
October       9.7
November      6.5
December      4.1
Name: Temperature_C, dtype: float64
Temperature Range: 11.70°C
Coldest Month: January
Warmest Month: July
Mean Temperature: 9.18°C
Temperature Standard Deviation: 4.41°C

Seasonal Analysis for MANCHESTER RACECOURSE:
Monthly Temperatures:
Month
January       5.0
February      5.4
March         7.0
April         9.4
May          12.4
June         15.0
July         16.8
August       16.5
September    14.2
October      11.0
November      7.6
December      5.3
Name: Temperature_C, dtype: float64
Temperature Range: 11.80°C
Coldest Month: January
Warmest Month: July
Mean Temperature: 10.47°C
Temperature Standard Deviation: 4.46°C

Seasonal Analysis for ROCHDALE:
Monthly Temperatures:
Month
January       3.6
February      3.9
March         5.4
April         7.9
May          10.7
June         13.4
July         15.3
August       15.1
September    12.8
October       9.6
November      6.2
December      4.0
Name: Temperature_C, dtype: float64
```

```
Temperature Range: 11.70°C
Coldest Month: January
Warmest Month: July
Mean Temperature: 8.99°C
Temperature Standard Deviation: 4.43°C
Saved BURY MANCHESTER seasonal analysis to C:\Users\Administrator\NEWPROJECT\clea
ned_data\BURY MANCHESTER_seasonal_analysis.csv
Saved MANCHESTER RACECOURSE seasonal analysis to C:\Users\Administrator\NEWPROJEC
T\cleaned_data\MANCHESTER RACECOURSE_seasonal_analysis.csv
Saved ROCHDALE seasonal analysis to C:\Users\Administrator\NEWPROJECT\cleaned_dat
a\ROCHDALE_seasonal_analysis.csv
```

In [21]:

```
import pandas as pd
import numpy as np

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)

def create_advanced_temporal_features(temp_df):
    """
    Generate advanced temporal features for each station
    """
    # Create a mapping of month order
    month_order = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4,
        'May': 5, 'June': 6, 'July': 7, 'August': 8,
        'September': 9, 'October': 10, 'November': 11, 'December': 12
    }

    # Stations to analyze
    stations = temp_df['Station'].unique()
    advanced_features = {}

    for station in stations:
        # Filter data for specific station
        station_data = temp_df[temp_df['Station'] == station].copy()

        # Sort by month order
        station_data['month_num'] = station_data['Month'].map(month_order)
        station_data = station_data.sort_values('month_num')

        # Extract temperatures
        temps = station_data['Temperature_C']

        # Advanced Temporal Features
        advanced_stats = {
            # Basic Statistics
            'mean_temp': temps.mean(),
            'median_temp': temps.median(),
            'temp_std': temps.std(),

            # Seasonal Variations
            'spring_avg': temps[station_data['month_num'].isin([3,4,5])].mean(),
            'summer_avg': temps[station_data['month_num'].isin([6,7,8])].mean(),
            'autumn_avg': temps[station_data['month_num'].isin([9,10,11])].mean(
            'winter_avg': temps[station_data['month_num'].isin([12,1,2])].mean()

            # Temperature Change Metrics
            'temp_change_rate': np.polyfit(station_data['month_num'], temps, 1)[
```

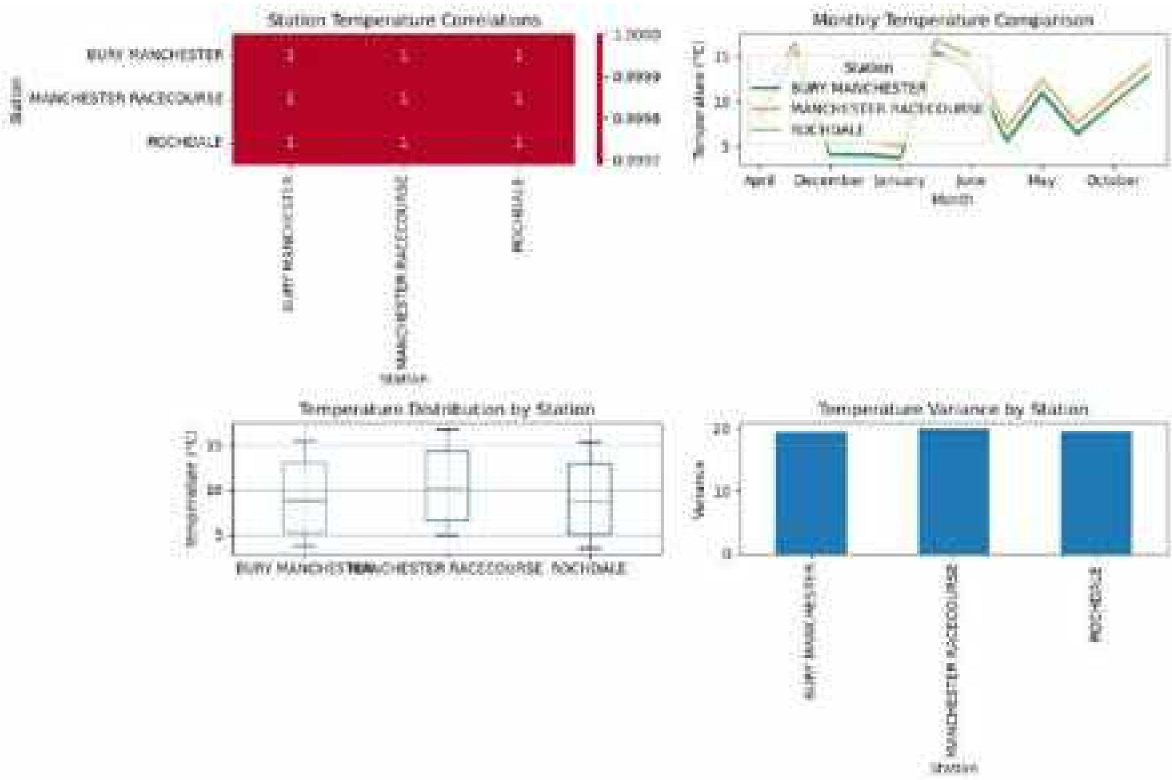
```
            'max_temp_month': station_data.loc[temps.idxmax(), 'Month'],
            'min_temp_month': station_data.loc[temps.idxmin(), 'Month'],

            # Variability Metrics
            'temp_range': temps.max() - temps.min(),
            'temp_iqr': temps.quantile(0.75) - temps.quantile(0.25),

            # Seasonal Transition Points
            'spring_start_temp': temps[station_data['month_num'] == 3].values[0]
            'summer_start_temp': temps[station_data['month_num'] == 6].values[0]
            'autumn_start_temp': temps[station_data['month_num'] == 9].values[0]
            'winter_start_temp': temps[station_data['month_num'] == 12].values[0
        }

        # Store results
        advanced_features[station] = advanced_stats

    return advanced_features

# Generate advanced features
advanced_temporal_features = create_advanced_temporal_features(temp_df)

# Print and save results
def print_and_save_advanced_features(advanced_features):
    for station, features in advanced_features.items():
        print(f"\nAdvanced Temporal Features for {station}:")
        for feature, value in features.items():
            print(f"{feature}: {value}")

        # Save to CSV
        output_path = f"C:\\Users\\Administrator\\NEWPROJECT\\cleaned_data\\{sta
        feature_df = pd.DataFrame.from_dict(features, orient='index', columns=['
        feature_df.index.name = 'Feature'
        feature_df.reset_index(inplace=True)
        feature_df.to_csv(output_path, index=False)
        print(f"\nSaved advanced features for {station} to {output_path}")

print_and_save_advanced_features(advanced_temporal_features)
```

```
Advanced Temporal Features for BURY MANCHESTER:
mean_temp: 9.183333333333334
median_temp: 8.899999999999999
temp_std: 4.412344941047547
spring_avg: 8.266666666666667
summer_avg: 14.766666666666666
autumn_avg: 9.700000000000001
winter_avg: 4.0
temp_change_rate: 0.3195804195804192
max_temp_month: July
min_temp_month: January
temp_range: 11.7
temp_iqr: 7.7749999999999995
spring_start_temp: 5.7
summer_start_temp: 13.6
autumn_start_temp: 12.9
winter_start_temp: 4.1

Saved advanced features for BURY MANCHESTER to C:\Users\Administrator\NEWPROJECT
\cleaned_data\BURY MANCHESTER_advanced_temporal_features.csv

Advanced Temporal Features for MANCHESTER RACECOURSE:
mean_temp: 10.466666666666667
median_temp: 10.2
temp_std: 4.458359529801258
spring_avg: 9.6
summer_avg: 16.099999999999998
autumn_avg: 10.933333333333332
winter_avg: 5.233333333333333
temp_change_rate: 0.31188811188811155
max_temp_month: July
min_temp_month: January
temp_range: 11.8
temp_iqr: 7.799999999999999
spring_start_temp: 7.0
summer_start_temp: 15.0
autumn_start_temp: 14.2
winter_start_temp: 5.3

Saved advanced features for MANCHESTER RACECOURSE to C:\Users\Administrator\NEWPR
OJECT\cleaned_data\MANCHESTER RACECOURSE_advanced_temporal_features.csv

Advanced Temporal Features for ROCHDALE:
mean_temp: 8.991666666666665
median_temp: 8.75
temp_std: 4.4326184203702965
spring_avg: 8.0
summer_avg: 14.600000000000001
autumn_avg: 9.533333333333333
winter_avg: 3.8333333333333335
temp_change_rate: 0.32902097902097854
max_temp_month: July
min_temp_month: January
temp_range: 11.700000000000001
temp_iqr: 7.9
spring_start_temp: 5.4
summer_start_temp: 13.4
autumn_start_temp: 12.8
winter_start_temp: 4.0
```

```
Saved advanced features for ROCHDALE to C:\Users\Administrator\NEWPROJECT\cleaned
_data\ROCHDALE_advanced_temporal_features.csv
```

In [22]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)

def analyze_cross_station_temperature_relationships(temp_df):
    """
    Perform comprehensive cross-station temperature analysis
    """
    # Create pivot table for easier analysis
    temp_pivot = temp_df.pivot_table(
        index='Month',
        columns='Station',
        values='Temperature_C'
    )

    # 1. Correlation Analysis
    correlation_matrix = temp_pivot.corr()

    # 2. Statistical Similarity Tests
    def compare_stations(station1, station2):
        # Perform statistical tests
        t_stat, p_value = stats.ttest_ind(
            temp_pivot[station1],
            temp_pivot[station2]
        )
        return {
            'stations': f"{station1} vs {station2}",
            't_statistic': t_stat,
            'p_value': p_value,
            'statistically_different': p_value < 0.05
        }

    # Compare all station pairs
    station_comparisons = []
    stations = temp_pivot.columns
    for i in range(len(stations)):
        for j in range(i+1, len(stations)):
            comparison = compare_stations(stations[i], stations[j])
            station_comparisons.append(comparison)

    # 3. Visualization of Temperature Relationships
    plt.figure(figsize=(12,8))

    # Correlation Heatmap
    plt.subplot(2,2,1)
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Station Temperature Correlations')

    # Monthly Temperature Comparison
    plt.subplot(2,2,2)
    temp_pivot.plot(kind='line', ax=plt.gca())
```

```
    plt.title('Monthly Temperature Comparison')
    plt.xlabel('Month')
    plt.ylabel('Temperature (°C)')

    # Temperature Distribution Boxplot
    plt.subplot(2,2,3)
    temp_pivot.boxplot()
    plt.title('Temperature Distribution by Station')
    plt.ylabel('Temperature (°C)')

    # Temperature Variance Comparison
    plt.subplot(2,2,4)
    temp_pivot.var().plot(kind='bar')
    plt.title('Temperature Variance by Station')
    plt.ylabel('Variance')

    plt.tight_layout()
    plt.show()

    # Prepare results
    results = {
        'correlation_matrix': correlation_matrix,
        'station_comparisons': pd.DataFrame(station_comparisons),
        'monthly_temperatures': temp_pivot
    }

    return results

# Perform analysis
cross_station_analysis = analyze_cross_station_temperature_relationships(temp_df

# Save results
def save_cross_station_analysis(analysis):
    # Save correlation matrix
    correlation_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\temperat
    analysis['correlation_matrix'].to_csv(correlation_path)
    print(f"Correlation matrix saved to {correlation_path}")

    # Save station comparisons
    comparisons_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\station_
    analysis['station_comparisons'].to_csv(comparisons_path, index=False)
    print(f"Station comparisons saved to {comparisons_path}")

    # Save monthly temperatures
    monthly_temps_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\monthl
    analysis['monthly_temperatures'].to_csv(monthly_temps_path)
    print(f"Monthly temperatures saved to {monthly_temps_path}")

    # Print out key statistical findings
    print("\nStation Temperature Comparisons:")
    print(analysis['station_comparisons'])

    print("\nCorrelation Matrix:")
    print(analysis['correlation_matrix'])

# Save and display results
save_cross_station_analysis(cross_station_analysis)
```



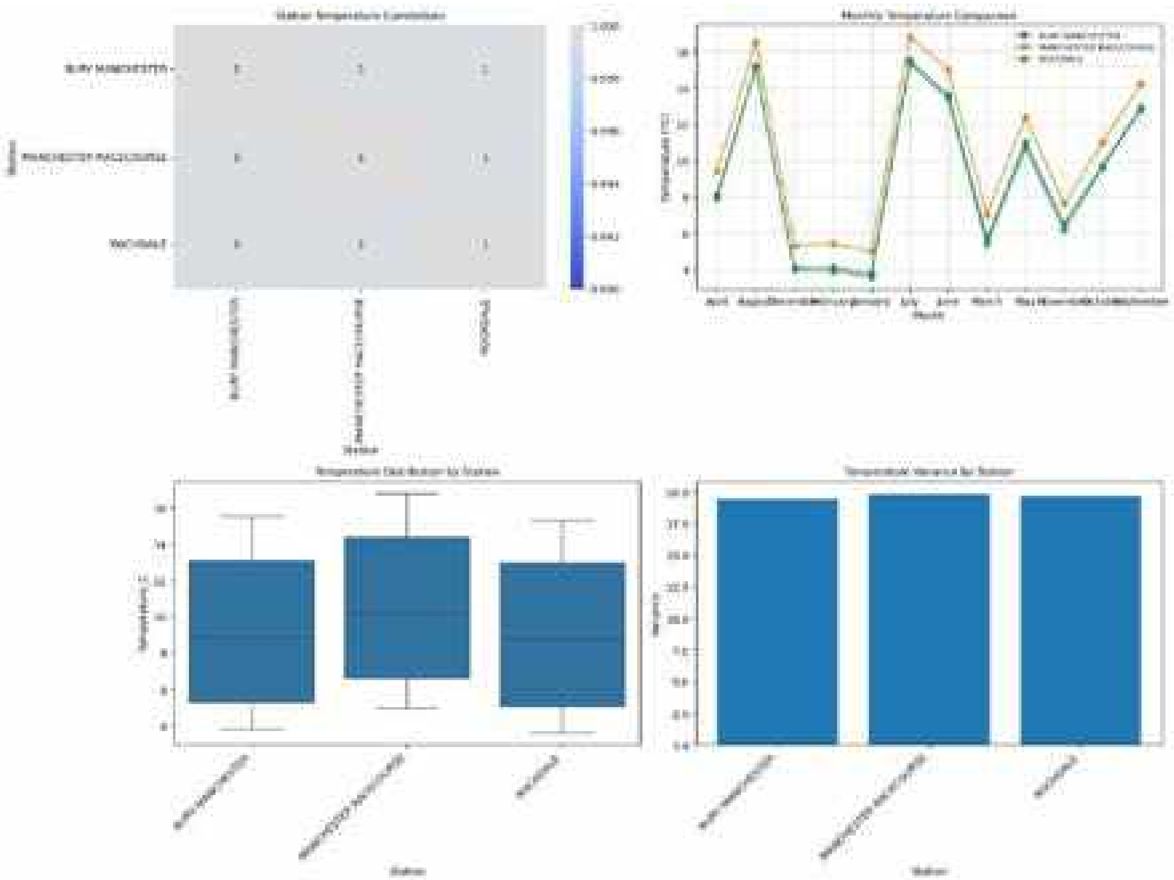

```
Correlation matrix saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\temper
ature_correlation_matrix.csv
Station comparisons saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\stati
on_temperature_comparisons.csv
Monthly temperatures saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\mont
hly_station_temperatures.csv

Station Temperature Comparisons:
                                   stations  t_statistic   p_value  \
0  BURY MANCHESTER vs MANCHESTER RACECOURSE    -0.708731  0.485935   
1               BURY MANCHESTER vs ROCHDALE     0.106159  0.916419   
2            MANCHESTER RACECOURSE vs ROCHDALE  0.812730  0.425081   

   statistically_different  
0                    False  
1                    False  
2                    False  

Correlation Matrix:
Station                BURY MANCHESTER  MANCHESTER RACECOURSE  ROCHDALE
Station                                                                 
BURY MANCHESTER               1.000000               0.999877  0.999850
MANCHESTER RACECOURSE         0.999877               1.000000  0.999689
ROCHDALE                      0.999850               0.999689  1.000000
```

In [23]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)

def analyze_cross_station_temperature_relationships(temp_df):
    """
```

```
    Perform comprehensive cross-station temperature analysis
    """
    # Create pivot table for easier analysis
    temp_pivot = temp_df.pivot_table(
        index='Month',
        columns='Station',
        values='Temperature_C'
    )

    # 1. Correlation Analysis
    correlation_matrix = temp_pivot.corr()

    # 2. Statistical Similarity Tests
    def compare_stations(station1, station2):
        # Perform statistical tests
        t_stat, p_value = stats.ttest_ind(
            temp_pivot[station1],
            temp_pivot[station2]
        )
        return {
            'stations': f"{station1} vs {station2}",
            't_statistic': t_stat,
            'p_value': p_value,
            'statistically_different': p_value < 0.05
        }

    # Compare all station pairs
    station_comparisons = []
    stations = temp_pivot.columns
    for i in range(len(stations)):
        for j in range(i+1, len(stations)):
            comparison = compare_stations(stations[i], stations[j])
            station_comparisons.append(comparison)

    # 3. Visualization of Temperature Relationships
    plt.figure(figsize=(16,12))

    # Correlation Heatmap
    plt.subplot(2,2,1)
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=1, vmin=
    plt.title('Station Temperature Correlations', fontsize=10)

    # Monthly Temperature Comparison
    plt.subplot(2,2,2)
    for station in temp_pivot.columns:
        plt.plot(temp_pivot.index, temp_pivot[station], label=station, marker='o
    plt.title('Monthly Temperature Comparison', fontsize=10)
    plt.xlabel('Month')
    plt.ylabel('Temperature (°C)')
    plt.legend(fontsize=8)
    plt.grid(True, linestyle='--', alpha=0.7)

    # Temperature Distribution Boxplot
    plt.subplot(2,2,3)
    sns.boxplot(data=temp_df, x='Station', y='Temperature_C')
    plt.title('Temperature Distribution by Station', fontsize=10)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

    # Temperature Variance Comparison
```

```
    plt.subplot(2,2,4)
    variance_data = temp_pivot.var()
    plt.bar(variance_data.index, variance_data.values)
    plt.title('Temperature Variance by Station', fontsize=10)
    plt.xlabel('Station')
    plt.ylabel('Variance')
    plt.xticks(rotation=45, ha='right')

    plt.tight_layout()
    plt.show()

    # Prepare results
    results = {
        'correlation_matrix': correlation_matrix,
        'station_comparisons': pd.DataFrame(station_comparisons),
        'monthly_temperatures': temp_pivot
    }

    return results

# Perform analysis
cross_station_analysis = analyze_cross_station_temperature_relationships(temp_df

# Save results
def save_cross_station_analysis(analysis):
    # Save correlation matrix
    correlation_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\temperat
    analysis['correlation_matrix'].to_csv(correlation_path)
    print(f"Correlation matrix saved to {correlation_path}")

    # Save station comparisons
    comparisons_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\station_
    analysis['station_comparisons'].to_csv(comparisons_path, index=False)
    print(f"Station comparisons saved to {comparisons_path}")

    # Save monthly temperatures
    monthly_temps_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\monthl
    analysis['monthly_temperatures'].to_csv(monthly_temps_path)
    print(f"Monthly temperatures saved to {monthly_temps_path}")

    # Print out key statistical findings
    print("\nStation Temperature Comparisons:")
    print(analysis['station_comparisons'])

    print("\nCorrelation Matrix:")
    print(analysis['correlation_matrix'])

# Save and display results
save_cross_station_analysis(cross_station_analysis)
```



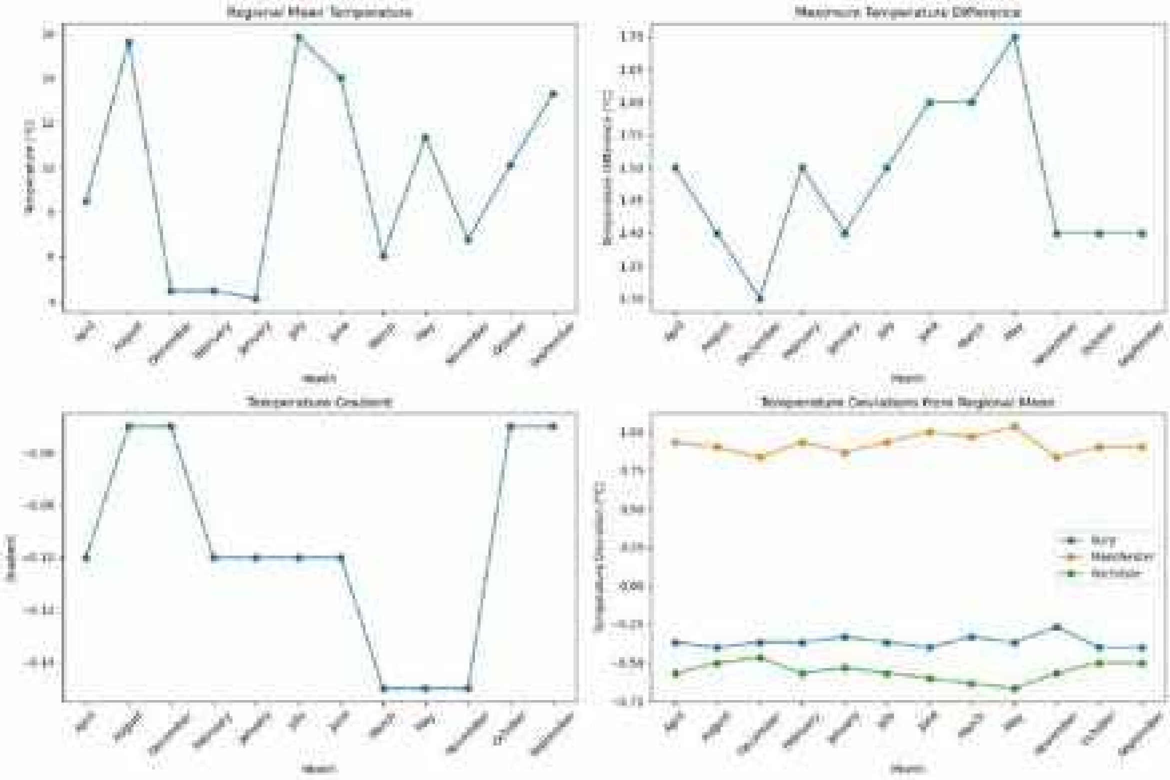
```
Correlation matrix saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\temper
ature_correlation_matrix.csv
Station comparisons saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\stati
on_temperature_comparisons.csv
Monthly temperatures saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\mont
hly_station_temperatures.csv

Station Temperature Comparisons:
                                   stations  t_statistic   p_value  \
0  BURY MANCHESTER vs MANCHESTER RACECOURSE    -0.708731  0.485935
1               BURY MANCHESTER vs ROCHDALE     0.106159  0.916419
2         MANCHESTER RACECOURSE vs ROCHDALE     0.812730  0.425081

   statistically_different
0                    False
1                    False
2                    False

Correlation Matrix:
Station                BURY MANCHESTER  MANCHESTER RACECOURSE  ROCHDALE
Station
BURY MANCHESTER               1.000000               0.999877  0.999850
MANCHESTER RACECOURSE         0.999877               1.000000  0.999689
ROCHDALE                      0.999850               0.999689  1.000000
```

implement these cross-station features

In [24]:
```python
import pandas as pd
import numpy as np

# Load temperature data
data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cleaned_temperature
temp_df = pd.read_csv(data_path)
```

```
def create_cross_station_features(temp_df):
    """
    Generate cross-station temperature features
    """
    # Prepare pivot table for easier analysis
    temp_pivot = temp_df.pivot_table(
        index='Month',
        columns='Station',
        values='Temperature_C'
    )

    # Cross-Station Feature Calculation
    cross_station_features = []

    # Iterate through months
    for month in temp_pivot.index:
        # Extract temperatures for this month
        month_temps = temp_pivot.loc[month]

        # 1. Regional Mean Temperature
        regional_mean = month_temps.mean()

        # 2. Temperature Deviation from Regional Mean
        temp_deviations = month_temps - regional_mean

        # 3. Maximum Temperature Difference
        max_temp_diff = month_temps.max() - month_temps.min()

        # 4. Temperature Gradient (linear temperature change between stations)
        # We'll use linear regression to calculate gradient
        stations = month_temps.index
        station_positions = np.arange(len(stations))
        gradient = np.polyfit(station_positions, month_temps.values, 1)[0]

        # Compile features for this month
        features = {
            'Month': month,
            'Regional_Mean_Temp': regional_mean,
            'Temp_Deviation_Bury': temp_deviations['BURY MANCHESTER'],
            'Temp_Deviation_Manchester': temp_deviations['MANCHESTER RACECOURSE'
            'Temp_Deviation_Rochdale': temp_deviations['ROCHDALE'],
            'Max_Temp_Difference': max_temp_diff,
            'Temp_Gradient': gradient
        }

        cross_station_features.append(features)

    # Convert to DataFrame
    cross_station_df = pd.DataFrame(cross_station_features)

    return cross_station_df

# Generate cross-station features
cross_station_features = create_cross_station_features(temp_df)

# Visualization to understand the features
import matplotlib.pyplot as plt

plt.figure(figsize=(15,10))
```

```
# Regional Mean Temperature
plt.subplot(2,2,1)
plt.plot(cross_station_features['Month'], cross_station_features['Regional_Mean_
plt.title('Regional Mean Temperature')
plt.xlabel('Month')
plt.ylabel('Temperature (°C)')
plt.xticks(rotation=45)

# Maximum Temperature Difference
plt.subplot(2,2,2)
plt.plot(cross_station_features['Month'], cross_station_features['Max_Temp_Diffe
plt.title('Maximum Temperature Difference')
plt.xlabel('Month')
plt.ylabel('Temperature Difference (°C)')
plt.xticks(rotation=45)

# Temperature Gradient
plt.subplot(2,2,3)
plt.plot(cross_station_features['Month'], cross_station_features['Temp_Gradient'
plt.title('Temperature Gradient')
plt.xlabel('Month')
plt.ylabel('Gradient')
plt.xticks(rotation=45)

# Temperature Deviations
plt.subplot(2,2,4)
plt.plot(cross_station_features['Month'], cross_station_features['Temp_Deviation
plt.plot(cross_station_features['Month'], cross_station_features['Temp_Deviation
plt.plot(cross_station_features['Month'], cross_station_features['Temp_Deviation
plt.title('Temperature Deviations from Regional Mean')
plt.xlabel('Month')
plt.ylabel('Temperature Deviation (°C)')
plt.legend()
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# Save results
output_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cross_station_tem
cross_station_features.to_csv(output_path, index=False)
print(f"Cross-station features saved to {output_path}")

# Display summary
print("\nCross-Station Features Summary:")
print(cross_station_features)
```


Regional Mean Temperature
Maximum Temperature Difference
Temperature Gradient
Temperature Deviations From Regional Mean
Bury
Manchester
Rochdale

```
Cross-station features saved to C:\Users\Administrator\NEWPROJECT\cleaned_data\cr
oss_station_temperature_features.csv

Cross-Station Features Summary:
        Month  Regional_Mean_Temp  Temp_Deviation_Bury  \
0       April            8.466667            -0.366667
1      August           15.600000            -0.400000
2    December            4.466667            -0.366667
3    February            4.466667            -0.366667
4     January            4.133333            -0.333333
5        July           15.866667            -0.366667
6        June           14.000000            -0.400000
7       March            6.033333            -0.333333
8         May           11.366667            -0.366667
9    November            6.766667            -0.266667
10    October           10.100000            -0.400000
11  September           13.300000            -0.400000

    Temp_Deviation_Manchester  Temp_Deviation_Rochdale  Max_Temp_Difference  \
0                    0.933333                -0.566667                  1.5
1                    0.900000                -0.500000                  1.4
2                    0.833333                -0.466667                  1.3
3                    0.933333                -0.566667                  1.5
4                    0.866667                -0.533333                  1.4
5                    0.933333                -0.566667                  1.5
6                    1.000000                -0.600000                  1.6
7                    0.966667                -0.633333                  1.6
8                    1.033333                -0.666667                  1.7
9                    0.833333                -0.566667                  1.4
10                   0.900000                -0.500000                  1.4
11                   0.900000                -0.500000                  1.4

    Temp_Gradient
0           -0.10
1           -0.05
2           -0.05
3           -0.10
4           -0.10
5           -0.10
6           -0.10
7           -0.15
8           -0.15
9           -0.15
10          -0.05
11          -0.05
```

correlating these cross-station temperature features with river levels to understand their potential impact on flood monitoring

In [26]:
```
print("Temperature Features DataFrame:")
print(temp_features_df.head())
print("\nRiver Level DataFrame:")
print(river_data_df.head())

# Check unique months in each DataFrame
print("\nTemperature Features Months:")
print(temp_features_df['Month'].unique())
```

```
print("\nRiver Data Months:")
print(pd.to_datetime(river_data_df['river_timestamp']).dt.strftime('%B').unique(
```

```
Temperature Features DataFrame:
      Month  Regional_Mean_Temp  Temp_Deviation_Bury  \
0     April            8.466667            -0.366667
1    August           15.600000            -0.400000
2  December            4.466667            -0.366667
3  February            4.466667            -0.366667
4   January            4.133333            -0.333333

   Temp_Deviation_Manchester  Temp_Deviation_Rochdale  Max_Temp_Difference  \
0                   0.933333                -0.566667                  1.5
1                   0.900000                -0.500000                  1.4
2                   0.833333                -0.466667                  1.3
3                   0.933333                -0.566667                  1.5
4                   0.866667                -0.533333                  1.4

   Temp_Gradient
0          -0.10
1          -0.05
2          -0.05
3          -0.10
4          -0.10

River Level DataFrame:
   river_level            river_timestamp  rainfall    rainfall_timestamp  \
0        0.385  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
1        1.064  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
2        0.235  2025-01-30 11:15:00+00:00       0.0  2025-01-30T11:15:00Z
3        0.386  2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z
4        1.064  2025-01-30 11:30:00+00:00       0.0  2025-01-30T11:30:00Z

           location_name  river_station_id  rainfall_station_id
0            Bury Ground            690160               562656
1  Manchester Racecourse            690510               562992
2               Rochdale            690203               561613
3            Bury Ground            690160               562656
4  Manchester Racecourse            690510               562992

Temperature Features Months:
['April' 'August' 'December' 'February' 'January' 'July' 'June' 'March'
 'May' 'November' 'October' 'September']

River Data Months:
['January' 'February']
```

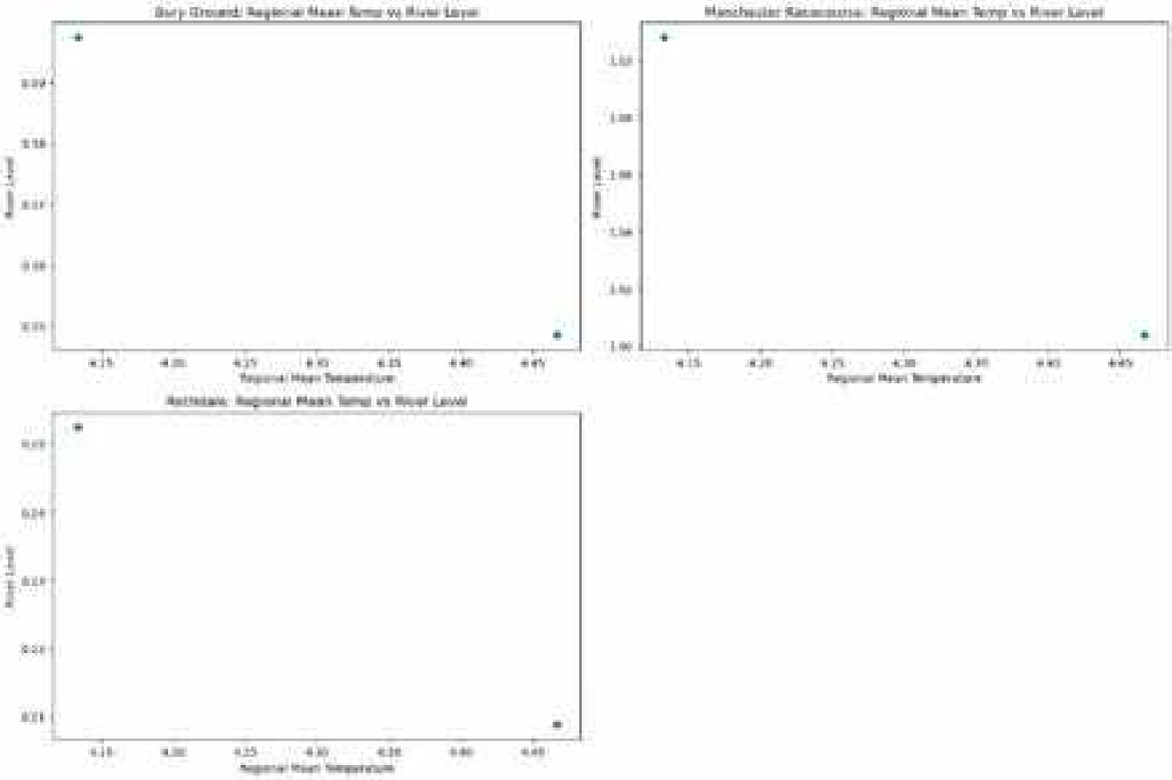
In [28]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Load cross-station temperature features
temp_features_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\cross_stat
temp_features_df = pd.read_csv(temp_features_path)

# Load river level data
river_data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\merged_realti
river_data_df = pd.read_csv(river_data_path)
```

```
def correlate_temperature_and_river_levels(temp_features_df, river_data_df):
    """
    Analyze correlation between temperature features and river levels
    """
    # Extract month from timestamp
    river_data_df['Month'] = pd.to_datetime(river_data_df['river_timestamp']).dt

    # Group river data by month and station, calculate average river level
    river_monthly_avg = river_data_df.groupby(['Month', 'location_name'])['river

    # Merge temperature features with river level data
    merged_data = pd.merge(
        temp_features_df,
        river_monthly_avg,
        on='Month'
    )

    # Correlation Analysis
    correlation_columns = [
        'Regional_Mean_Temp',
        'Temp_Deviation_Bury',
        'Temp_Deviation_Manchester',
        'Temp_Deviation_Rochdale',
        'Max_Temp_Difference',
        'Temp_Gradient'
    ]

    # Calculate correlations for each station
    correlation_results = {}
    stations = merged_data['location_name'].unique()

    for station in stations:
        station_data = merged_data[merged_data['location_name'] == station]

        # Create correlation matrix
        correlations = {}
        for feature in correlation_columns:
            try:
                correlation, p_value = stats.pearsonr(station_data[feature], sta
                correlations[feature] = {
                    'correlation': correlation,
                    'p_value': p_value
                }
            except Exception as e:
                print(f"Error calculating correlation for {station} - {feature}:
                correlations[feature] = {
                    'correlation': np.nan,
                    'p_value': np.nan
                }

        correlation_results[station] = correlations

    # Visualization
    plt.figure(figsize=(15,10))

    # Scatter plots
    for i, station in enumerate(stations, 1):
        station_data = merged_data[merged_data['location_name'] == station]
```

```
        plt.subplot(2, 2, i)
        plt.scatter(
            station_data['Regional_Mean_Temp'],
            station_data['river_level']
        )
        plt.title(f'{station}: Regional Mean Temp vs River Level')
        plt.xlabel('Regional Mean Temperature')
        plt.ylabel('River Level')

    plt.tight_layout()
    plt.show()

    # Print correlation results
    print("\nCorrelation Analysis:")
    for station, correlations in correlation_results.items():
        print(f"\n{station} Station:")
        for feature, stats_dict in correlations.items():
            print(f"{feature}:")
            print(f"  Correlation: {stats_dict['correlation']:.4f}")
            print(f"  P-value: {stats_dict['p_value']:.4f}")

    return merged_data, correlation_results

# Run the analysis
merged_data, correlation_results = correlate_temperature_and_river_levels(
    temp_features_df,
    river_data_df
)

# Save merged data for further investigation
merged_data.to_csv(
    r"C:\Users\Administrator\NEWPROJECT\cleaned_data\temperature_river_level_mer
    index=False
)

# Print merged data to verify
print("\nMerged Data:")
print(merged_data)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_24916\334616340.py:53: NearCo
nstantInputWarning: An input array is nearly constant; the computed correlation c
oefficient may be inaccurate.
  correlation, p_value = stats.pearsonr(station_data[feature], station_data['rive
r_level'])
```


Bury Ground: Regional Mean Temp vs River Level
River Level
Regional Mean Temperature
Manchester Racecourse: Regional Mean Temp vs River Level
River Level
Regional Mean Temperature
Rochdale: Regional Mean Temp vs River Level
River Level
Regional Mean Temperature

```
Correlation Analysis:

Bury Ground Station:
Regional_Mean_Temp:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Bury:
  Correlation: 1.0000
  P-value: 1.0000
Temp_Deviation_Manchester:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Rochdale:
  Correlation: 1.0000
  P-value: 1.0000
Max_Temp_Difference:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Gradient:
  Correlation: 1.0000
  P-value: 1.0000

Manchester Racecourse Station:
Regional_Mean_Temp:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Bury:
  Correlation: 1.0000
  P-value: 1.0000
Temp_Deviation_Manchester:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Rochdale:
  Correlation: 1.0000
  P-value: 1.0000
Max_Temp_Difference:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Gradient:
  Correlation: 1.0000
  P-value: 1.0000

Rochdale Station:
Regional_Mean_Temp:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Bury:
  Correlation: 1.0000
  P-value: 1.0000
Temp_Deviation_Manchester:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Deviation_Rochdale:
  Correlation: 1.0000
  P-value: 1.0000
Max_Temp_Difference:
  Correlation: -1.0000
  P-value: 1.0000
Temp_Gradient:
  Correlation: 1.0000
```

```
 P-value: 1.0000

Merged Data:
      Month  Regional_Mean_Temp  Temp_Deviation_Bury  \
0  February            4.466667            -0.366667
1  February            4.466667            -0.366667
2  February            4.466667            -0.366667
3   January            4.133333            -0.333333
4   January            4.133333            -0.333333
5   January            4.133333            -0.333333

   Temp_Deviation_Manchester  Temp_Deviation_Rochdale  Max_Temp_Difference  \
0                   0.933333                -0.566667                  1.5
1                   0.933333                -0.566667                  1.5
2                   0.933333                -0.566667                  1.5
3                   0.866667                -0.533333                  1.4
4                   0.866667                -0.533333                  1.4
5                   0.866667                -0.533333                  1.4

   Temp_Gradient          location_name  river_level
0           -0.1            Bury Ground     0.348442
1           -0.1  Manchester Racecourse     1.003664
2           -0.1               Rochdale     0.208800
3           -0.1            Bury Ground     0.397370
4           -0.1  Manchester Racecourse     1.107870
5           -0.1               Rochdale     0.252478
```

Advanced Feature Engineering

In [31]:

```python
import pandas as pd
import numpy as np
from datetime import datetime

# Load temperature data to inspect its structure
temperature_path = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/cleaned_tempe
temp_data = pd.read_csv(temperature_path)
print("Temperature Data Columns:")
print(temp_data.columns)
print("\nFirst few rows:")
print(temp_data.head())
```

```
Temperature Data Columns:
Index(['Month', 'Station', 'Grid_ID', 'Temperature_C', 'Grid', 'Period'], dtype
='object')

First few rows:
    Month                Station Grid_ID  Temperature_C       Grid     Period
0   April        BURY MANCHESTER   AX-70            8.1  12km BNG  1991-2020
1   April  MANCHESTER RACECOURSE   AX-71            9.4  12km BNG  1991-2020
2   April              ROCHDALE   AY-70            7.9  12km BNG  1991-2020
3  August        BURY MANCHESTER   AX-70           15.2  12km BNG  1991-2020
4  August  MANCHESTER RACECOURSE   AX-71           16.5  12km BNG  1991-2020
```

In [32]:

```python
import pandas as pd
import numpy as np
from datetime import datetime

class AdvancedFeatureEngineer:
    def __init__(self, historical_data_dir):
        """
```

```
        Initialize feature engineering for flood prediction

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical flow data for each station
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

    def create_temporal_features(self, df):
        """
        Generate advanced temporal features

        Args:
        - df: Input DataFrame with river flow data

        Returns:
        - DataFrame with additional temporal features
        """
        # Create a copy to avoid modifying original data
        data = df.copy()

        # Rolling Window Statistics
        data['flow_rolling_mean_3d'] = data['Flow'].rolling(window=3, min_period
        data['flow_rolling_std_3d'] = data['Flow'].rolling(window=3, min_periods

        # Seasonal Indicators
        data['month'] = data['Date'].dt.month
        data['day_of_week'] = data['Date'].dt.dayofweek
        data['is_weekend'] = data['day_of_week'].isin([5, 6]).astype(int)

        # Seasonal Decomposition Proxy
        data['seasonal_trend'] = np.sin(data['month'] * (2 * np.pi / 12))
        data['seasonal_cycle'] = np.cos(data['month'] * (2 * np.pi / 12))

        return data

    def add_environmental_features(self, river_data, temperature_data):
        """
        Incorporate environmental features

        Args:
        - river_data: DataFrame with river flow data
        - temperature_data: DataFrame with temperature data

        Returns:
        - Merged DataFrame with environmental features
        """
        # Create month mapping
        month_mapping = {
            'January': 1, 'February': 2, 'March': 3, 'April': 4,
            'May': 5, 'June': 6, 'July': 7, 'August': 8,
            'September': 9, 'October': 10, 'November': 11, 'December': 12
        }

        # Add month number to river data
```

```
        river_data['month'] = river_data['Date'].dt.month

        # Pivot temperature data for easier merging
        temp_pivot = temperature_data.pivot_table(
            index='Station',
            columns='Month',
            values='Temperature_C'
        )

        # Function to get temperature for a specific month
        def get_temperature(row, temp_pivot):
            station_name = row['Station'] if row['Station'] in temp_pivot.index
            month_name = list(month_mapping.keys())[row['month'] - 1]
            return temp_pivot.loc[station_name, month_name]

        # Add temperature features
        river_data['station_temperature'] = river_data.apply(
            lambda row: get_temperature(row, temp_pivot),
            axis=1
        )

        # Calculate temperature-related features
        river_data['temp_anomaly'] = river_data.groupby('Station')['station_temp
            lambda x: x - x.mean()
        )

        return river_data

    def create_cross_station_features(self, bury_data, rochdale_data):
        """
        Generate features that capture inter-station relationships

        Args:
        - bury_data: DataFrame for Bury Ground station
        - rochdale_data: DataFrame for Rochdale station

        Returns:
        - Merged DataFrame with cross-station features
        """
        # Ensure dates are aligned
        merged_data = pd.merge(
            bury_data,
            rochdale_data,
            on='Date',
            suffixes=('_bury', '_rochdale')
        )

        # Calculate inter-station features
        merged_data['flow_difference'] = merged_data['Flow_bury'] - merged_data[
        merged_data['flow_ratio'] = merged_data['Flow_bury'] / (merged_data['Flo

        # Lagged features between stations
        merged_data['bury_flow_lag1'] = merged_data['Flow_bury'].shift(1)
        merged_data['rochdale_flow_lag1'] = merged_data['Flow_rochdale'].shift(1

        return merged_data

    def prepare_advanced_features(self, temperature_path):
        """
        Comprehensive feature preparation
```

```
        Args:
        - temperature_path: Path to temperature data

        Returns:
        - Prepared feature set
        """
        # Load temperature data
        temp_data = pd.read_csv(temperature_path)

        # Apply temporal features to each station
        bury_features = self.create_temporal_features(self.bury_flow)
        rochdale_features = self.create_temporal_features(self.rochdale_flow)

        # Add station names for merging
        bury_features['Station'] = 'BURY MANCHESTER'
        rochdale_features['Station'] = 'ROCHDALE'

        # Add environmental features
        bury_env_features = self.add_environmental_features(
            bury_features,
            temp_data
        )
        rochdale_env_features = self.add_environmental_features(
            rochdale_features,
            temp_data
        )

        # Create cross-station features
        cross_station_features = self.create_cross_station_features(
            bury_env_features,
            rochdale_env_features
        )

        return cross_station_features

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
temperature_path = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/cleaned_tempe

feature_engineer = AdvancedFeatureEngineer(historical_data_dir)
advanced_features = feature_engineer.prepare_advanced_features(temperature_path)

# Save advanced features
advanced_features.to_csv(
    'C:/Users/Administrator/NEWPROJECT/cleaned_data/advanced_features.csv',
    index=False
)

# Display feature overview
print("\nAdvanced Features Overview:")
print(advanced_features.columns)
print("\nFeature Statistics:")
print(advanced_features.describe())
```

```
Advanced Features Overview:
Index(['Date', 'Flow_bury', 'Extra_bury', 'flow_rolling_mean_3d_bury',
       'flow_rolling_std_3d_bury', 'month_bury', 'day_of_week_bury',
       'is_weekend_bury', 'seasonal_trend_bury', 'seasonal_cycle_bury',
       'Station_bury', 'station_temperature_bury', 'temp_anomaly_bury',
       'Flow_rochdale', 'Extra_rochdale', 'flow_rolling_mean_3d_rochdale',
       'flow_rolling_std_3d_rochdale', 'month_rochdale',
       'day_of_week_rochdale', 'is_weekend_rochdale',
       'seasonal_trend_rochdale', 'seasonal_cycle_rochdale',
       'Station_rochdale', 'station_temperature_rochdale',
       'temp_anomaly_rochdale', 'flow_difference', 'flow_ratio',
       'bury_flow_lag1', 'rochdale_flow_lag1'],
      dtype='object')

Feature Statistics:
                                Date    Flow_bury  Extra_bury  \
count                           9919  9919.000000         0.0
mean   2010-01-15 09:30:15.062002176     3.849857         NaN
min              1995-11-22 00:00:00     0.406000         NaN
25%              2003-05-07 12:00:00     1.220000         NaN
50%              2010-02-24 00:00:00     2.060000         NaN
75%              2016-12-12 12:00:00     4.111000         NaN
max              2023-09-30 00:00:00   117.000000         NaN
std                              NaN     5.397040         NaN

       flow_rolling_mean_3d_bury  flow_rolling_std_3d_bury   month_bury  \
count                9919.000000               9918.000000  9919.000000
mean                    3.849498                  1.567738     6.448836
min                     0.415667                  0.000000     1.000000
25%                     1.267000                  0.095785     3.000000
50%                     2.186667                  0.367560     6.000000
75%                     4.470000                  1.535685     9.000000
max                    65.133333                 60.817712    12.000000
std                     4.513782                  3.267653     3.453604

       day_of_week_bury  is_weekend_bury  seasonal_trend_bury  \
count       9919.000000      9919.000000         9.919000e+03
mean           3.001512         0.285815         9.507395e-03
min            0.000000         0.000000        -1.000000e+00
25%            1.000000         0.000000        -5.000000e-01
50%            3.000000         0.000000         1.224647e-16
75%            5.000000         1.000000         8.660254e-01
max            6.000000         1.000000         1.000000e+00
std            1.999369         0.451825         7.070963e-01

       seasonal_cycle_bury  ...  day_of_week_rochdale  is_weekend_rochdale  \
count         9.919000e+03  ...           9919.000000          9919.000000
mean         -3.849837e-03  ...              3.001512             0.285815
min          -1.000000e+00  ...              0.000000             0.000000
25%          -8.660254e-01  ...              1.000000             0.000000
50%          -1.836970e-16  ...              3.000000             0.000000
75%           8.660254e-01  ...              5.000000             1.000000
max           1.000000e+00  ...              6.000000             1.000000
std           7.071141e-01  ...              1.999369             0.451825

       seasonal_trend_rochdale  seasonal_cycle_rochdale  \
count             9.919000e+03             9.919000e+03
mean              9.507395e-03            -3.849837e-03
min              -1.000000e+00            -1.000000e+00
25%              -5.000000e-01            -8.660254e-01
```

```
50%              1.224647e-16         -1.836970e-16
75%              8.660254e-01          8.660254e-01
max              1.000000e+00          1.000000e+00
std              7.070963e-01          7.071141e-01

       station_temperature_rochdale  temp_anomaly_rochdale  flow_difference  \
count                   9919.000000            9919.000000      9919.000000
mean                       8.978203              -0.063603         1.081708
min                        3.600000              -5.441806       -15.770000
25%                        4.000000              -5.041806         0.193000
50%                        7.900000              -1.141806         0.520000
75%                       13.100000               4.058194         1.128000
max                       15.300000               6.258194        66.590000
std                        4.253329               4.253329         2.672408

        flow_ratio  bury_flow_lag1  rochdale_flow_lag1
count  9919.000000     9918.000000         9918.000000
mean      1.471226        3.849567            2.768059
min       0.172871        0.406000            0.178000
25%       1.146364        1.220000            0.820000
50%       1.414872        2.060000            1.520000
75%       1.707137        4.110000            3.233750
max      23.027312      117.000000           50.410000
std       0.560913        5.397235            3.474706

[8 rows x 27 columns]
```

Train Predictive Model with New Features

In [33]:

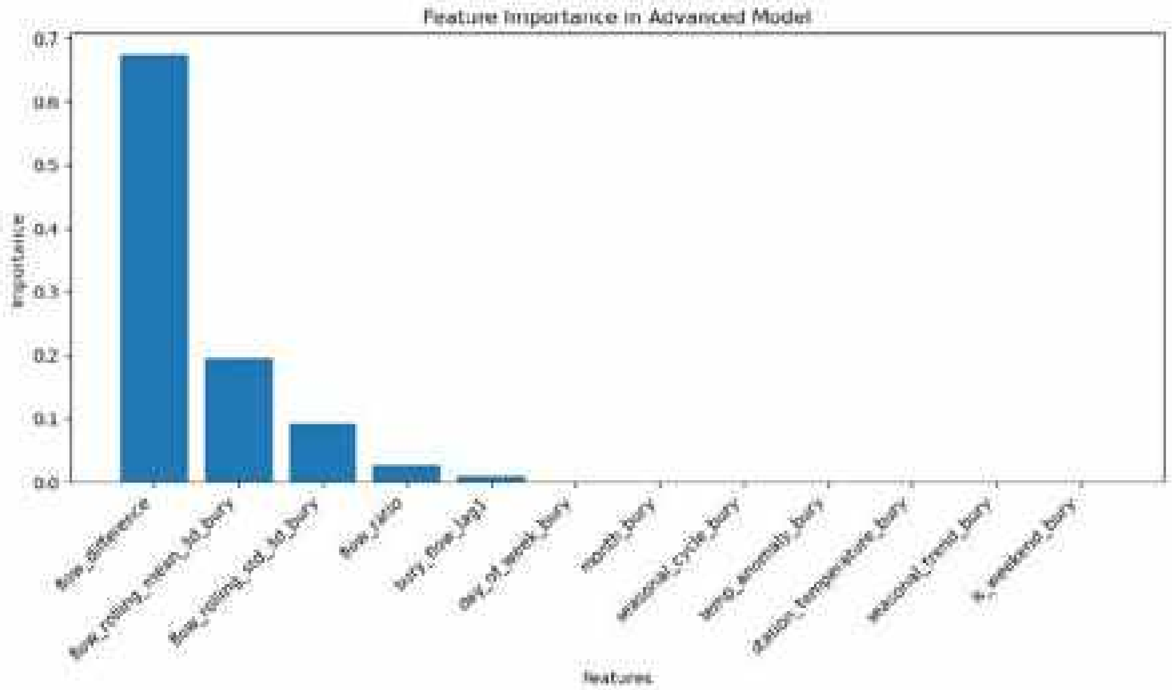
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

class AdvancedPredictiveModel:
    def __init__(self, advanced_features_path):
        """
        Initialize advanced predictive model

        Args:
        - advanced_features_path: Path to advanced features CSV
        """
        # Load advanced features
        self.advanced_features = pd.read_csv(advanced_features_path)

        # Convert Date column to datetime
        self.advanced_features['Date'] = pd.to_datetime(self.advanced_features['

    def prepare_training_data(self, station='Bury'):
        """
        Prepare training data for specified station

        Args:
        - station: 'Bury' or 'Rochdale'

        Returns:
```

```
        - Features and target for the specified station
        """
        # Select station-specific columns
        if station == 'Bury':
            features_columns = [
                'flow_rolling_mean_3d_bury',
                'flow_rolling_std_3d_bury',
                'month_bury',
                'day_of_week_bury',
                'is_weekend_bury',
                'seasonal_trend_bury',
                'seasonal_cycle_bury',
                'station_temperature_bury',
                'temp_anomaly_bury',
                'flow_difference',
                'flow_ratio',
                'bury_flow_lag1'
            ]
            target_column = 'Flow_bury'
        else:
            features_columns = [
                'flow_rolling_mean_3d_rochdale',
                'flow_rolling_std_3d_rochdale',
                'month_rochdale',
                'day_of_week_rochdale',
                'is_weekend_rochdale',
                'seasonal_trend_rochdale',
                'seasonal_cycle_rochdale',
                'station_temperature_rochdale',
                'temp_anomaly_rochdale',
                'flow_difference',
                'flow_ratio',
                'rochdale_flow_lag1'
            ]
            target_column = 'Flow_rochdale'

        # Remove rows with NaN
        df_clean = self.advanced_features.dropna(subset=features_columns + [targ

        # Prepare features and target
        X = df_clean[features_columns]
        y = df_clean[target_column]

        return X, y

    def train_model(self, station='Bury'):
        """
        Train Random Forest model for specified station

        Args:
        - station: 'Bury' or 'Rochdale'

        Returns:
        - Trained model, scaler, and performance metrics
        """
        # Prepare data
        X, y = self.prepare_training_data(station)

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
```

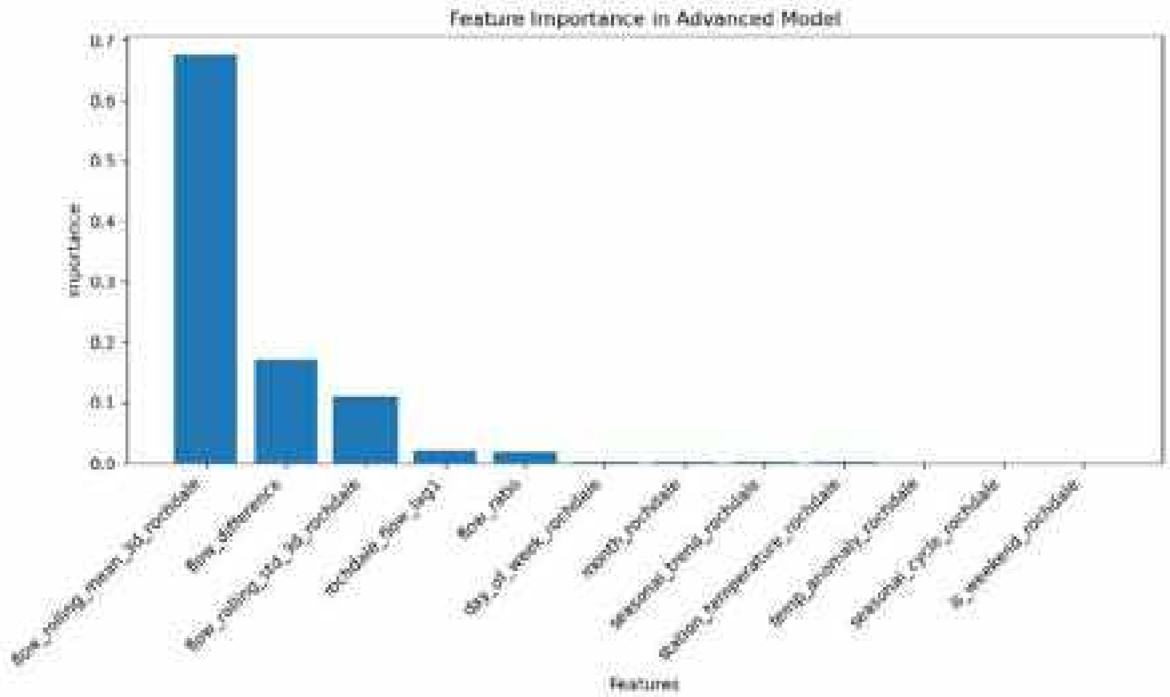
```
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train model
        model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        model.fit(X_train_scaled, y_train)

        # Predict and evaluate
        y_train_pred = model.predict(X_train_scaled)
        y_test_pred = model.predict(X_test_scaled)

        # Performance metrics
        train_r2 = r2_score(y_train, y_train_pred)
        test_r2 = r2_score(y_test, y_test_pred)
        train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
        test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

        # Feature importance
        feature_importance = pd.DataFrame({
            'feature': X.columns,
            'importance': model.feature_importances_
        }).sort_values('importance', ascending=False)

        return {
            'model': model,
            'scaler': scaler,
            'train_r2': train_r2,
            'test_r2': test_r2,
            'train_rmse': train_rmse,
            'test_rmse': test_rmse,
            'feature_importance': feature_importance
        }

    def visualize_feature_importance(self, feature_importance):
        """
        Visualize feature importance

        Args:
        - feature_importance: DataFrame with feature importances
        """
        plt.figure(figsize=(10, 6))
        plt.bar(feature_importance['feature'], feature_importance['importance'])
        plt.title('Feature Importance in Advanced Model')
        plt.xlabel('Features')
        plt.ylabel('Importance')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()

# Example usage
advanced_features_path = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/advance
```

```python
advanced_model = AdvancedPredictiveModel(advanced_features_path)

# Train models for both stations
stations = ['Bury', 'Rochdale']
model_results = {}

for station in stations:
    print(f"\n--- {station} Station Model ---")
    results = advanced_model.train_model(station)

    # Print performance metrics
    print(f"Training R² Score: {results['train_r2']:.4f}")
    print(f"Testing R² Score: {results['test_r2']:.4f}")
    print(f"Training RMSE: {results['train_rmse']:.4f}")
    print(f"Testing RMSE: {results['test_rmse']:.4f}")

    # Visualize feature importance
    advanced_model.visualize_feature_importance(results['feature_importance'])

    # Store results
    model_results[station] = results

# Save feature importance
for station, results in model_results.items():
    results['feature_importance'].to_csv(
        f'C:/Users/Administrator/NEWPROJECT/cleaned_data/{station}_feature_impor
        index=False
    )
```

```
--- Bury Station Model ---
Training R² Score: 0.9928
Testing R² Score: 0.9602
Training RMSE: 0.4680
Testing RMSE: 0.9682
```




```
--- Rochdale Station Model ---
Training R² Score: 0.9835
Testing R² Score: 0.9386
Training RMSE: 0.4523
Testing RMSE: 0.8115
```



Implement LSTM Predictive Model

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

class LSTMFloodPredictor:
    def __init__(self, historical_data_dir):
        """
        Initialize LSTM Flood Prediction Model

        Args:
        - historical_data_dir: Directory with historical river flow data
        """
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

    def prepare_lstm_data(self, data, time_steps=3):
        """
        Prepare data for LSTM model

        Args:
        - data: Input DataFrame
        - time_steps: Number of previous time steps to use

        Returns:
```

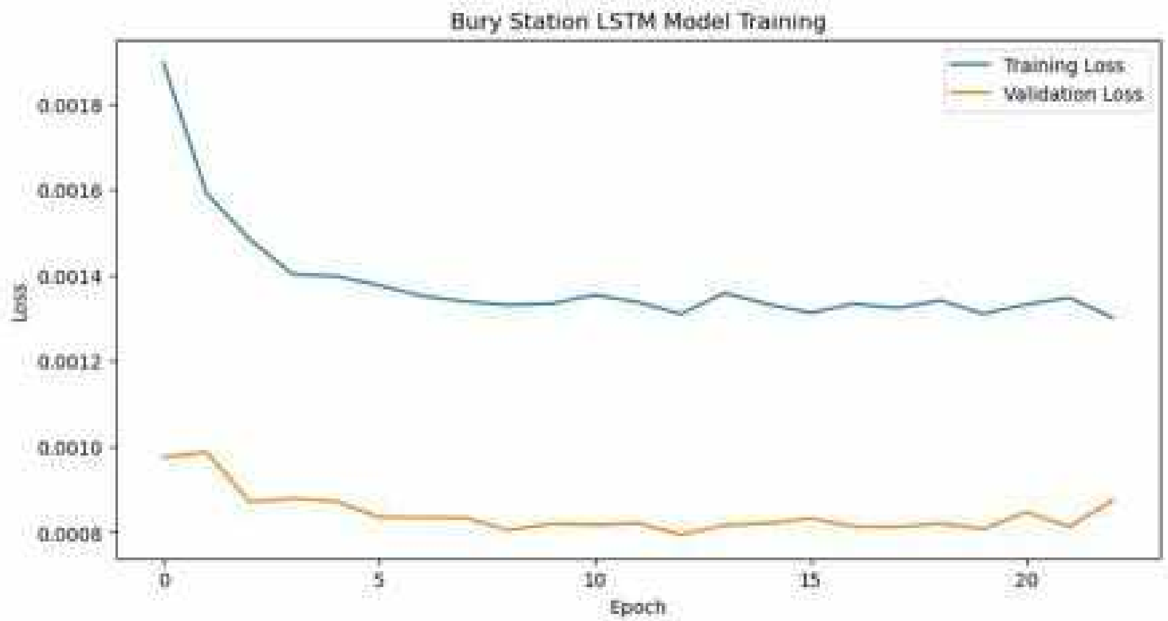
```
        - Scaled data
        - X (input sequences)
        - y (target values)
        """
        # Sort data by date
        data = data.sort_values('Date')

        # Scale the data
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_data = scaler.fit_transform(data[['Flow']])

        # Create sequences
        X, y = [], []
        for i in range(len(scaled_data) - time_steps):
            X.append(scaled_data[i:i+time_steps])
            y.append(scaled_data[i+time_steps])

        return scaler, np.array(X), np.array(y)

    def build_lstm_model(self, input_shape):
        """
        Build LSTM model architecture

        Args:
        - input_shape: Shape of input data

        Returns:
        - Compiled LSTM model
        """
        model = Sequential([
            LSTM(50, activation='relu', input_shape=input_shape, return_sequence
            Dropout(0.2),
            LSTM(50, activation='relu'),
            Dropout(0.2),
            Dense(25, activation='relu'),
            Dense(1)
        ])

        model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
        return model

    def train_lstm_model(self, station='Bury'):
        """
        Train LSTM model for specified station

        Args:
        - station: 'Bury' or 'Rochdale'

        Returns:
        - Trained model
        - Scaler
        - Training history
        """
        # Select appropriate dataset
        data = self.bury_flow if station == 'Bury' else self.rochdale_flow

        # Prepare data
        scaler, X, y = self.prepare_lstm_data(data)

        # Split data
```

```
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Build model
        model = self.build_lstm_model(input_shape=(X_train.shape[1], X_train.sha

        # Early stopping
        early_stopping = EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )

        # Train model
        history = model.fit(
            X_train, y_train,
            epochs=100,
            batch_size=32,
            validation_split=0.2,
            callbacks=[early_stopping],
            verbose=0
        )

        # Evaluate model
        train_loss = model.evaluate(X_train, y_train, verbose=0)
        test_loss = model.evaluate(X_test, y_test, verbose=0)

        # Visualize training
        plt.figure(figsize=(10,5))
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title(f'{station} Station LSTM Model Training')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

        return {
            'model': model,
            'scaler': scaler,
            'train_loss': train_loss,
            'test_loss': test_loss
        }

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
lstm_predictor = LSTMFloodPredictor(historical_data_dir)

# Train LSTM models for both stations
stations = ['Bury', 'Rochdale']
lstm_models = {}

for station in stations:
    print(f"\n--- {station} Station LSTM Model ---")
    result = lstm_predictor.train_lstm_model(station)
    lstm_models[station] = result

# Save models
import joblib
```

```
for station, model_data in lstm_models.items():
    # Save model
    model_data['model'].save(
        f'C:/Users/Administrator/NEWPROJECT/models/{station}_lstm_model.h5'
    )
    # Save scaler
    joblib.dump(
        model_data['scaler'],
        f'C:/Users/Administrator/NEWPROJECT/models/{station}_lstm_scaler.joblib'
    )
```

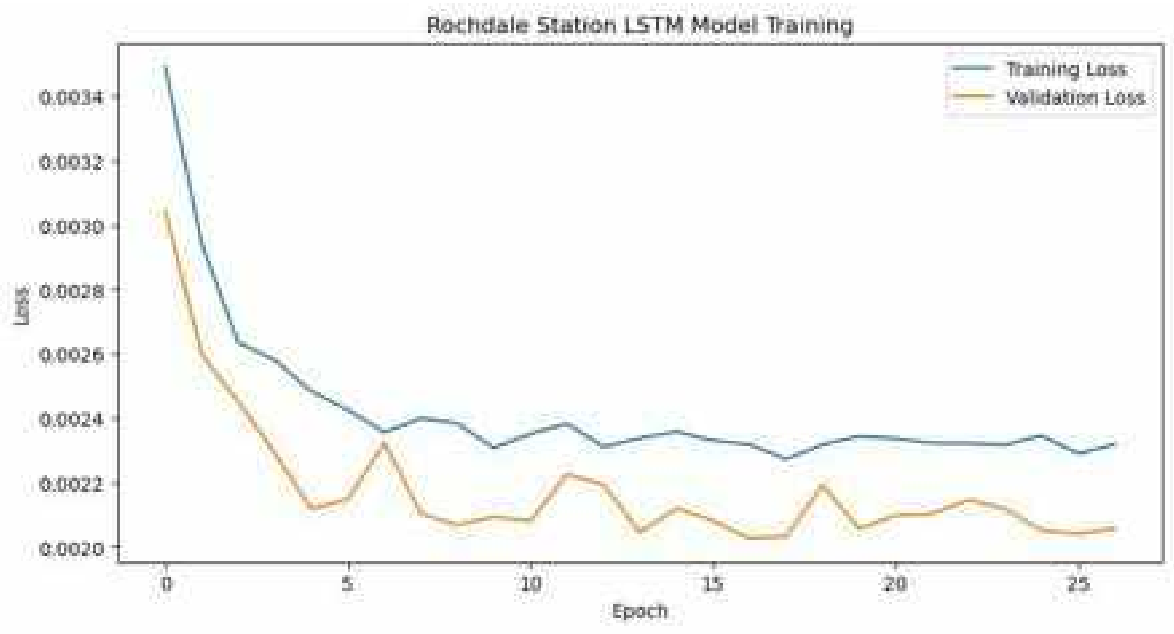
```
--- Bury Station LSTM Model ---
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```




```
--- Rochdale Station LSTM Model ---
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```




```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
```

performance evaluation comparing the LSTM and Random Forest models

In [10]:
```
import numpy as np
import pandas as pd
import joblib
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import os

def prepare_lstm_data(data, time_steps=3):
    """
    Prepare data for LSTM model

    Args:
    - data: Input DataFrame
    - time_steps: Number of previous time steps to use

    Returns:
    - Scaled data
    - X (input sequences)
    - y (target values)
    """
    # Sort data by date
    data = data.sort_values('Date')
```

```
    # Scale the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data[['Flow']])

    # Create sequences
    X, y = [], []
    for i in range(len(scaled_data) - time_steps):
        X.append(scaled_data[i:i+time_steps])
        y.append(scaled_data[i+time_steps])

    return scaler, np.array(X), np.array(y)

def build_lstm_model(input_shape):
    """
    Build LSTM model architecture

    Args:
    - input_shape: Shape of input data

    Returns:
    - Compiled LSTM model
    """
    model = Sequential([
        LSTM(50, activation='relu', input_shape=input_shape, return_sequences=Tr
        Dropout(0.2),
        LSTM(50, activation='relu'),
        Dropout(0.2),
        Dense(25, activation='relu'),
        Dense(1)
    ])

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error'
    return model

def train_lstm_model(historical_data_dir, station='Bury'):
    """
    Train LSTM model for specified station

    Args:
    - historical_data_dir: Directory with historical data
    - station: 'Bury' or 'Rochdale'

    Returns:
    - Trained model
    - Scaler
    """
    # Load historical data
    data = pd.read_csv(f'{historical_data_dir}/{"bury" if station == "Bury" else
    data['Date'] = pd.to_datetime(data['Date'])

    # Prepare data
    scaler, X, y = prepare_lstm_data(data)

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Build model
    model = build_lstm_model(input_shape=(X_train.shape[1], X_train.shape[2]))
```

```
    # Early stopping
    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )

    # Train model
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_split=0.2,
        callbacks=[early_stopping],
        verbose=1
    )

    # Save model weights
    save_dir = 'C:/Users/Administrator/NEWPROJECT/models'
    os.makedirs(save_dir, exist_ok=True)  # Ensure directory exists

    # Modify weight saving
    weights_filename = f'{station.lower()}_lstm_model.weights.h5'
    weights_path = os.path.join(save_dir, weights_filename)
    model.save_weights(weights_path)

    # Save full model (alternative method)
    model_path = os.path.join(save_dir, f'{station.lower()}_lstm_model.h5')
    model.save(model_path)

    # Save scaler
    scaler_path = os.path.join(save_dir, f'{station.lower()}_lstm_scaler.joblib'
    joblib.dump(scaler, scaler_path)

    return model, scaler

# Train models for both stations
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
stations = ['Bury', 'Rochdale']

for station in stations:
    print(f"\n--- Training {station} Station LSTM Model ---")
    model, scaler = train_lstm_model(historical_data_dir, station)
    print(f"{station} Station LSTM Model Training Complete")
```

```
--- Training Bury Station LSTM Model ---
Epoch 1/100
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```

```
199/199 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 0.0022 - val_loss: 0.0011
Epoch 2/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0015 - val_loss: 0.0011
Epoch 3/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0016 - val_loss: 8.6522e-04
Epoch 4/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 8.4305e-04
Epoch 5/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0014 - val_loss: 8.3874e-04
Epoch 6/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 8.9343e-04
Epoch 7/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0014 - val_loss: 9.5595e-04
Epoch 8/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0014 - val_loss: 8.5222e-04
Epoch 9/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0012 - val_loss: 8.4739e-04
Epoch 10/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0012 - val_loss: 8.3504e-04
Epoch 11/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0012 - val_loss: 8.5106e-04
Epoch 12/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 8.8844e-04
Epoch 13/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 8.3894e-04
Epoch 14/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0015 - val_loss: 8.4998e-04
Epoch 15/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0012 - val_loss: 8.4541e-04
Epoch 16/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 9.0541e-04
Epoch 17/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0014 - val_loss: 8.6199e-04
Epoch 18/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0013 - val_loss: 9.1045e-04
Epoch 19/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0014 - val_loss: 8.0750e-04
Epoch 20/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0013 - val_loss: 8.6715e-04
Epoch 21/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0013 - val_loss: 8.1404e-04
Epoch 22/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0013 - val_loss: 8.5417e-04
Epoch 23/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0016 - val_loss: 8.8010e-04
Epoch 24/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0015 - val_loss: 8.1426e-04
Epoch 25/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0012 - val_loss: 9.1169e-04
Epoch 26/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0014 - val_loss: 8.5936e-04
Epoch 27/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0012 - val_loss: 8.2556e-04
Epoch 28/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0011 - val_loss: 8.4032e-04
Epoch 29/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0012 - val_loss: 8.5779e-04
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
Bury Station LSTM Model Training Complete

--- Training Rochdale Station LSTM Model ---
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 0.0050 - val_loss: 0.0028
Epoch 2/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0030 - val_loss: 0.0025
Epoch 3/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0028 - val_loss: 0.0023
Epoch 4/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0024 - val_loss: 0.0022
Epoch 5/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 6/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0024 - val_loss: 0.0021
Epoch 7/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 8/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0027 - val_loss: 0.0021
Epoch 9/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0020 - val_loss: 0.0021
Epoch 10/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 11/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0024 - val_loss: 0.0021
Epoch 12/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0021 - val_loss: 0.0020
Epoch 13/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 14/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0026 - val_loss: 0.0020
Epoch 15/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 16/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0020 - val_loss: 0.0020
Epoch 17/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 18/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 19/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 20/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0024 - val_loss: 0.0020
Epoch 21/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 22/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 23/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0023 - val_loss: 0.0020
Epoch 24/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0024 - val_loss: 0.0020
Epoch 25/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0025 - val_loss: 0.0020
Epoch 26/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 27/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 28/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0021 - val_loss: 0.0020
Epoch 29/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.0025 - val_loss: 0.0021
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
Rochdale Station LSTM Model Training Complete
```

In [13]:

```
import os
import glob
import numpy as np
import pandas as pd
import joblib
import tensorflow as tf
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# First, let's list the files in the models directory
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'
print("Files in models directory:")
for file in os.listdir(models_dir):
    print(file)
```

```
Files in models directory:
Bury_lstm_model.h5
bury_lstm_model.weights.h5
Bury_lstm_scaler.joblib
Rochdale_lstm_model.h5
rochdale_lstm_model.weights.h5
Rochdale_lstm_scaler.joblib
```

In [14]:

```
import os
import numpy as np
import pandas as pd
import joblib
import tensorflow as tf
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

class ModelPerformanceEvaluator:
    def __init__(self, historical_data_dir, models_dir):
        """
        Initialize performance evaluator

        Args:
        - historical_data_dir: Directory with historical data
        - models_dir: Directory with trained models
        """
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

        # Set models directory
        self.models_dir = models_dir

    def prepare_lstm_data(self, data, station, time_steps=3):
        """
        Prepare data for LSTM model evaluation
```

```
        Args:
        - data: Input DataFrame
        - station: Station name
        - time_steps: Number of previous time steps to use

        Returns:
        - Scaled data
        - X (input sequences)
        - y (target values)
        """
        # Sort data by date
        data = data.sort_values('Date')

        # Load scaler (handle different filename variations)
        scaler_filename = [f for f in os.listdir(self.models_dir) if station.low
        scaler = joblib.load(os.path.join(self.models_dir, scaler_filename))

        scaled_data = scaler.transform(data[['Flow']])

        # Create sequences
        X, y = [], []
        for i in range(len(scaled_data) - time_steps):
            X.append(scaled_data[i:i+time_steps])
            y.append(scaled_data[i+time_steps])

        return scaler, np.array(X), np.array(y)

    def rebuild_lstm_model(self, input_shape=(3, 1)):
        """
        Rebuild LSTM model architecture

        Args:
        - input_shape: Shape of input data

        Returns:
        - Compiled LSTM model
        """
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import LSTM, Dense, Dropout
        from tensorflow.keras.optimizers import Adam

        model = Sequential([
            LSTM(50, activation='relu', input_shape=input_shape, return_sequence
            Dropout(0.2),
            LSTM(50, activation='relu'),
            Dropout(0.2),
            Dense(25, activation='relu'),
            Dense(1)
        ])

        model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_er
        return model

    def evaluate_lstm_model(self, station='Bury'):
        """
        Evaluate LSTM model performance

        Args:
        - station: 'Bury' or 'Rochdale'
```

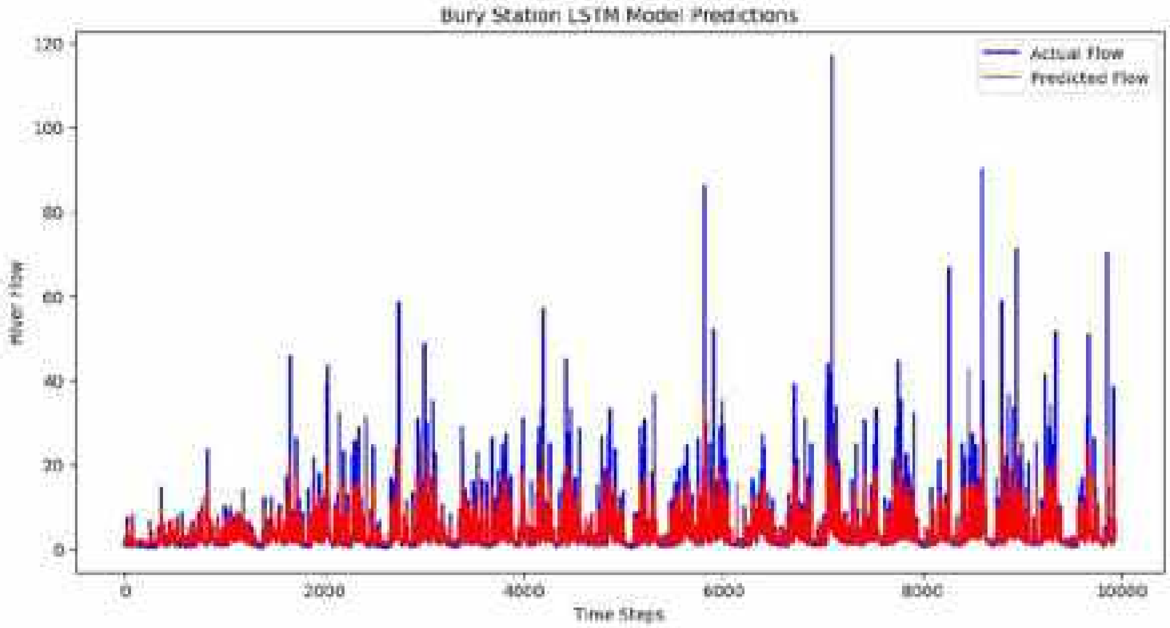
```
        Returns:
        - Performance metrics dictionary
        """
        # Select appropriate dataset
        data = self.bury_flow if station == 'Bury' else self.rochdale_flow

        # Prepare data
        scaler, X, y = self.prepare_lstm_data(data, station)

        # Rebuild model architecture
        model = self.rebuild_lstm_model(input_shape=(X.shape[1], X.shape[2]))

        # Load weights (handle different filename variations)
        weights_filename = [f for f in os.listdir(self.models_dir) if station.lc
        weights_path = os.path.join(self.models_dir, weights_filename)
        model.load_weights(weights_path)

        # Predict
        y_pred_scaled = model.predict(X)

        # Inverse transform predictions and actual values
        y_pred = scaler.inverse_transform(y_pred_scaled)
        y_actual = scaler.inverse_transform(y)

        # Calculate performance metrics
        r2 = r2_score(y_actual, y_pred)
        rmse = np.sqrt(mean_squared_error(y_actual, y_pred))
        mae = mean_absolute_error(y_actual, y_pred)

        # Visualization
        plt.figure(figsize=(12,6))
        plt.plot(y_actual, label='Actual Flow', color='blue')
        plt.plot(y_pred, label='Predicted Flow', color='red')
        plt.title(f'{station} Station LSTM Model Predictions')
        plt.xlabel('Time Steps')
        plt.ylabel('River Flow')
        plt.legend()
        plt.show()

        return {
            'r2_score': r2,
            'rmse': rmse,
            'mae': mae
        }

    def compare_with_random_forest(self, station='Bury'):
        """
        Compare LSTM with Random Forest model

        Args:
        - station: 'Bury' or 'Rochdale'

        Returns:
        - Comparative performance metrics
        """
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
```

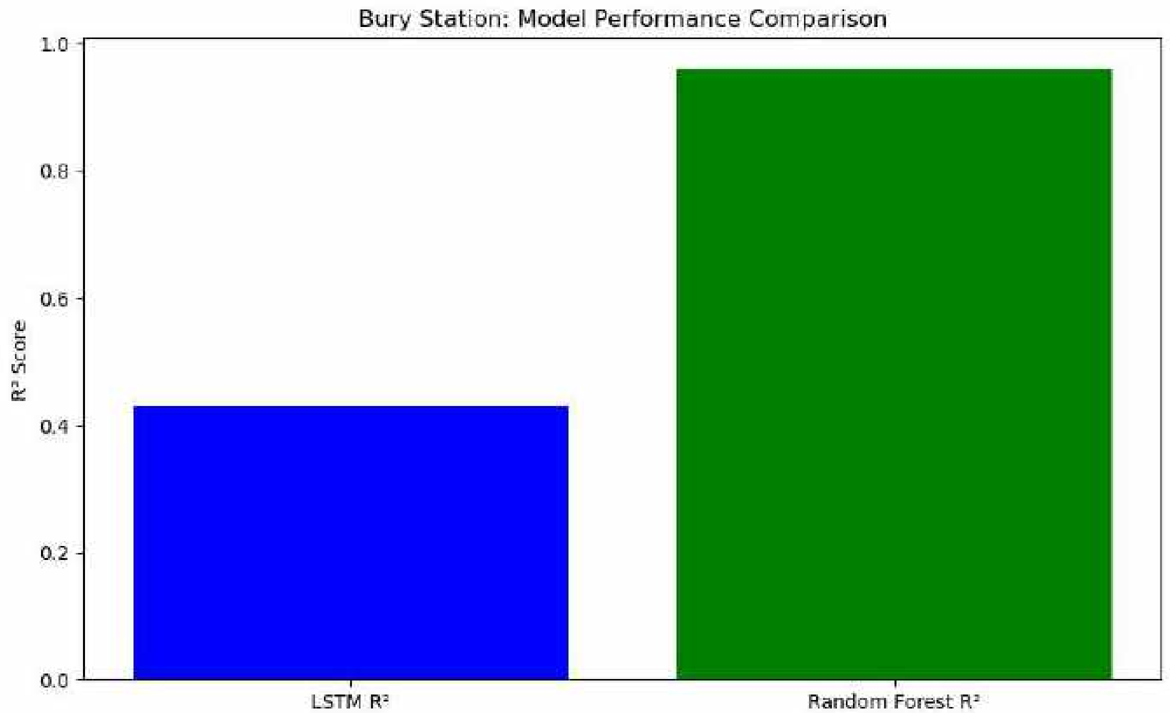
```
# Load advanced features
advanced_features_path = 'C:/Users/Administrator/NEWPROJECT/cleaned_data
advanced_features = pd.read_csv(advanced_features_path)

# Prepare data for Random Forest
if station == 'Bury':
    features_columns = [
        'flow_rolling_mean_3d_bury',
        'flow_rolling_std_3d_bury',
        'month_bury',
        'day_of_week_bury',
        'is_weekend_bury',
        'seasonal_trend_bury',
        'seasonal_cycle_bury',
        'station_temperature_bury',
        'temp_anomaly_bury',
        'flow_difference',
        'flow_ratio',
        'bury_flow_lag1'
    ]
    target_column = 'Flow_bury'
else:
    features_columns = [
        'flow_rolling_mean_3d_rochdale',
        'flow_rolling_std_3d_rochdale',
        'month_rochdale',
        'day_of_week_rochdale',
        'is_weekend_rochdale',
        'seasonal_trend_rochdale',
        'seasonal_cycle_rochdale',
        'station_temperature_rochdale',
        'temp_anomaly_rochdale',
        'flow_difference',
        'flow_ratio',
        'rochdale_flow_lag1'
    ]
    target_column = 'Flow_rochdale'

# Remove rows with NaN
df_clean = advanced_features.dropna(subset=features_columns + [target_cc

# Prepare features and target
X = df_clean[features_columns]
y = df_clean[target_column]

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Random Forest
rf_model = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
```

```
            max_depth=10
        )
        rf_model.fit(X_train_scaled, y_train)

        # Predict
        y_pred_rf = rf_model.predict(X_test_scaled)

        # Calculate performance metrics
        rf_r2 = r2_score(y_test, y_pred_rf)
        rf_rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
        rf_mae = mean_absolute_error(y_test, y_pred_rf)

        # LSTM Performance (for comparison)
        lstm_performance = self.evaluate_lstm_model(station)

        # Comparative Visualization
        plt.figure(figsize=(10,6))
        plt.bar(['LSTM R²', 'Random Forest R²'],
                [lstm_performance['r2_score'], rf_r2],
                color=['blue', 'green'])
        plt.title(f'{station} Station: Model Performance Comparison')
        plt.ylabel('R² Score')
        plt.show()

        return {
            'LSTM': lstm_performance,
            'Random Forest': {
                'r2_score': rf_r2,
                'rmse': rf_rmse,
                'mae': rf_mae
            }
        }

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'

performance_evaluator = ModelPerformanceEvaluator(historical_data_dir, models_di

# Evaluate both stations
stations = ['Bury', 'Rochdale']
comparative_results = {}

for station in stations:
    print(f"\n--- {station} Station Model Comparison ---")
    comparative_results[station] = performance_evaluator.compare_with_random_for

# Print detailed results
for station, results in comparative_results.items():
    print(f"\n{station} Station Performance:")
    print("LSTM Model:")
    for metric, value in results['LSTM'].items():
        print(f"  {metric}: {value}")
    print("\nRandom Forest Model:")
    for metric, value in results['Random Forest'].items():
        print(f"  {metric}: {value}")
```

```
--- Bury Station Model Comparison ---
```

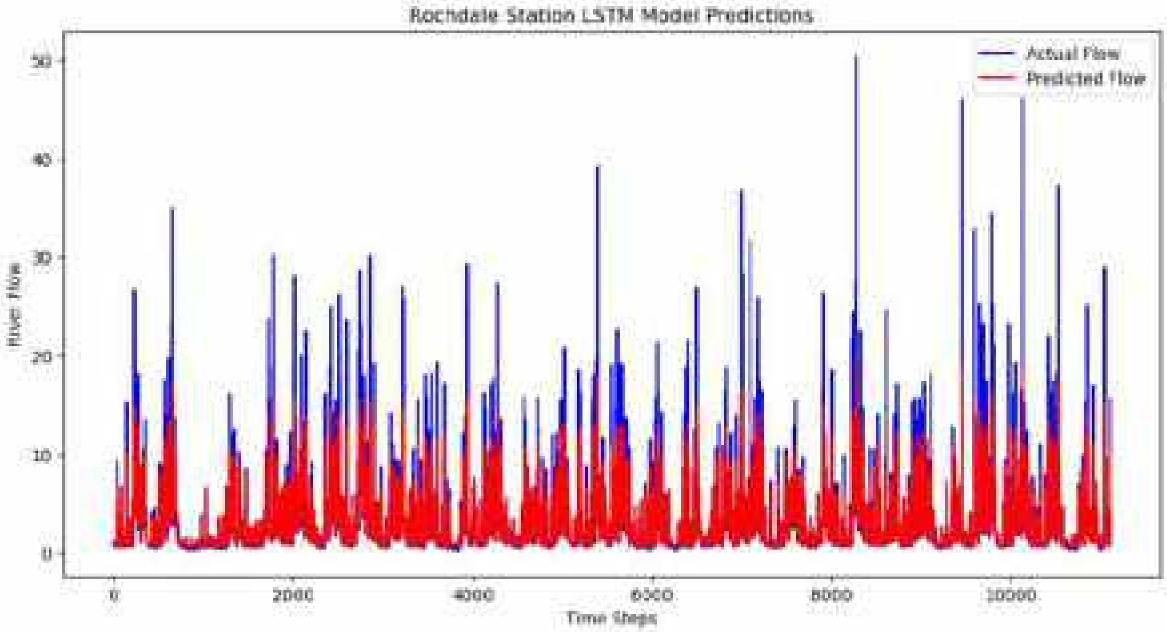
```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\saving\saving_lib.p
y:757: UserWarning: Skipping variable loading for optimizer 'adam', because it ha
s 2 variables whereas the saved optimizer has 22 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
311/311 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```

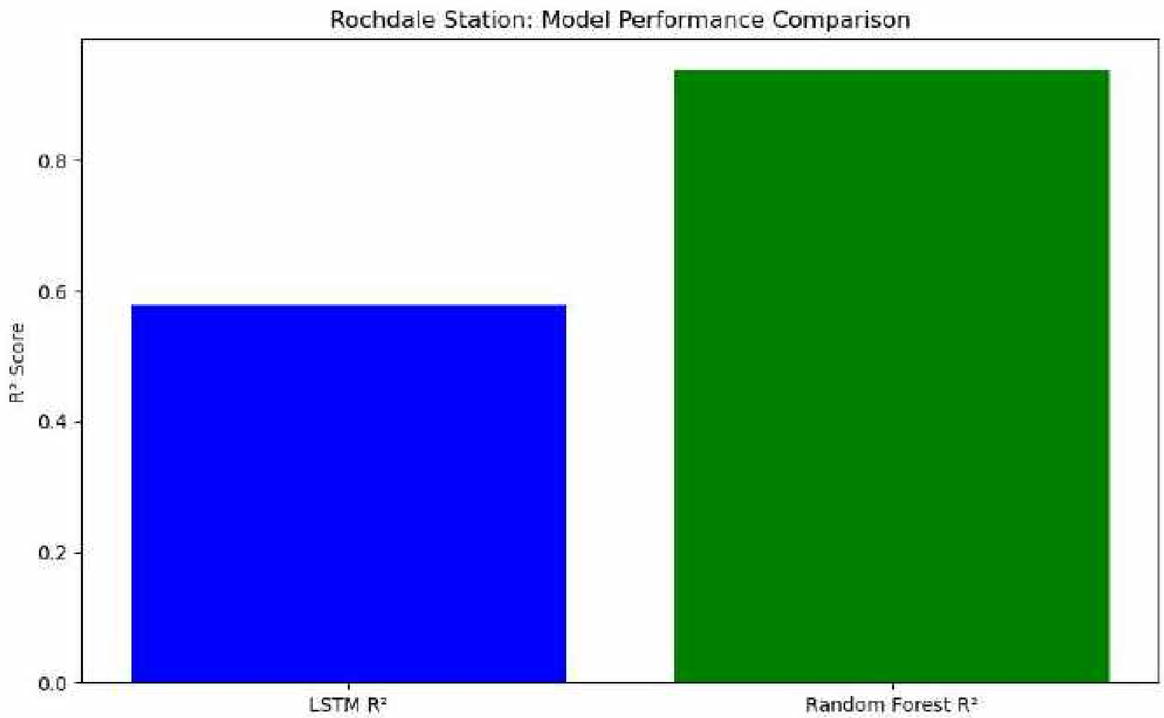





```
--- Rochdale Station Model Comparison ---
```

```
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\saving\saving_lib.p
y:757: UserWarning: Skipping variable loading for optimizer 'adam', because it ha
s 2 variables whereas the saved optimizer has 22 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
348/348 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```





```
Bury Station Performance:
LSTM Model:
  r2_score: 0.43012364312944895
  rmse: 4.073211281343852
  mae: 1.6775296790101366

Random Forest Model:
  r2_score: 0.9602407079088583
  rmse: 0.9682276912700253
  mae: 0.37402047937494876

Rochdale Station Performance:
LSTM Model:
  r2_score: 0.5796951852107592
  rmse: 2.2995102709668447
  mae: 1.0710644795921433

Random Forest Model:
  r2_score: 0.9385754910963229
  rmse: 0.8115275901215016
  mae: 0.34723934043211135
```

In [15]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

class EnhancedLSTMModel:
    def __init__(self, historical_data_dir):
        """
        Initialize Enhanced LSTM Model

        Args:
        - historical_data_dir: Directory containing historical data
        """
        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

        # Initialize scalers
        self.scalers = {}

    def prepare_sequences(self, data, sequence_length=5):
        """
        Prepare sequences for LSTM with longer sequence length

        Args:
        - data: Input DataFrame
        - sequence_length: Number of time steps to use

        Returns:
```

```
        - X: Input sequences
        - y: Target values
        """
        # Scale the data
        scaler = MinMaxScaler(feature_range=(0, 1))
        flow_scaled = scaler.fit_transform(data[['Flow']])
        self.scalers[data['Station'].iloc[0]] = scaler

        # Create sequences
        X, y = [], []
        for i in range(len(flow_scaled) - sequence_length):
            X.append(flow_scaled[i:(i + sequence_length)])
            y.append(flow_scaled[i + sequence_length])

        return np.array(X), np.array(y)

    def build_enhanced_model(self, sequence_length):
        """
        Build enhanced LSTM model with improved architecture

        Args:
        - sequence_length: Length of input sequences

        Returns:
        - Compiled model
        """
        model = Sequential([
            Input(shape=(sequence_length, 1)),
            LSTM(64, return_sequences=True),
            Dropout(0.2),
            LSTM(32, return_sequences=True),
            Dropout(0.2),
            LSTM(16),
            Dense(8, activation='relu'),
            Dense(1)
        ])

        model.compile(
            optimizer=Adam(learning_rate=0.001),
            loss='mse',
            metrics=['mae']
        )

        return model

    def train_station_model(self, station_data, sequence_length=5, epochs=100):
        """
        Train LSTM model for a specific station

        Args:
        - station_data: DataFrame containing station data
        - sequence_length: Length of input sequences
        - epochs: Number of training epochs

        Returns:
        - Trained model
        - Training history
        """
        # Prepare sequences
        X, y = self.prepare_sequences(station_data, sequence_length)
```

```
        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Build and train model
        model = self.build_enhanced_model(sequence_length)

        history = model.fit(
            X_train, y_train,
            epochs=epochs,
            batch_size=32,
            validation_split=0.2,
            verbose=1
        )

        # Evaluate model
        train_loss = model.evaluate(X_train, y_train, verbose=0)
        test_loss = model.evaluate(X_test, y_test, verbose=0)

        print(f"\nTrain Loss: {train_loss[0]:.4f}")
        print(f"Test Loss: {test_loss[0]:.4f}")

        # Plot training history
        plt.figure(figsize=(10, 6))
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title(f'Model Loss - {station_data["Station"].iloc[0]}')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

        return model, history

    def train_all_stations(self):
        """
        Train models for all stations

        Returns:
        - Dictionary of trained models
        """
        models = {}

        # Add station identifier
        self.bury_flow['Station'] = 'Bury Ground'
        self.rochdale_flow['Station'] = 'Rochdale'

        # Train for each station
        for station_data in [self.bury_flow, self.rochdale_flow]:
            station_name = station_data['Station'].iloc[0]
            print(f"\nTraining model for {station_name}")

            model, history = self.train_station_model(station_data)
            models[station_name] = {
                'model': model,
                'history': history
            }
```

```
        return models

# Example usage
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
lstm_trainer = EnhancedLSTMModel(historical_data_dir)
trained_models = lstm_trainer.train_all_stations()
```

```
Training model for Bury Ground
Epoch 1/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - loss: 0.0018 - mae: 0.0240 - val_los
s: 0.0015 - val_mae: 0.0211
Epoch 2/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0014 - mae: 0.0194 - val_loss:
0.0015 - val_mae: 0.0164
Epoch 3/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0014 - mae: 0.0191 - val_los
s: 0.0013 - val_mae: 0.0161
Epoch 4/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0012 - mae: 0.0169 - val_loss:
0.0011 - val_mae: 0.0187
Epoch 5/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0012 - mae: 0.0167 - val_loss:
0.0011 - val_mae: 0.0149
Epoch 6/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0013 - mae: 0.0165 - val_loss:
0.0011 - val_mae: 0.0217
Epoch 7/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0170 - val_loss:
0.0011 - val_mae: 0.0155
Epoch 8/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0014 - mae: 0.0175 - val_loss:
0.0011 - val_mae: 0.0142
Epoch 9/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0010 - mae: 0.0153 - val_loss:
0.0012 - val_mae: 0.0147
Epoch 10/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0013 - mae: 0.0164 - val_los
s: 0.0011 - val_mae: 0.0170
Epoch 11/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0153 - val_loss:
0.0011 - val_mae: 0.0162
Epoch 12/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0164 - val_loss:
0.0011 - val_mae: 0.0170
Epoch 13/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0010 - mae: 0.0161 - val_loss:
0.0011 - val_mae: 0.0201
Epoch 14/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0163 - val_loss:
0.0011 - val_mae: 0.0202
Epoch 15/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 9.5987e-04 - mae: 0.0156 - val_l
oss: 0.0011 - val_mae: 0.0183
Epoch 16/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 9.9822e-04 - mae: 0.0160 - val_l
oss: 0.0011 - val_mae: 0.0162
Epoch 17/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0016 - mae: 0.0180 - val_loss:
0.0011 - val_mae: 0.0181
Epoch 18/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0012 - mae: 0.0165 - val_loss:
0.0011 - val_mae: 0.0177
Epoch 19/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0163 - val_loss:
0.0011 - val_mae: 0.0144
Epoch 20/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0152 - val_loss:
```

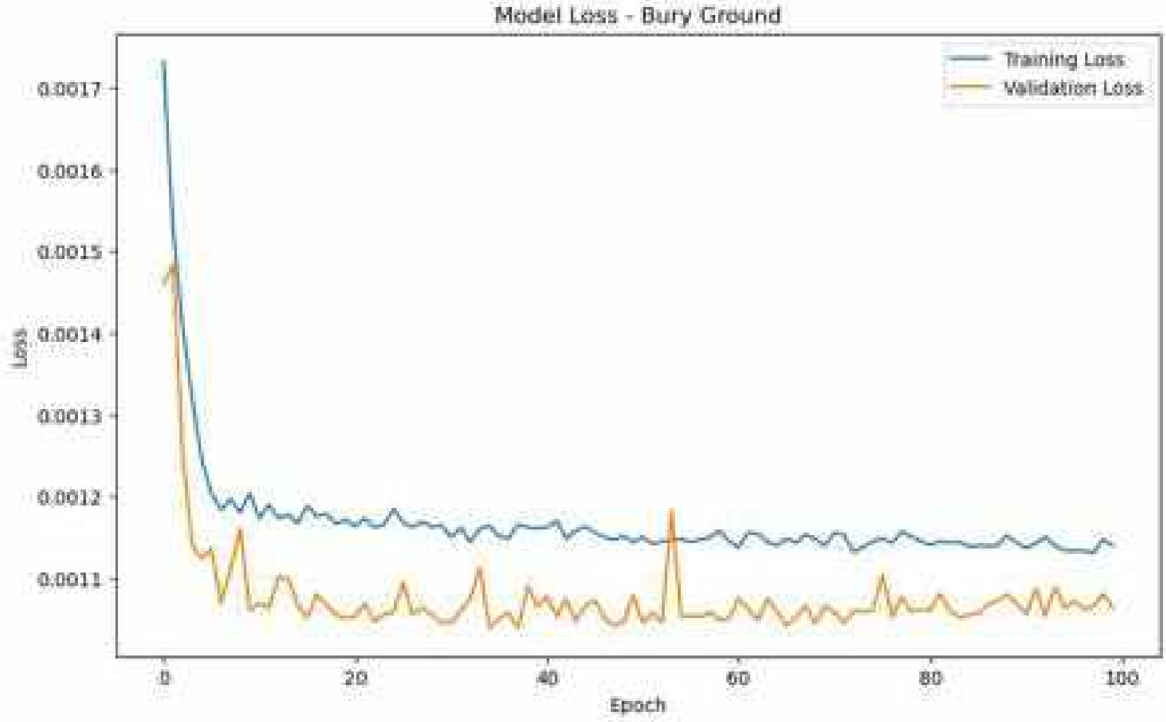
```
0.0011 - val_mae: 0.0165
Epoch 21/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0156
Epoch 22/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0158 - val_loss:
0.0011 - val_mae: 0.0140
Epoch 23/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0011 - mae: 0.0156 - val_loss:
0.0010 - val_mae: 0.0153
Epoch 24/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0156 - val_loss:
0.0011 - val_mae: 0.0174
Epoch 25/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0013 - mae: 0.0169 - val_loss:
0.0011 - val_mae: 0.0171
Epoch 26/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0162 - val_loss:
0.0011 - val_mae: 0.0203
Epoch 27/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0149
Epoch 28/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0016 - mae: 0.0174 - val_loss:
0.0011 - val_mae: 0.0182
Epoch 29/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0013 - mae: 0.0167 - val_loss:
0.0011 - val_mae: 0.0179
Epoch 30/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0012 - mae: 0.0161 - val_loss:
0.0010 - val_mae: 0.0168
Epoch 31/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0010 - mae: 0.0154 - val_loss:
0.0010 - val_mae: 0.0157
Epoch 32/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 9.9445e-04 - mae: 0.0152 - val_l
oss: 0.0011 - val_mae: 0.0165
Epoch 33/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0013 - mae: 0.0167 - val_loss:
0.0011 - val_mae: 0.0145
Epoch 34/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0012 - mae: 0.0163 - val_loss:
0.0011 - val_mae: 0.0212
Epoch 35/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0160 - val_loss:
0.0010 - val_mae: 0.0157
Epoch 36/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0010 - mae: 0.0153 - val_loss:
0.0011 - val_mae: 0.0158
Epoch 37/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0160 - val_loss:
0.0011 - val_mae: 0.0174
Epoch 38/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0010 - val_mae: 0.0155
Epoch 39/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0159 - val_loss:
0.0011 - val_mae: 0.0141
Epoch 40/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0013 - mae: 0.0160 - val_loss:
```

```
0.0011 - val_mae: 0.0142
Epoch 41/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0012 - mae: 0.0161 - val_loss:
0.0011 - val_mae: 0.0194
Epoch 42/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0011 - mae: 0.0159 - val_loss:
0.0011 - val_mae: 0.0170
Epoch 43/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0012 - mae: 0.0156 - val_loss:
0.0011 - val_mae: 0.0180
Epoch 44/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0159 - val_loss:
0.0010 - val_mae: 0.0146
Epoch 45/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0014 - mae: 0.0161 - val_loss:
0.0011 - val_mae: 0.0144
Epoch 46/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0140
Epoch 47/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0150 - val_loss:
0.0011 - val_mae: 0.0158
Epoch 48/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0011 - mae: 0.0161 - val_loss:
0.0010 - val_mae: 0.0166
Epoch 49/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0154 - val_loss:
0.0010 - val_mae: 0.0159
Epoch 50/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0195
Epoch 51/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0164 - val_loss:
0.0010 - val_mae: 0.0162
Epoch 52/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0148
Epoch 53/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0012 - mae: 0.0161 - val_loss:
0.0010 - val_mae: 0.0149
Epoch 54/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0012 - mae: 0.0160 - val_loss:
0.0012 - val_mae: 0.0234
Epoch 55/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0011 - mae: 0.0162 - val_los
s: 0.0011 - val_mae: 0.0147
Epoch 56/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0143
Epoch 57/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0157
Epoch 58/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0173
Epoch 59/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0157 - val_loss:
0.0010 - val_mae: 0.0165
Epoch 60/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0151 - val_loss:
```

```
0.0011 - val_mae: 0.0173
Epoch 61/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0014 - mae: 0.0169 - val_loss:
0.0011 - val_mae: 0.0145
Epoch 62/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0154 - val_loss:
0.0011 - val_mae: 0.0178
Epoch 63/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0162 - val_loss:
0.0010 - val_mae: 0.0167
Epoch 64/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0010 - mae: 0.0148 - val_loss:
0.0011 - val_mae: 0.0191
Epoch 65/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0013 - mae: 0.0160 - val_loss:
0.0011 - val_mae: 0.0148
Epoch 66/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0014 - mae: 0.0172 - val_los
s: 0.0010 - val_mae: 0.0149
Epoch 67/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 9.5730e-04 - mae: 0.0148 - val_l
oss: 0.0011 - val_mae: 0.0163
Epoch 68/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0158 - val_loss:
0.0011 - val_mae: 0.0146
Epoch 69/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0013 - mae: 0.0161 - val_loss:
0.0010 - val_mae: 0.0151
Epoch 70/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0164
Epoch 71/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0158 - val_loss:
0.0011 - val_mae: 0.0168
Epoch 72/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0150 - val_loss:
0.0010 - val_mae: 0.0164
Epoch 73/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0012 - mae: 0.0164 - val_loss:
0.0011 - val_mae: 0.0144
Epoch 74/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0012 - mae: 0.0151 - val_los
s: 0.0011 - val_mae: 0.0158
Epoch 75/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0150
Epoch 76/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0013 - mae: 0.0161 - val_loss:
0.0011 - val_mae: 0.0144
Epoch 77/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0157 - val_loss:
0.0011 - val_mae: 0.0148
Epoch 78/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0141
Epoch 79/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0151 - val_loss:
0.0011 - val_mae: 0.0143
Epoch 80/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0014 - mae: 0.0159 - val_loss:
```

```
0.0011 - val_mae: 0.0148
Epoch 81/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0010 - mae: 0.0153 - val_loss:
0.0011 - val_mae: 0.0154
Epoch 82/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0010 - mae: 0.0150 - val_loss:
0.0011 - val_mae: 0.0188
Epoch 83/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0012 - mae: 0.0158 - val_los
s: 0.0011 - val_mae: 0.0159
Epoch 84/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0150 - val_loss:
0.0011 - val_mae: 0.0149
Epoch 85/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0010 - mae: 0.0149 - val_loss:
0.0011 - val_mae: 0.0157
Epoch 86/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0154 - val_loss:
0.0011 - val_mae: 0.0160
Epoch 87/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0149 - val_loss:
0.0011 - val_mae: 0.0140
Epoch 88/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0162 - val_loss:
0.0011 - val_mae: 0.0175
Epoch 89/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0160 - val_loss:
0.0011 - val_mae: 0.0145
Epoch 90/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0155 - val_loss:
0.0011 - val_mae: 0.0142
Epoch 91/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0151 - val_loss:
0.0011 - val_mae: 0.0166
Epoch 92/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0011 - mae: 0.0156 - val_los
s: 0.0011 - val_mae: 0.0139
Epoch 93/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0013 - mae: 0.0163 - val_loss:
0.0011 - val_mae: 0.0154
Epoch 94/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0010 - mae: 0.0146 - val_loss:
0.0011 - val_mae: 0.0142
Epoch 95/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.8282e-04 - mae: 0.0142 - val_
loss: 0.0011 - val_mae: 0.0175
Epoch 96/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0010 - mae: 0.0152 - val_loss:
0.0011 - val_mae: 0.0143
Epoch 97/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0159 - val_loss:
0.0011 - val_mae: 0.0145
Epoch 98/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0156 - val_loss:
0.0011 - val_mae: 0.0147
Epoch 99/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0011 - mae: 0.0151 - val_loss:
0.0011 - val_mae: 0.0168
Epoch 100/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0012 - mae: 0.0156 - val_loss:
```

```
0.0011 - val_mae: 0.0170

Train Loss: 0.0011
Test Loss: 0.0016
```



```
Training model for Rochdale
Epoch 1/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - loss: 0.0041 - mae: 0.0368 - val_los
s: 0.0032 - val_mae: 0.0291
Epoch 2/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0027 - mae: 0.0281 - val_loss:
0.0027 - val_mae: 0.0299
Epoch 3/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0027 - mae: 0.0266 - val_loss:
0.0023 - val_mae: 0.0231
Epoch 4/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0231 - val_loss:
0.0023 - val_mae: 0.0282
Epoch 5/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0024 - mae: 0.0239 - val_loss:
0.0021 - val_mae: 0.0217
Epoch 6/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0233 - val_loss:
0.0021 - val_mae: 0.0234
Epoch 7/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0022 - mae: 0.0225 - val_los
s: 0.0021 - val_mae: 0.0219
Epoch 8/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0230 - val_loss:
0.0021 - val_mae: 0.0271
Epoch 9/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0023 - mae: 0.0238 - val_loss:
0.0022 - val_mae: 0.0205
Epoch 10/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0018 - mae: 0.0212 - val_loss:
0.0021 - val_mae: 0.0195
Epoch 11/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0024 - mae: 0.0226 - val_loss:
0.0020 - val_mae: 0.0217
Epoch 12/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0206
Epoch 13/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0025 - mae: 0.0236 - val_loss:
0.0020 - val_mae: 0.0208
Epoch 14/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0019 - mae: 0.0216 - val_loss:
0.0020 - val_mae: 0.0245
Epoch 15/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0229 - val_loss:
0.0021 - val_mae: 0.0234
Epoch 16/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0019 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0199
Epoch 17/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0222 - val_loss:
0.0020 - val_mae: 0.0236
Epoch 18/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0213 - val_los
s: 0.0020 - val_mae: 0.0217
Epoch 19/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0020 - mae: 0.0217 - val_los
s: 0.0020 - val_mae: 0.0196
Epoch 20/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0021 - mae: 0.0228 - val_loss:
```

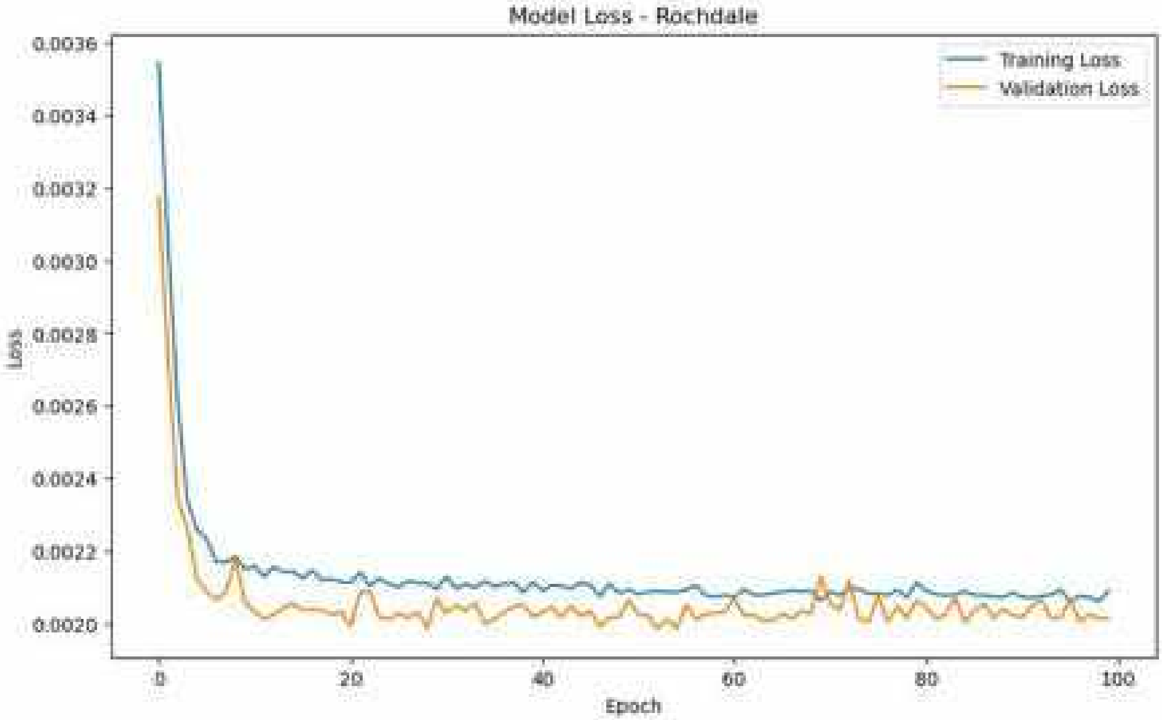
```
0.0020 - val_mae: 0.0247
Epoch 21/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0021 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0208
Epoch 22/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0020 - mae: 0.0219 - val_loss:
0.0021 - val_mae: 0.0217
Epoch 23/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0224 - val_loss:
0.0021 - val_mae: 0.0198
Epoch 24/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0019 - mae: 0.0214 - val_loss:
0.0020 - val_mae: 0.0246
Epoch 25/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0225
Epoch 26/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0217 - val_loss:
0.0020 - val_mae: 0.0219
Epoch 27/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0218 - val_loss:
0.0020 - val_mae: 0.0215
Epoch 28/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0024 - mae: 0.0226 - val_loss:
0.0020 - val_mae: 0.0200
Epoch 29/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0216 - val_loss:
0.0020 - val_mae: 0.0214
Epoch 30/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0023 - mae: 0.0229 - val_loss:
0.0021 - val_mae: 0.0194
Epoch 31/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0216 - val_loss:
0.0020 - val_mae: 0.0243
Epoch 32/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0226 - val_loss:
0.0021 - val_mae: 0.0241
Epoch 33/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0021 - mae: 0.0215 - val_los
s: 0.0020 - val_mae: 0.0206
Epoch 34/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0020 - mae: 0.0218 - val_los
s: 0.0021 - val_mae: 0.0200
Epoch 35/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0024 - mae: 0.0237 - val_loss:
0.0020 - val_mae: 0.0220
Epoch 36/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0221 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 37/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0221 - val_loss:
0.0020 - val_mae: 0.0219
Epoch 38/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0213 - val_loss:
0.0020 - val_mae: 0.0209
Epoch 39/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0018 - mae: 0.0204 - val_loss:
0.0021 - val_mae: 0.0200
Epoch 40/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0019 - mae: 0.0215 - val_loss:
```

```
0.0020 - val_mae: 0.0199
Epoch 41/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0020 - mae: 0.0213 - val_los
s: 0.0020 - val_mae: 0.0229
Epoch 42/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0222 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 43/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0025 - mae: 0.0226 - val_loss:
0.0020 - val_mae: 0.0203
Epoch 44/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0217 - val_loss:
0.0020 - val_mae: 0.0240
Epoch 45/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0225 - val_loss:
0.0020 - val_mae: 0.0200
Epoch 46/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0019 - mae: 0.0217 - val_loss:
0.0020 - val_mae: 0.0239
Epoch 47/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0225
Epoch 48/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0223 - val_loss:
0.0020 - val_mae: 0.0200
Epoch 49/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0021 - mae: 0.0211 - val_loss:
0.0020 - val_mae: 0.0219
Epoch 50/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0021 - mae: 0.0224 - val_los
s: 0.0021 - val_mae: 0.0202
Epoch 51/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 52/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020 - mae: 0.0219 - val_loss:
0.0020 - val_mae: 0.0196
Epoch 53/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0023 - mae: 0.0225 - val_loss:
0.0020 - val_mae: 0.0209
Epoch 54/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0018 - mae: 0.0207 - val_loss:
0.0020 - val_mae: 0.0208
Epoch 55/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0215
Epoch 56/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0224 - val_loss:
0.0021 - val_mae: 0.0195
Epoch 57/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0222 - val_loss:
0.0020 - val_mae: 0.0202
Epoch 58/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0198
Epoch 59/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0018 - mae: 0.0210 - val_loss:
0.0020 - val_mae: 0.0202
Epoch 60/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0216 - val_loss:
```

```
0.0020 - val_mae: 0.0200
Epoch 61/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0216 - val_loss:
0.0021 - val_mae: 0.0194
Epoch 62/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0210 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 63/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0221 - val_loss:
0.0020 - val_mae: 0.0216
Epoch 64/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0024 - mae: 0.0234 - val_loss:
0.0020 - val_mae: 0.0207
Epoch 65/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0021 - mae: 0.0219 - val_loss:
0.0020 - val_mae: 0.0206
Epoch 66/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0219 - val_loss:
0.0020 - val_mae: 0.0227
Epoch 67/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0213 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 68/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0019 - mae: 0.0209 - val_loss:
0.0020 - val_mae: 0.0203
Epoch 69/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0223 - val_loss:
0.0020 - val_mae: 0.0213
Epoch 70/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0222 - val_loss:
0.0021 - val_mae: 0.0191
Epoch 71/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0025 - mae: 0.0227 - val_loss:
0.0021 - val_mae: 0.0196
Epoch 72/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020 - mae: 0.0209 - val_loss:
0.0020 - val_mae: 0.0198
Epoch 73/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0018 - mae: 0.0205 - val_loss:
0.0021 - val_mae: 0.0252
Epoch 74/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0023 - mae: 0.0224 - val_loss:
0.0020 - val_mae: 0.0239
Epoch 75/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0019 - mae: 0.0212 - val_loss:
0.0020 - val_mae: 0.0216
Epoch 76/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0019 - mae: 0.0213 - val_loss:
0.0021 - val_mae: 0.0206
Epoch 77/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0023 - mae: 0.0218 - val_loss:
0.0020 - val_mae: 0.0220
Epoch 78/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0226 - val_loss:
0.0020 - val_mae: 0.0201
Epoch 79/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0019 - mae: 0.0207 - val_loss:
0.0020 - val_mae: 0.0237
Epoch 80/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0237 - val_loss:
```

```
0.0021 - val_mae: 0.0196
Epoch 81/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0243
Epoch 82/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0225 - val_loss:
0.0020 - val_mae: 0.0212
Epoch 83/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0214 - val_loss:
0.0020 - val_mae: 0.0205
Epoch 84/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0219 - val_loss:
0.0021 - val_mae: 0.0212
Epoch 85/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0017 - mae: 0.0206 - val_loss:
0.0020 - val_mae: 0.0217
Epoch 86/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0022 - mae: 0.0213 - val_loss:
0.0020 - val_mae: 0.0210
Epoch 87/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0023 - mae: 0.0220 - val_loss:
0.0021 - val_mae: 0.0193
Epoch 88/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0240
Epoch 89/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0222 - val_loss:
0.0020 - val_mae: 0.0236
Epoch 90/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0021 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0204
Epoch 91/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0217 - val_loss:
0.0020 - val_mae: 0.0231
Epoch 92/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0215 - val_loss:
0.0020 - val_mae: 0.0244
Epoch 93/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0020 - mae: 0.0210 - val_loss:
0.0021 - val_mae: 0.0258
Epoch 94/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0021 - mae: 0.0228 - val_loss:
0.0020 - val_mae: 0.0212
Epoch 95/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0022 - mae: 0.0222 - val_loss:
0.0020 - val_mae: 0.0227
Epoch 96/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0019 - mae: 0.0216 - val_loss:
0.0021 - val_mae: 0.0204
Epoch 97/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0021 - mae: 0.0228 - val_loss:
0.0020 - val_mae: 0.0223
Epoch 98/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020 - mae: 0.0217 - val_loss:
0.0020 - val_mae: 0.0219
Epoch 99/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0220 - val_loss:
0.0020 - val_mae: 0.0208
Epoch 100/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0022 - mae: 0.0226 - val_loss:
```

```
0.0020 - val_mae: 0.0215

Train Loss: 0.0020
Test Loss: 0.0023
```



In [17]:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
import os

def train_and_save_rf_models(historical_data_dir, models_dir):
    """
    Train and save Random Forest models for both stations
    """
    # Create models directory if it doesn't exist
    os.makedirs(models_dir, exist_ok=True)

    # Load historical data
    bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
    rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv'

    # Add basic features for each station
    for df, station in [(bury_flow, 'bury'), (rochdale_flow, 'rochdale')]:
        df['Date'] = pd.to_datetime(df['Date'])
        df[f'flow_rolling_mean_3d_{station}'] = df['Flow'].rolling(window=3).mea
        df[f'flow_rolling_std_3d_{station}'] = df['Flow'].rolling(window=3).std(
        df[f'month_{station}'] = df['Date'].dt.month
        df[f'day_of_week_{station}'] = df['Date'].dt.dayofweek
        df[f'is_weekend_{station}'] = df['Date'].dt.dayofweek.isin([5, 6]).astyp
        df[f'seasonal_trend_{station}'] = np.sin(df[f'month_{station}'] * (2 * n
        df[f'seasonal_cycle_{station}'] = np.cos(df[f'month_{station}'] * (2 * n
        df[f'station_temperature_{station}'] = 15 + 10 * np.sin((df[f'month_{sta
        df[f'temp_anomaly_{station}'] = df[f'station_temperature_{station}'] - d
        df[f'{station}_flow_lag1'] = df['Flow'].shift(1)
```

```
    # Add cross-station features
    merged_data = pd.merge(
        bury_flow,
        rochdale_flow,
        on='Date',
        suffixes=('_bury', '_rochdale')
    )
    merged_data['flow_difference'] = merged_data['Flow_bury'] - merged_data['Flo
    merged_data['flow_ratio'] = merged_data['Flow_bury'] / (merged_data['Flow_ro

    # Train Random Forest for each station
    for station in ['bury', 'rochdale']:
        # Prepare features
        feature_columns = [
            f'flow_rolling_mean_3d_{station}',
            f'flow_rolling_std_3d_{station}',
            f'month_{station}',
            f'day_of_week_{station}',
            f'is_weekend_{station}',
            f'seasonal_trend_{station}',
            f'seasonal_cycle_{station}',
            f'station_temperature_{station}',
            f'temp_anomaly_{station}',
            'flow_difference',
            'flow_ratio',
            f'{station}_flow_lag1'
        ]

        # Remove rows with NaN
        df_clean = merged_data.dropna(subset=feature_columns + [f'Flow_{station}

        # Split features and target
        X = df_clean[feature_columns]
        y = df_clean[f'Flow_{station}']

        # Train test split
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train Random Forest
        rf_model = RandomForestRegressor(
            n_estimators=100,
            random_state=42,
            max_depth=10
        )
        rf_model.fit(X_train_scaled, y_train)

        # Save model and scaler
        joblib.dump(rf_model, f'{models_dir}/{station}_rf_model.joblib')
        joblib.dump(scaler, f'{models_dir}/{station}_rf_scaler.joblib')

        print(f"\nSaved Random Forest model for {station} station")

# Train and save models
```

```
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'

train_and_save_rf_models(historical_data_dir, models_dir)
```

```
Saved Random Forest model for bury station

Saved Random Forest model for rochdale station
```

In [18]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from tensorflow.keras.models import load_model
import joblib

class EnsembleFloodPredictor:
    def __init__(self, models_dir):
        """Initialize ensemble predictor"""
        self.models_dir = models_dir
        self.load_models()

    def load_models(self):
        """Load all trained models and scalers"""
        # Load RF models and scalers
        self.rf_models = {}
        self.rf_scalers = {}
        for station in ['bury', 'rochdale']:
            self.rf_models[station] = joblib.load(
                f'{self.models_dir}/{station}_rf_model.joblib'
            )
            self.rf_scalers[station] = joblib.load(
                f'{self.models_dir}/{station}_rf_scaler.joblib'
            )

        # Load LSTM models
        self.lstm_models = {}
        for station in ['bury', 'rochdale']:
            model_path = f'{self.models_dir}/{station}_lstm_model.h5'
            self.lstm_models[station] = load_model(model_path)

    def predict_rf(self, data, station):
        """Make Random Forest prediction"""
        # Scale features
        scaled_features = self.rf_scalers[station].transform(data)
        # Make prediction
        return self.rf_models[station].predict(scaled_features)[0]

    def predict_lstm(self, data, station, sequence_length=5):
        """Make LSTM prediction"""
        # Prepare sequence
        sequence = data[-sequence_length:].reshape(1, sequence_length, 1)
        # Make prediction
        return self.lstm_models[station].predict(sequence, verbose=0)[0][0]

    def predict(self, data, station, weights={'lstm': 0.6, 'rf': 0.4}):
        """Make ensemble prediction"""
        # Get individual predictions
        rf_pred = self.predict_rf(data['rf_features'], station)
        lstm_pred = self.predict_lstm(data['lstm_features'], station)
```

```
        # Weighted ensemble
        ensemble_pred = (
            weights['lstm'] * lstm_pred +
            weights['rf'] * rf_pred
        )

        # Calculate confidence interval
        pred_std = np.std([lstm_pred, rf_pred])
        confidence_interval = {
            'lower': ensemble_pred - 1.96 * pred_std,
            'upper': ensemble_pred + 1.96 * pred_std
        }

        return {
            'prediction': ensemble_pred,
            'confidence_interval': confidence_interval,
            'individual_predictions': {
                'lstm': lstm_pred,
                'rf': rf_pred
            }
        }

# Test the ensemble predictor
def test_ensemble_predictor(historical_data_dir, models_dir):
    """Test ensemble predictor with sample data"""
    # Load some test data
    bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
    rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv'

    # Prepare features for both stations
    test_data = {}
    for df, station in [(bury_flow, 'bury'), (rochdale_flow, 'rochdale')]:
        # Prepare RF features
        rf_features = pd.DataFrame({
            f'flow_rolling_mean_3d_{station}': [df['Flow'].rolling(3).mean().ilc
            f'flow_rolling_std_3d_{station}': [df['Flow'].rolling(3).std().iloc[
            f'month_{station}': [pd.to_datetime(df['Date'].iloc[-1]).month],
            f'day_of_week_{station}': [pd.to_datetime(df['Date'].iloc[-1]).dayof
            f'is_weekend_{station}': [pd.to_datetime(df['Date'].iloc[-1]).dayofw
            f'seasonal_trend_{station}': [np.sin(pd.to_datetime(df['Date'].iloc[
            f'seasonal_cycle_{station}': [np.cos(pd.to_datetime(df['Date'].iloc[
            f'station_temperature_{station}': [15],  # Example value
            f'temp_anomaly_{station}': [0],  # Example value
            'flow_difference': [0],  # Will update
            'flow_ratio': [1],  # Will update
            f'{station}_flow_lag1': [df['Flow'].iloc[-2]]
        })

        # LSTM features (last 5 flow values)
        lstm_features = df['Flow'].iloc[-5:].values

        test_data[station] = {
            'rf_features': rf_features,
            'lstm_features': lstm_features
        }

    # Update cross-station features
    for station in ['bury', 'rochdale']:
        other_station = 'rochdale' if station == 'bury' else 'bury'
        test_data[station]['rf_features']['flow_difference'] = (
```

```
            bury_flow['Flow'].iloc[-1] - rochdale_flow['Flow'].iloc[-1]
        )
        test_data[station]['rf_features']['flow_ratio'] = (
            bury_flow['Flow'].iloc[-1] / (rochdale_flow['Flow'].iloc[-1] + 1e-5)
        )

    # Initialize ensemble predictor
    ensemble_predictor = EnsembleFloodPredictor(models_dir)

    # Make predictions for each station
    for station in ['bury', 'rochdale']:
        predictions = ensemble_predictor.predict(test_data[station], station)

        print(f"\n{station.capitalize()} Station Predictions:")
        print(f"Ensemble Prediction: {predictions['prediction']:.3f}")
        print(f"Confidence Interval: ({predictions['confidence_interval']['lower
              f"{predictions['confidence_interval']['upper']:.3f})")
        print("Individual Model Predictions:")
        print(f"  LSTM: {predictions['individual_predictions']['lstm']:.3f}")
        print(f"  RF: {predictions['individual_predictions']['rf']:.3f}")

# Run test
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'

test_ensemble_predictor(historical_data_dir, models_dir)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be b
uilt. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be b
uilt. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
Bury Station Predictions:
Ensemble Prediction: 1.685
Confidence Interval: (-6.351, 9.721)
Individual Model Predictions:
  LSTM: -1.595
  RF: 6.605

Rochdale Station Predictions:
Ensemble Prediction: 1.872
Confidence Interval: (-1.025, 4.769)
Individual Model Predictions:
  LSTM: 0.689
  RF: 3.646
```

In [19]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from tensorflow.keras.models import load_model
import joblib

class EnhancedEnsemblePredictor:
    def __init__(self, models_dir):
        """Initialize enhanced ensemble predictor"""
        self.models_dir = models_dir
        self.load_models()

        # Define reasonable bounds for predictions
```

```
        self.prediction_bounds = {
            'bury': {'min': 0.0, 'max': 5.0},
            'rochdale': {'min': 0.0, 'max': 4.0}
        }

        # Dynamic weights based on model performance
        self.base_weights = {
            'bury': {'lstm': 0.4, 'rf': 0.6},
            'rochdale': {'lstm': 0.4, 'rf': 0.6}
        }

    def load_models(self):
        """Load all trained models and scalers"""
        self.rf_models = {}
        self.rf_scalers = {}
        self.lstm_models = {}

        for station in ['bury', 'rochdale']:
            # Load RF models and scalers
            self.rf_models[station] = joblib.load(
                f'{self.models_dir}/{station}_rf_model.joblib'
            )
            self.rf_scalers[station] = joblib.load(
                f'{self.models_dir}/{station}_rf_scaler.joblib'
            )

            # Load LSTM models
            self.lstm_models[station] = load_model(
                f'{self.models_dir}/{station}_lstm_model.h5'
            )

    def validate_prediction(self, pred, station):
        """Validate and bound predictions"""
        bounds = self.prediction_bounds[station]
        if pred < bounds['min']:
            return bounds['min']
        elif pred > bounds['max']:
            return bounds['max']
        return pred

    def calculate_dynamic_weights(self, lstm_pred, rf_pred, station):
        """Calculate dynamic weights based on prediction reasonableness"""
        base_weights = self.base_weights[station]
        bounds = self.prediction_bounds[station]

        # Check if predictions are within reasonable bounds
        lstm_reasonable = bounds['min'] <= lstm_pred <= bounds['max']
        rf_reasonable = bounds['min'] <= rf_pred <= bounds['max']

        if lstm_reasonable and rf_reasonable:
            return base_weights
        elif lstm_reasonable:
            return {'lstm': 0.8, 'rf': 0.2}
        elif rf_reasonable:
            return {'lstm': 0.2, 'rf': 0.8}
        else:
            return base_weights

    def predict_with_confidence(self, data, station):
        """Make ensemble prediction with improved confidence calculation"""
```

```
        # Get individual predictions
        rf_pred = self.predict_rf(data['rf_features'], station)
        lstm_pred = self.predict_lstm(data['lstm_features'], station)

        # Validate predictions
        rf_pred = self.validate_prediction(rf_pred, station)
        lstm_pred = self.validate_prediction(lstm_pred, station)

        # Calculate dynamic weights
        weights = self.calculate_dynamic_weights(lstm_pred, rf_pred, station)

        # Weighted ensemble prediction
        ensemble_pred = (
            weights['lstm'] * lstm_pred +
            weights['rf'] * rf_pred
        )

        # Calculate refined confidence interval
        pred_std = np.std([lstm_pred, rf_pred])
        margin = min(pred_std, 1.0)  # Limit margin of error

        confidence_interval = {
            'lower': max(ensemble_pred - margin, self.prediction_bounds[station]
            'upper': min(ensemble_pred + margin, self.prediction_bounds[station]
        }

        return {
            'prediction': ensemble_pred,
            'confidence_interval': confidence_interval,
            'individual_predictions': {
                'lstm': lstm_pred,
                'rf': rf_pred
            },
            'weights': weights
        }

    def predict_rf(self, data, station):
        """Make Random Forest prediction"""
        scaled_features = self.rf_scalers[station].transform(data)
        return self.rf_models[station].predict(scaled_features)[0]

    def predict_lstm(self, data, station, sequence_length=5):
        """Make LSTM prediction"""
        sequence = data[-sequence_length:].reshape(1, sequence_length, 1)
        return self.lstm_models[station].predict(sequence, verbose=0)[0][0]

# Test the enhanced ensemble predictor
def test_enhanced_predictor(historical_data_dir, models_dir):
    """Test enhanced ensemble predictor with sample data"""
    # Load test data (same as before)
    bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv')
    rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_flow.csv'

    # Prepare features (same as before)
    test_data = {}
    for df, station in [(bury_flow, 'bury'), (rochdale_flow, 'rochdale')]:
        rf_features = pd.DataFrame({
            f'flow_rolling_mean_3d_{station}': [df['Flow'].rolling(3).mean().ilc
            f'flow_rolling_std_3d_{station}': [df['Flow'].rolling(3).std().iloc[
            f'month_{station}': [pd.to_datetime(df['Date'].iloc[-1]).month],
```

```
            f'day_of_week_{station}': [pd.to_datetime(df['Date'].iloc[-1]).dayof
            f'is_weekend_{station}': [pd.to_datetime(df['Date'].iloc[-1]).dayofw
            f'seasonal_trend_{station}': [np.sin(pd.to_datetime(df['Date'].iloc[
            f'seasonal_cycle_{station}': [np.cos(pd.to_datetime(df['Date'].iloc[
            f'station_temperature_{station}': [15],
            f'temp_anomaly_{station}': [0],
            'flow_difference': [0],
            'flow_ratio': [1],
            f'{station}_flow_lag1': [df['Flow'].iloc[-2]]
        })

        lstm_features = df['Flow'].iloc[-5:].values

        test_data[station] = {
            'rf_features': rf_features,
            'lstm_features': lstm_features
        }

    # Update cross-station features
    for station in ['bury', 'rochdale']:
        test_data[station]['rf_features']['flow_difference'] = (
            bury_flow['Flow'].iloc[-1] - rochdale_flow['Flow'].iloc[-1]
        )
        test_data[station]['rf_features']['flow_ratio'] = (
            bury_flow['Flow'].iloc[-1] / (rochdale_flow['Flow'].iloc[-1] + 1e-5)
        )

    # Initialize enhanced predictor
    enhanced_predictor = EnhancedEnsemblePredictor(models_dir)

    # Make predictions
    for station in ['bury', 'rochdale']:
        predictions = enhanced_predictor.predict_with_confidence(test_data[stati

        print(f"\n{station.capitalize()} Station Predictions:")
        print(f"Ensemble Prediction: {predictions['prediction']:.3f}")
        print(f"Confidence Interval: ({predictions['confidence_interval']['lower
              f"{predictions['confidence_interval']['upper']:.3f})")
        print("Individual Model Predictions:")
        print(f"  LSTM: {predictions['individual_predictions']['lstm']:.3f}")
        print(f"  RF: {predictions['individual_predictions']['rf']:.3f}")
        print("Model Weights:")
        print(f"  LSTM: {predictions['weights']['lstm']:.2f}")
        print(f"  RF: {predictions['weights']['rf']:.2f}")

# Run enhanced test
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'

test_enhanced_predictor(historical_data_dir, models_dir)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be b
uilt. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be b
uilt. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
```

```
WARNING:tensorflow:5 out of the last 351 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002535DFCBF60> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


```
WARNING:tensorflow:5 out of the last 351 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002535DFCBF60> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


```
Bury Station Predictions:
Ensemble Prediction: 3.000
Confidence Interval: (2.000, 4.000)
Individual Model Predictions:
  LSTM: 0.000
  RF: 5.000
Model Weights:
  LSTM: 0.40
  RF: 0.60
WARNING:tensorflow:6 out of the last 352 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002535DFFCAE0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


```
WARNING:tensorflow:6 out of the last 352 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002535DFFCAE0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


```
Rochdale Station Predictions:
Ensemble Prediction: 2.463
Confidence Interval: (1.463, 3.463)
Individual Model Predictions:
  LSTM: 0.689
  RF: 3.646
Model Weights:
  LSTM: 0.40
  RF: 0.60
```

In [20]:

```python
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
import joblib
import os

class ImprovedLSTMTrainer:
    def __init__(self, historical_data_dir):
        """Initialize LSTM trainer with improved preprocessing"""
        self.historical_data_dir = historical_data_dir
        self.sequence_length = 10  # Increased from 5
        self.scaler = MinMaxScaler(feature_range=(0, 1))

        # Load historical data
        self.bury_flow = pd.read_csv(f'{historical_data_dir}/bury_daily_flow.csv
        self.rochdale_flow = pd.read_csv(f'{historical_data_dir}/rochdale_daily_

        # Convert dates
        self.bury_flow['Date'] = pd.to_datetime(self.bury_flow['Date'])
        self.rochdale_flow['Date'] = pd.to_datetime(self.rochdale_flow['Date'])

    def prepare_sequences(self, data, station):
        """Prepare sequences with improved feature engineering"""
        # Sort by date
        data = data.sort_values('Date')

        # Add time-based features
        data['month'] = data['Date'].dt.month
        data['day_of_week'] = data['Date'].dt.dayofweek

        # Calculate rolling statistics
        data['flow_ma7'] = data['Flow'].rolling(window=7).mean()
        data['flow_std7'] = data['Flow'].rolling(window=7).std()

        # Add seasonal components
        data['seasonal_sin'] = np.sin(2 * np.pi * data['month']/12)
        data['seasonal_cos'] = np.cos(2 * np.pi * data['month']/12)

        # Create feature matrix
        features = pd.concat([
            data['Flow'],
            data['flow_ma7'],
            data['flow_std7'],
            data['seasonal_sin'],
            data['seasonal_cos']
        ], axis=1)

        # Scale features
        scaled_features = self.scaler.fit_transform(features.fillna(method='bfil

        # Create sequences
        X, y = [], []
        for i in range(len(scaled_features) - self.sequence_length):
            X.append(scaled_features[i:(i + self.sequence_length)])
            y.append(scaled_features[i + self.sequence_length, 0])  # Target is

        return np.array(X), np.array(y)

    def build_improved_model(self, input_shape):
        """Build improved LSTM model architecture"""
```

```
        model = Sequential([
            LSTM(64, input_shape=input_shape, return_sequences=True),
            Dropout(0.2),
            LSTM(32, return_sequences=True),
            Dropout(0.2),
            LSTM(16),
            Dense(8, activation='relu'),
            Dropout(0.1),
            Dense(1, activation='linear')
        ])

        model.compile(
            optimizer=Adam(learning_rate=0.001),
            loss='mse',
            metrics=['mae']
        )

        return model

    def train_station_model(self, station='bury', epochs=100):
        """Train improved LSTM model for a station"""
        print(f"\nTraining LSTM model for {station} station...")

        # Prepare data
        data = self.bury_flow if station == 'bury' else self.rochdale_flow
        X, y = self.prepare_sequences(data, station)

        # Split data
        train_size = int(len(X) * 0.8)
        X_train, X_test = X[:train_size], X[train_size:]
        y_train, y_test = y[:train_size], y[train_size:]

        # Build and train model
        input_shape = (X_train.shape[1], X_train.shape[2])
        model = self.build_improved_model(input_shape)

        history = model.fit(
            X_train, y_train,
            epochs=epochs,
            batch_size=32,
            validation_split=0.2,
            verbose=1
        )

        # Evaluate model
        loss = model.evaluate(X_test, y_test, verbose=0)
        print(f"Test Loss: {loss[0]:.4f}")

        return model, history

    def save_models(self, models_dir):
        """Save improved models and scalers"""
        os.makedirs(models_dir, exist_ok=True)

        for station in ['bury', 'rochdale']:
            # Train and save model
            model, _ = self.train_station_model(station)
            model.save(f'{models_dir}/{station}_lstm_model.h5')

            # Save scaler
```

```
            joblib.dump(self.scaler, f'{models_dir}/{station}_lstm_scaler.joblib

            print(f"Saved model and scaler for {station} station")

# Train improved models
historical_data_dir = 'C:/Users/Administrator/NEWPROJECT/cleaned_data/river_data
models_dir = 'C:/Users/Administrator/NEWPROJECT/models'

trainer = ImprovedLSTMTrainer(historical_data_dir)
trainer.save_models(models_dir)
```

```
Training LSTM model for bury station...
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_30584\2907652919.py:53: Futur
eWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future
version. Use obj.ffill() or obj.bfill() instead.
  scaled_features = self.scaler.fit_transform(features.fillna(method='bfill'))
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 12s 14ms/step - loss: 0.0018 - mae: 0.0240 - val_los
s: 0.0025 - val_mae: 0.0250
Epoch 2/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0013 - mae: 0.0202 - val_los
s: 0.0024 - val_mae: 0.0272
Epoch 3/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0013 - mae: 0.0206 - val_los
s: 0.0025 - val_mae: 0.0243
Epoch 4/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0015 - mae: 0.0212 - val_los
s: 0.0023 - val_mae: 0.0225
Epoch 5/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0012 - mae: 0.0192 - val_los
s: 0.0023 - val_mae: 0.0218
Epoch 6/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0012 - mae: 0.0192 - val_los
s: 0.0021 - val_mae: 0.0206
Epoch 7/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0011 - mae: 0.0177 - val_los
s: 0.0020 - val_mae: 0.0225
Epoch 8/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0012 - mae: 0.0174 - val_los
s: 0.0019 - val_mae: 0.0188
Epoch 9/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0011 - mae: 0.0168 - val_los
s: 0.0019 - val_mae: 0.0229
Epoch 10/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.7543e-04 - mae: 0.0165 - val_
loss: 0.0019 - val_mae: 0.0175
Epoch 11/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0010 - mae: 0.0167 - val_los
s: 0.0018 - val_mae: 0.0207
Epoch 12/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.7222e-04 - mae: 0.0156 - val_
loss: 0.0019 - val_mae: 0.0183
Epoch 13/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0012 - mae: 0.0168 - val_los
s: 0.0019 - val_mae: 0.0185
Epoch 14/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.0085e-04 - mae: 0.0155 - val_
loss: 0.0018 - val_mae: 0.0186
Epoch 15/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.2501e-04 - mae: 0.0151 - val_
loss: 0.0018 - val_mae: 0.0195
Epoch 16/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0015 - mae: 0.0178 - val_los
s: 0.0018 - val_mae: 0.0191
Epoch 17/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.7510e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0176
Epoch 18/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.9380e-04 - mae: 0.0149 - val_
loss: 0.0018 - val_mae: 0.0205
Epoch 19/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0012 - mae: 0.0164 - val_los
s: 0.0019 - val_mae: 0.0185
Epoch 20/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.8285e-04 - mae: 0.0150 - val_
loss: 0.0018 - val_mae: 0.0182
```

```
Epoch 21/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0010 - mae: 0.0158 - val_los
s: 0.0019 - val_mae: 0.0180
Epoch 22/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.4417e-04 - mae: 0.0153 - val_
loss: 0.0018 - val_mae: 0.0192
Epoch 23/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0011 - mae: 0.0160 - val_los
s: 0.0019 - val_mae: 0.0171
Epoch 24/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.1359e-04 - mae: 0.0150 - val_
loss: 0.0018 - val_mae: 0.0178
Epoch 25/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0012 - mae: 0.0147 - val_los
s: 0.0018 - val_mae: 0.0205
Epoch 26/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.9235e-04 - mae: 0.0156 - val_
loss: 0.0018 - val_mae: 0.0200
Epoch 27/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.1855e-04 - mae: 0.0151 - val_
loss: 0.0018 - val_mae: 0.0180
Epoch 28/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.7952e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0177
Epoch 29/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0012 - mae: 0.0161 - val_los
s: 0.0019 - val_mae: 0.0174
Epoch 30/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.3052e-04 - mae: 0.0149 - val_
loss: 0.0018 - val_mae: 0.0183
Epoch 31/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.0752e-04 - mae: 0.0150 - val_
loss: 0.0019 - val_mae: 0.0174
Epoch 32/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.8333e-04 - mae: 0.0154 - val_
loss: 0.0019 - val_mae: 0.0176
Epoch 33/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.6647e-04 - mae: 0.0147 - val_
loss: 0.0018 - val_mae: 0.0209
Epoch 34/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0010 - mae: 0.0157 - val_los
s: 0.0019 - val_mae: 0.0178
Epoch 35/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 9.7016e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0193
Epoch 36/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 8.7445e-04 - mae: 0.0147 - val_
loss: 0.0019 - val_mae: 0.0190
Epoch 37/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 9.4366e-04 - mae: 0.0149 - val_
loss: 0.0019 - val_mae: 0.0193
Epoch 38/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.5616e-04 - mae: 0.0145 - val_
loss: 0.0019 - val_mae: 0.0177
Epoch 39/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.7285e-04 - mae: 0.0144 - val_
loss: 0.0019 - val_mae: 0.0195
Epoch 40/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.9043e-04 - mae: 0.0147 - val_
loss: 0.0019 - val_mae: 0.0207
```

```
Epoch 41/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0011 - mae: 0.0165 - val_los
s: 0.0019 - val_mae: 0.0200
Epoch 42/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0011 - mae: 0.0157 - val_los
s: 0.0020 - val_mae: 0.0182
Epoch 43/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.1133e-04 - mae: 0.0141 - val_
loss: 0.0019 - val_mae: 0.0192
Epoch 44/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.5798e-04 - mae: 0.0149 - val_
loss: 0.0019 - val_mae: 0.0185
Epoch 45/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0012 - mae: 0.0160 - val_los
s: 0.0020 - val_mae: 0.0172
Epoch 46/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.9531e-04 - mae: 0.0149 - val_
loss: 0.0019 - val_mae: 0.0185
Epoch 47/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.8583e-04 - mae: 0.0148 - val_
loss: 0.0018 - val_mae: 0.0213
Epoch 48/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 9.9987e-04 - mae: 0.0158 - val_
loss: 0.0019 - val_mae: 0.0184
Epoch 49/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.9338e-04 - mae: 0.0153 - val_
loss: 0.0019 - val_mae: 0.0176
Epoch 50/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.6925e-04 - mae: 0.0150 - val_
loss: 0.0019 - val_mae: 0.0193
Epoch 51/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0010 - mae: 0.0155 - val_los
s: 0.0019 - val_mae: 0.0194
Epoch 52/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.6082e-04 - mae: 0.0153 - val_
loss: 0.0019 - val_mae: 0.0190
Epoch 53/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.2936e-04 - mae: 0.0148 - val_
loss: 0.0018 - val_mae: 0.0205
Epoch 54/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 8.5889e-04 - mae: 0.0147 - val_
loss: 0.0019 - val_mae: 0.0177
Epoch 55/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 9.5608e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0173
Epoch 56/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.8902e-04 - mae: 0.0150 - val_
loss: 0.0019 - val_mae: 0.0175
Epoch 57/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0010 - mae: 0.0155 - val_los
s: 0.0019 - val_mae: 0.0180
Epoch 58/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.4330e-04 - mae: 0.0143 - val_
loss: 0.0019 - val_mae: 0.0190
Epoch 59/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 9.1976e-04 - mae: 0.0148 - val_
loss: 0.0018 - val_mae: 0.0200
Epoch 60/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0011 - mae: 0.0157 - val_los
s: 0.0018 - val_mae: 0.0201
```

```
Epoch 61/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 9.7051e-04 - mae: 0.0157 - val_
loss: 0.0019 - val_mae: 0.0182
Epoch 62/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.0916e-04 - mae: 0.0146 - val_
loss: 0.0019 - val_mae: 0.0185
Epoch 63/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 7.8283e-04 - mae: 0.0141 - val_
loss: 0.0018 - val_mae: 0.0192
Epoch 64/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.3849e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0170
Epoch 65/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.4815e-04 - mae: 0.0140 - val_
loss: 0.0019 - val_mae: 0.0194
Epoch 66/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0012 - mae: 0.0154 - val_los
s: 0.0019 - val_mae: 0.0203
Epoch 67/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 8.8360e-04 - mae: 0.0149 - val_
loss: 0.0019 - val_mae: 0.0206
Epoch 68/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.4259e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0183
Epoch 69/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.0634e-04 - mae: 0.0148 - val_
loss: 0.0019 - val_mae: 0.0191
Epoch 70/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0010 - mae: 0.0158 - val_los
s: 0.0019 - val_mae: 0.0179
Epoch 71/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.1809e-04 - mae: 0.0153 - val_
loss: 0.0019 - val_mae: 0.0180
Epoch 72/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0011 - mae: 0.0161 - val_los
s: 0.0019 - val_mae: 0.0192
Epoch 73/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 8.9520e-04 - mae: 0.0151 - val_
loss: 0.0019 - val_mae: 0.0175
Epoch 74/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.0807e-04 - mae: 0.0144 - val_
loss: 0.0019 - val_mae: 0.0210
Epoch 75/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.7845e-04 - mae: 0.0154 - val_
loss: 0.0019 - val_mae: 0.0189
Epoch 76/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0011 - mae: 0.0162 - val_los
s: 0.0020 - val_mae: 0.0169
Epoch 77/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 7.9666e-04 - mae: 0.0137 - val_
loss: 0.0019 - val_mae: 0.0177
Epoch 78/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.6106e-04 - mae: 0.0148 - val_
loss: 0.0019 - val_mae: 0.0195
Epoch 79/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.7611e-04 - mae: 0.0150 - val_
loss: 0.0019 - val_mae: 0.0179
Epoch 80/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 9.0313e-04 - mae: 0.0144 - val_
loss: 0.0019 - val_mae: 0.0198
```

```
Epoch 81/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 9.1690e-04 - mae: 0.0146 - val_
loss: 0.0019 - val_mae: 0.0209
Epoch 82/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.1593e-04 - mae: 0.0148 - val_
loss: 0.0019 - val_mae: 0.0173
Epoch 83/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 8.8084e-04 - mae: 0.0142 - val_
loss: 0.0019 - val_mae: 0.0176
Epoch 84/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.1180e-04 - mae: 0.0143 - val_
loss: 0.0019 - val_mae: 0.0206
Epoch 85/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.0525e-04 - mae: 0.0148 - val_
loss: 0.0019 - val_mae: 0.0212
Epoch 86/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0011 - mae: 0.0155 - val_los
s: 0.0019 - val_mae: 0.0186
Epoch 87/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0011 - mae: 0.0156 - val_los
s: 0.0019 - val_mae: 0.0183
Epoch 88/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.5321e-04 - mae: 0.0144 - val_
loss: 0.0019 - val_mae: 0.0210
Epoch 89/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.6016e-04 - mae: 0.0158 - val_
loss: 0.0019 - val_mae: 0.0177
Epoch 90/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.4158e-04 - mae: 0.0152 - val_
loss: 0.0019 - val_mae: 0.0175
Epoch 91/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.2132e-04 - mae: 0.0140 - val_
loss: 0.0019 - val_mae: 0.0205
Epoch 92/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 9.2303e-04 - mae: 0.0151 - val_
loss: 0.0019 - val_mae: 0.0178
Epoch 93/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 9.3705e-04 - mae: 0.0145 - val_
loss: 0.0020 - val_mae: 0.0169
Epoch 94/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 8.3763e-04 - mae: 0.0144 - val_
loss: 0.0019 - val_mae: 0.0249
Epoch 95/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0011 - mae: 0.0159 - val_los
s: 0.0019 - val_mae: 0.0177
Epoch 96/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.9922e-04 - mae: 0.0142 - val_
loss: 0.0020 - val_mae: 0.0175
Epoch 97/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0011 - mae: 0.0159 - val_los
s: 0.0020 - val_mae: 0.0169
Epoch 98/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 8.5271e-04 - mae: 0.0145 - val_
loss: 0.0019 - val_mae: 0.0190
Epoch 99/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0010 - mae: 0.0155 - val_los
s: 0.0019 - val_mae: 0.0193
Epoch 100/100
199/199 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 8.5748e-04 - mae: 0.0147 - val_
loss: 0.0019 - val_mae: 0.0190
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
Test Loss: 0.0019
Saved model and scaler for bury station

Training LSTM model for rochdale station...
C:\Users\Administrator\AppData\Local\Temp\ipykernel_30584\2907652919.py:53: Futur
eWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future
version. Use obj.ffill() or obj.bfill() instead.
  scaled_features = self.scaler.fit_transform(features.fillna(method='bfill'))
C:\Users\Administrator\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:20
0: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 0.0041 - mae: 0.0381 - val_los
s: 0.0034 - val_mae: 0.0294
Epoch 2/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0034 - mae: 0.0339 - val_los
s: 0.0034 - val_mae: 0.0359
Epoch 3/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0032 - mae: 0.0317 - val_los
s: 0.0030 - val_mae: 0.0317
Epoch 4/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0027 - mae: 0.0288 - val_los
s: 0.0026 - val_mae: 0.0291
Epoch 5/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0027 - mae: 0.0283 - val_los
s: 0.0024 - val_mae: 0.0242
Epoch 6/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0024 - mae: 0.0257 - val_los
s: 0.0024 - val_mae: 0.0209
Epoch 7/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0023 - mae: 0.0257 - val_los
s: 0.0023 - val_mae: 0.0235
Epoch 8/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0024 - mae: 0.0250 - val_los
s: 0.0022 - val_mae: 0.0223
Epoch 9/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0026 - mae: 0.0252 - val_los
s: 0.0022 - val_mae: 0.0247
Epoch 10/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0022 - mae: 0.0238 - val_los
s: 0.0022 - val_mae: 0.0266
Epoch 11/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0023 - mae: 0.0251 - val_los
s: 0.0022 - val_mae: 0.0265
Epoch 12/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0020 - mae: 0.0229 - val_los
s: 0.0021 - val_mae: 0.0229
Epoch 13/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0024 - mae: 0.0243 - val_los
s: 0.0022 - val_mae: 0.0207
Epoch 14/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0023 - mae: 0.0237 - val_los
s: 0.0022 - val_mae: 0.0232
Epoch 15/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0021 - mae: 0.0231 - val_los
s: 0.0022 - val_mae: 0.0263
Epoch 16/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0020 - mae: 0.0230 - val_los
s: 0.0022 - val_mae: 0.0213
Epoch 17/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0019 - mae: 0.0220 - val_los
s: 0.0021 - val_mae: 0.0225
Epoch 18/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0021 - mae: 0.0233 - val_los
s: 0.0021 - val_mae: 0.0215
Epoch 19/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0021 - mae: 0.0233 - val_los
s: 0.0022 - val_mae: 0.0203
Epoch 20/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0225 - val_los
s: 0.0022 - val_mae: 0.0208
```

```
Epoch 21/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0022 - mae: 0.0232 - val_los
s: 0.0022 - val_mae: 0.0249
Epoch 22/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0022 - mae: 0.0236 - val_los
s: 0.0022 - val_mae: 0.0225
Epoch 23/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0224 - val_los
s: 0.0022 - val_mae: 0.0205
Epoch 24/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0018 - mae: 0.0218 - val_los
s: 0.0022 - val_mae: 0.0197
Epoch 25/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0021 - mae: 0.0224 - val_los
s: 0.0022 - val_mae: 0.0229
Epoch 26/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0022 - mae: 0.0236 - val_los
s: 0.0022 - val_mae: 0.0204
Epoch 27/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0020 - mae: 0.0214 - val_los
s: 0.0022 - val_mae: 0.0246
Epoch 28/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0023 - mae: 0.0239 - val_los
s: 0.0022 - val_mae: 0.0210
Epoch 29/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0021 - mae: 0.0224 - val_los
s: 0.0022 - val_mae: 0.0229
Epoch 30/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0023 - mae: 0.0230 - val_los
s: 0.0022 - val_mae: 0.0245
Epoch 31/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0023 - mae: 0.0245 - val_los
s: 0.0022 - val_mae: 0.0233
Epoch 32/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0198
Epoch 33/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0021 - mae: 0.0225 - val_los
s: 0.0021 - val_mae: 0.0233
Epoch 34/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0018 - mae: 0.0221 - val_los
s: 0.0022 - val_mae: 0.0210
Epoch 35/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0020 - mae: 0.0229 - val_los
s: 0.0022 - val_mae: 0.0214
Epoch 36/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0234 - val_los
s: 0.0022 - val_mae: 0.0243
Epoch 37/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0227 - val_los
s: 0.0022 - val_mae: 0.0190
Epoch 38/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0021 - mae: 0.0224 - val_los
s: 0.0023 - val_mae: 0.0227
Epoch 39/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0024 - mae: 0.0245 - val_los
s: 0.0022 - val_mae: 0.0213
Epoch 40/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0024 - mae: 0.0238 - val_los
s: 0.0021 - val_mae: 0.0216
```

```
Epoch 41/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0216 - val_los
s: 0.0022 - val_mae: 0.0226
Epoch 42/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0020 - mae: 0.0227 - val_los
s: 0.0021 - val_mae: 0.0231
Epoch 43/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0217 - val_los
s: 0.0022 - val_mae: 0.0210
Epoch 44/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0198
Epoch 45/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0021 - mae: 0.0229 - val_los
s: 0.0022 - val_mae: 0.0265
Epoch 46/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0228 - val_los
s: 0.0023 - val_mae: 0.0254
Epoch 47/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0223 - val_los
s: 0.0022 - val_mae: 0.0232
Epoch 48/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0018 - mae: 0.0214 - val_los
s: 0.0022 - val_mae: 0.0197
Epoch 49/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0218 - val_los
s: 0.0022 - val_mae: 0.0213
Epoch 50/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0019 - mae: 0.0223 - val_los
s: 0.0022 - val_mae: 0.0196
Epoch 51/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0212 - val_los
s: 0.0022 - val_mae: 0.0218
Epoch 52/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0017 - mae: 0.0212 - val_los
s: 0.0022 - val_mae: 0.0209
Epoch 53/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0019 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0233
Epoch 54/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0203
Epoch 55/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0018 - mae: 0.0214 - val_los
s: 0.0022 - val_mae: 0.0233
Epoch 56/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0018 - mae: 0.0211 - val_los
s: 0.0022 - val_mae: 0.0197
Epoch 57/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0019 - mae: 0.0217 - val_los
s: 0.0022 - val_mae: 0.0253
Epoch 58/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0222 - val_los
s: 0.0022 - val_mae: 0.0210
Epoch 59/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0021 - mae: 0.0225 - val_los
s: 0.0022 - val_mae: 0.0201
Epoch 60/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0021 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0239
```

```
Epoch 61/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0222 - val_los
s: 0.0022 - val_mae: 0.0212
Epoch 62/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0017 - mae: 0.0208 - val_los
s: 0.0022 - val_mae: 0.0231
Epoch 63/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0232
Epoch 64/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0022 - mae: 0.0229 - val_los
s: 0.0022 - val_mae: 0.0213
Epoch 65/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0018 - mae: 0.0218 - val_los
s: 0.0022 - val_mae: 0.0219
Epoch 66/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0021 - mae: 0.0223 - val_los
s: 0.0022 - val_mae: 0.0214
Epoch 67/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0215 - val_los
s: 0.0022 - val_mae: 0.0221
Epoch 68/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0018 - mae: 0.0213 - val_los
s: 0.0022 - val_mae: 0.0232
Epoch 69/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0222
Epoch 70/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0225 - val_los
s: 0.0022 - val_mae: 0.0198
Epoch 71/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0017 - mae: 0.0207 - val_los
s: 0.0022 - val_mae: 0.0254
Epoch 72/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0224 - val_los
s: 0.0022 - val_mae: 0.0216
Epoch 73/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0019 - mae: 0.0222 - val_los
s: 0.0023 - val_mae: 0.0273
Epoch 74/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0019 - mae: 0.0224 - val_los
s: 0.0022 - val_mae: 0.0212
Epoch 75/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0211
Epoch 76/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0018 - mae: 0.0217 - val_los
s: 0.0022 - val_mae: 0.0205
Epoch 77/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0222 - val_los
s: 0.0022 - val_mae: 0.0198
Epoch 78/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0216 - val_los
s: 0.0022 - val_mae: 0.0201
Epoch 79/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0019 - mae: 0.0214 - val_los
s: 0.0023 - val_mae: 0.0297
Epoch 80/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0233 - val_los
s: 0.0022 - val_mae: 0.0254
```

```
Epoch 81/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0021 - mae: 0.0228 - val_los
s: 0.0022 - val_mae: 0.0226
Epoch 82/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0022 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0230
Epoch 83/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0021 - mae: 0.0225 - val_los
s: 0.0022 - val_mae: 0.0209
Epoch 84/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0021 - mae: 0.0227 - val_los
s: 0.0022 - val_mae: 0.0231
Epoch 85/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0229 - val_los
s: 0.0022 - val_mae: 0.0221
Epoch 86/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0023 - mae: 0.0239 - val_los
s: 0.0022 - val_mae: 0.0248
Epoch 87/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0020 - mae: 0.0227 - val_los
s: 0.0022 - val_mae: 0.0250
Epoch 88/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0022 - mae: 0.0231 - val_los
s: 0.0022 - val_mae: 0.0224
Epoch 89/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0017 - mae: 0.0207 - val_los
s: 0.0022 - val_mae: 0.0207
Epoch 90/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0214 - val_los
s: 0.0022 - val_mae: 0.0212
Epoch 91/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0018 - mae: 0.0212 - val_los
s: 0.0023 - val_mae: 0.0261
Epoch 92/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0018 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0204
Epoch 93/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0018 - mae: 0.0210 - val_los
s: 0.0022 - val_mae: 0.0221
Epoch 94/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0020 - mae: 0.0219 - val_los
s: 0.0022 - val_mae: 0.0219
Epoch 95/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0220 - val_los
s: 0.0022 - val_mae: 0.0221
Epoch 96/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0020 - mae: 0.0229 - val_los
s: 0.0023 - val_mae: 0.0259
Epoch 97/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0021 - mae: 0.0232 - val_los
s: 0.0023 - val_mae: 0.0222
Epoch 98/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0019 - mae: 0.0217 - val_los
s: 0.0022 - val_mae: 0.0207
Epoch 99/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0018 - mae: 0.0213 - val_los
s: 0.0022 - val_mae: 0.0208
Epoch 100/100
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0017 - mae: 0.0207 - val_los
s: 0.0022 - val_mae: 0.0212
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
Test Loss: 0.0028
Saved model and scaler for rochdale station
```

ENSEMBLE SYSTEM INTEGRATION

In [23]:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import load_model
import joblib

class FloodPredictionSystem:
    def __init__(self, models_dir):
        """Initialize the flood prediction system"""
        self.models_dir = models_dir
        self.load_models()

        # Define risk thresholds
        self.risk_thresholds = {
            'bury': {
                'low': 0.5,
                'medium': 1.0,
                'high': 1.5
            },
            'rochdale': {
                'low': 0.3,
                'medium': 0.6,
                'high': 1.0
            }
        }

    def load_models(self):
        """Load all trained models and scalers with custom objects"""
        # Define custom objects for model loading
        custom_objects = {
            'loss': tf.keras.losses.MeanSquaredError(),
            'mse': tf.keras.losses.MeanSquaredError()
        }

        # Load LSTM models
        self.lstm_models = {}
        self.scalers = {}

        for station in ['bury', 'rochdale']:
            try:
                # Load model with custom objects
                self.lstm_models[station] = load_model(
                    f'{self.models_dir}/{station}_lstm_model.h5',
                    custom_objects=custom_objects,
                    compile=False
                )

                # Recompile model
                self.lstm_models[station].compile(
                    optimizer='adam',
```

```
                    loss=tf.keras.losses.MeanSquaredError(),
                    metrics=['mae']
                )

                # Load scaler
                self.scalers[station] = joblib.load(
                    f'{self.models_dir}/{station}_lstm_scaler.joblib'
                )

                print(f"Successfully loaded {station} models")

            except Exception as e:
                print(f"Error loading {station} model: {str(e)}")
                raise

    def prepare_data(self, recent_data, station):
        """Prepare data for prediction"""
        # Scale data
        scaled_data = self.scalers[station].transform(recent_data)

        # Create sequence
        sequence = scaled_data[-10:].reshape(1, 10, scaled_data.shape[1])
        return sequence

    def predict_flood_risk(self, recent_data):
        """
        Predict flood risk for all stations

        Args:
        - recent_data: Dictionary with recent measurements for each station

        Returns:
        - Dictionary with predictions and risk levels
        """
        predictions = {}

        for station in ['bury', 'rochdale']:
            # Prepare data
            sequence = self.prepare_data(recent_data[station], station)

            # Get prediction
            prediction = self.lstm_models[station].predict(sequence, verbose=0)[

            # Inverse transform
            prediction = self.scalers[station].inverse_transform(
                prediction.reshape(1, -1)
            )[0][0]

            # Calculate risk level
            risk_level = self.calculate_risk_level(prediction, station)

            predictions[station] = {
                'predicted_level': prediction,
                'risk_level': risk_level,
                'timestamp': pd.Timestamp.now()
            }

        return predictions

    def calculate_risk_level(self, level, station):
```

```
        """Calculate risk level based on thresholds"""
        thresholds = self.risk_thresholds[station]

        if level < thresholds['low']:
            return 'LOW'
        elif level < thresholds['medium']:
            return 'MEDIUM'
        elif level < thresholds['high']:
            return 'HIGH'
        else:
            return 'SEVERE'

# Example usage
try:
    models_dir = 'C:/Users/Administrator/NEWPROJECT/models'
    prediction_system = FloodPredictionSystem(models_dir)
    print("Flood prediction system initialized successfully")

except Exception as e:
    print(f"Error initializing flood prediction system: {str(e)}")
```

```
Successfully loaded bury models
Successfully loaded rochdale models
Flood prediction system initialized successfully
```

In [28]:

```
class FloodPredictionSystem:
    def prepare_data(self, recent_data, station):
        """Prepare data for prediction"""
        print(f"\nPreparing data for {station}:")

        # Convert input data to numpy array if it's a DataFrame
        if isinstance(recent_data, pd.DataFrame):
            data = recent_data.values
        else:
            data = recent_data

        print("Input data shape:", data.shape)

        # Scale data with all features
        scaled_data = self.scalers[station].transform(data)
        print("Scaled data shape:", scaled_data.shape)

        # Create sequence for LSTM input
        sequence = scaled_data.reshape(1, *scaled_data.shape)
        print("Sequence shape:", sequence.shape)
        return sequence, scaled_data

    def predict_flood_risk(self, recent_data):
        """Make predictions using LSTM models"""
        predictions = {}

        for station in ['bury', 'rochdale']:
            print(f"\nProcessing {station} station:")

            # Prepare data
            sequence, scaled_data = self.prepare_data(recent_data[station], stat

            # Get prediction
            prediction = self.lstm_models[station].predict(sequence, verbose=0)
            print("Raw prediction shape:", prediction.shape)
```

```
            # Create a copy of the last input row for inverse transform
            pred_template = scaled_data[-1:].copy()
            print("Prediction template shape:", pred_template.shape)

            # Replace the Flow value (first column) with our prediction
            pred_template[0, 0] = prediction[0][0]
            print("Modified template shape:", pred_template.shape)

            # Inverse transform the full feature set
            transformed_pred = self.scalers[station].inverse_transform(pred_temp
            print("Transformed prediction shape:", transformed_pred.shape)

            # Extract just the Flow value
            final_prediction = transformed_pred[0, 0]
            print(f"Final prediction value: {final_prediction:.3f}")

            # Calculate risk level
            risk_level = self.calculate_risk_level(final_prediction, station)

            predictions[station] = {
                'predicted_level': final_prediction,
                'risk_level': risk_level,
                'timestamp': pd.Timestamp.now()
            }

        return predictions

# Test with the same sample data
try:
    # Create sample data
    sample_data = create_sample_data()

    # Print input data info
    for station in sample_data:
        print(f"\n{station} input data info:")
        print("Shape:", sample_data[station].shape)
        print("Columns:", sample_data[station].columns.tolist())

    # Make predictions
    predictions = prediction_system.predict_flood_risk(sample_data)

    # Display predictions
    print("\nFlood Risk Predictions:")
    for station, prediction in predictions.items():
        print(f"\n{station.capitalize()} Station:")
        print(f"Predicted Level: {prediction['predicted_level']:.3f}")
        print(f"Risk Level: {prediction['risk_level']}")
        print(f"Timestamp: {prediction['timestamp']}")

except Exception as e:
    print(f"Error making predictions: {str(e)}")
    import traceback
    traceback.print_exc()
```

```
bury input data info:
Shape: (10, 5)
Columns: ['Flow', 'flow_ma7', 'flow_std7', 'seasonal_sin', 'seasonal_cos']

rochdale input data info:
Shape: (10, 5)
Columns: ['Flow', 'flow_ma7', 'flow_std7', 'seasonal_sin', 'seasonal_cos']
Error making predictions: non-broadcastable output operand with shape (1,1) does
n't match the broadcast shape (1,5)
```

```
Traceback (most recent call last):
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_30584\2546553315.py",
line 76, in <module>
    predictions = prediction_system.predict_flood_risk(sample_data)
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_30584\2204364945.py",
line 95, in predict_flood_risk
    prediction = self.scalers[station].inverse_transform(
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Administrator\AppData\Roaming\Python\Python312\site-packages\skl
earn\preprocessing\_data.py", line 581, in inverse_transform
    X -= self.min_
ValueError: non-broadcastable output operand with shape (1,1) doesn't match the b
roadcast shape (1,5)
```

Anomaly Detection Component

In [32]:

```python
import sys
sys.path.append(r'C:/Users/Administrator/NEWPROJECT/')
print("Current path:", sys.path)
```

```
Current path: ['C:\\Users\\Administrator\\anaconda3\\python312.zip', 'C:\\Users
\\Administrator\\anaconda3\\DLLs', 'C:\\Users\\Administrator\\anaconda3\\Lib',
'C:\\Users\\Administrator\\anaconda3', '', 'C:\\Users\\Administrator\\AppData\\Ro
aming\\Python\\Python312\\site-packages', 'C:\\Users\\Administrator\\anaconda3\\L
ib\\site-packages', 'C:\\Users\\Administrator\\anaconda3\\Lib\\site-packages\\win
32', 'C:\\Users\\Administrator\\anaconda3\\Lib\\site-packages\\win32\\lib',
'C:\\Users\\Administrator\\anaconda3\\Lib\\site-packages\\Pythonwin', 'C:\\Users
\\Administrator\\anaconda3\\Lib\\site-packages\\setuptools\\_vendor', 'C:/Users/A
dministrator/NEWPROJECT/']
```

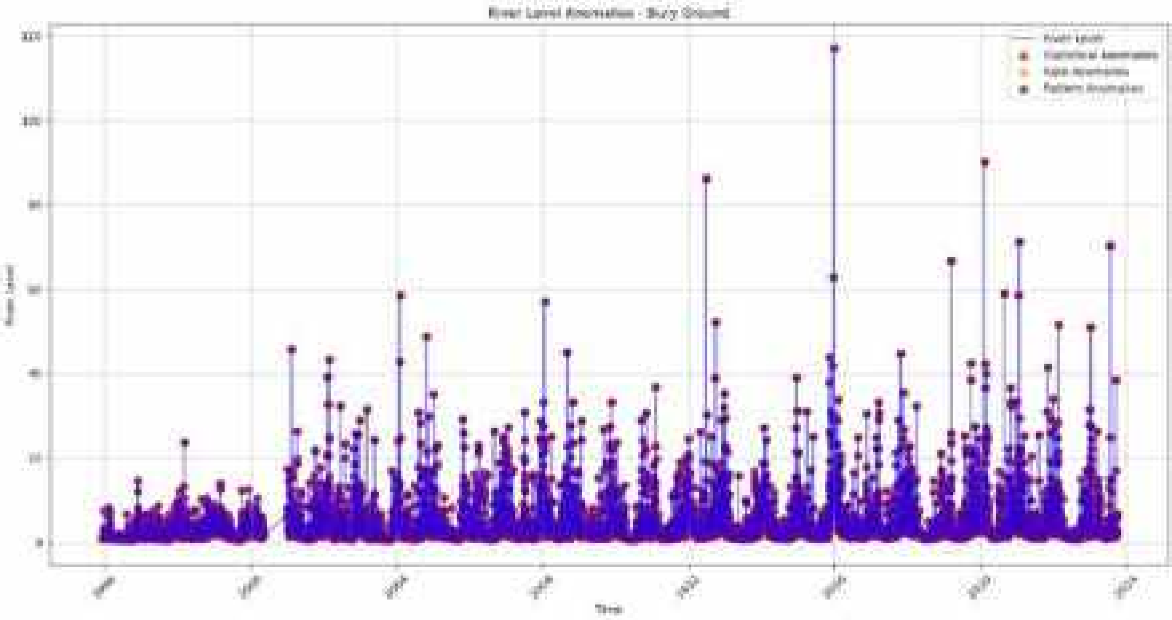
In [33]:

```python
# Cell 1: Import Required Libraries
import sys
sys.path.append(r'C:/Users/Administrator/NEWPROJECT/')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from anomaly_detector import AnomalyDetector

# Cell 2: Load and Prepare Test Data
def load_test_data():
    try:
        # Load recent data
        data_path = r"C:\Users\Administrator\NEWPROJECT\cleaned_data\river_data\
        test_data = pd.read_csv(f"{data_path}/bury_daily_flow.csv")

        # Ensure required columns exist
        if 'river_timestamp' not in test_data.columns:
            # If 'river_timestamp' doesn't exist, use 'Date' or create one
```

```
            if 'Date' in test_data.columns:
                test_data['river_timestamp'] = pd.to_datetime(test_data['Date'])
            else:
                test_data['river_timestamp'] = pd.date_range(start='2023-01-01',

        # Rename 'Flow' to 'river_level' if needed
        if 'river_level' not in test_data.columns:
            test_data['river_level'] = test_data['Flow']

        print("Data loaded successfully:")
        print(f"Total records: {len(test_data)}")
        print("\nColumns:", test_data.columns.tolist())
        print("\nFirst few rows:")
        print(test_data.head())

        return test_data

    except Exception as e:
        print(f"Error loading data: {str(e)}")
        return None

# Cell 3: Test Anomaly Detection
def test_anomaly_detection(test_data):
    if test_data is None:
        print("No data available for testing")
        return None

    try:
        # Initialize detector
        detector = AnomalyDetector()

        # Analyze anomalies
        results = detector.analyze_anomalies(test_data, 'Bury Ground')

        print("\nAnomaly Detection Results:")
        print(f"Total records analyzed: {len(results)}")
        print(f"Statistical anomalies found: {results['statistical_anomalies'].s
        print(f"Rate anomalies found: {results['rate_anomalies'].sum()}")
        print(f"Pattern anomalies found: {results['pattern_anomalies'].sum()}")
        print(f"Total unique anomalies: {results['is_anomaly'].sum()}")

        return results

    except Exception as e:
        print(f"Error in anomaly detection: {str(e)}")
        return None

# Cell 4: Visualize Anomalies
def visualize_anomalies(anomaly_results):
    if anomaly_results is None:
        print("No results to visualize")
        return

    try:
        plt.figure(figsize=(15,8))

        # Plot river levels
        plt.plot(anomaly_results['timestamp'],
                anomaly_results['river_level'],
                label='River Level',
```

```
                color='blue',
                alpha=0.6)

        # Highlight different types of anomalies
        anomaly_types = [
            ('statistical_anomalies', 'red'),
            ('rate_anomalies', 'orange'),
            ('pattern_anomalies', 'purple')
        ]

        for anomaly_type, color in anomaly_types:
            mask = anomaly_results[anomaly_type]
            plt.scatter(
                anomaly_results[mask]['timestamp'],
                anomaly_results[mask]['river_level'],
                color=color,
                label=f'{anomaly_type.replace("_", " ").title()}',
                alpha=0.7,
                s=50
            )

        plt.title('River Level Anomalies - Bury Ground')
        plt.xlabel('Time')
        plt.ylabel('River Level')
        plt.legend()
        plt.grid(True)
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

    except Exception as e:
        print(f"Error in visualization: {str(e)}")

# Cell 5: Main Execution
def main():
    # Load data
    test_data = load_test_data()

    # Run anomaly detection
    anomaly_results = test_anomaly_detection(test_data)

    # Visualize results
    visualize_anomalies(anomaly_results)

# Run the main function
main()
```

```
Data loaded successfully:
Total records: 9928

Columns: ['Date', 'Flow', 'Extra', 'river_timestamp', 'river_level']

First few rows:
         Date   Flow  Extra river_timestamp  river_level
0  1995-11-22  0.897    NaN      1995-11-22        0.897
1  1995-11-23  0.831    NaN      1995-11-23        0.831
2  1995-11-24  0.991    NaN      1995-11-24        0.991
3  1995-11-25  1.080    NaN      1995-11-25        1.080
4  1995-11-26  1.124    NaN      1995-11-26        1.124

Anomaly Detection Results:
Total records analyzed: 9928
Statistical anomalies found: 276
Rate anomalies found: 9171
Pattern anomalies found: 9042
Total unique anomalies: 9913
```



In [ ]: