Predicting the Missing data using historic & new data combined with Rainfall Data

There are multiple applications of machine learning and artificial intelligence (AI) in the water industry, such as predicting blockages, spillages, signal coverage issues, and more. For this project, I have chosen to focus on addressing the problem of missing data that occasionally appears in the trend lines of data collected from sewer systems. Water utilities aim to ensure no gaps in their data, as missing data impacts the accuracy of analytics and decision-making. If we can deploy a model on edge-computing devices to address this issue, it would significantly improve data reliability and make the devices more robust.

The Problem:

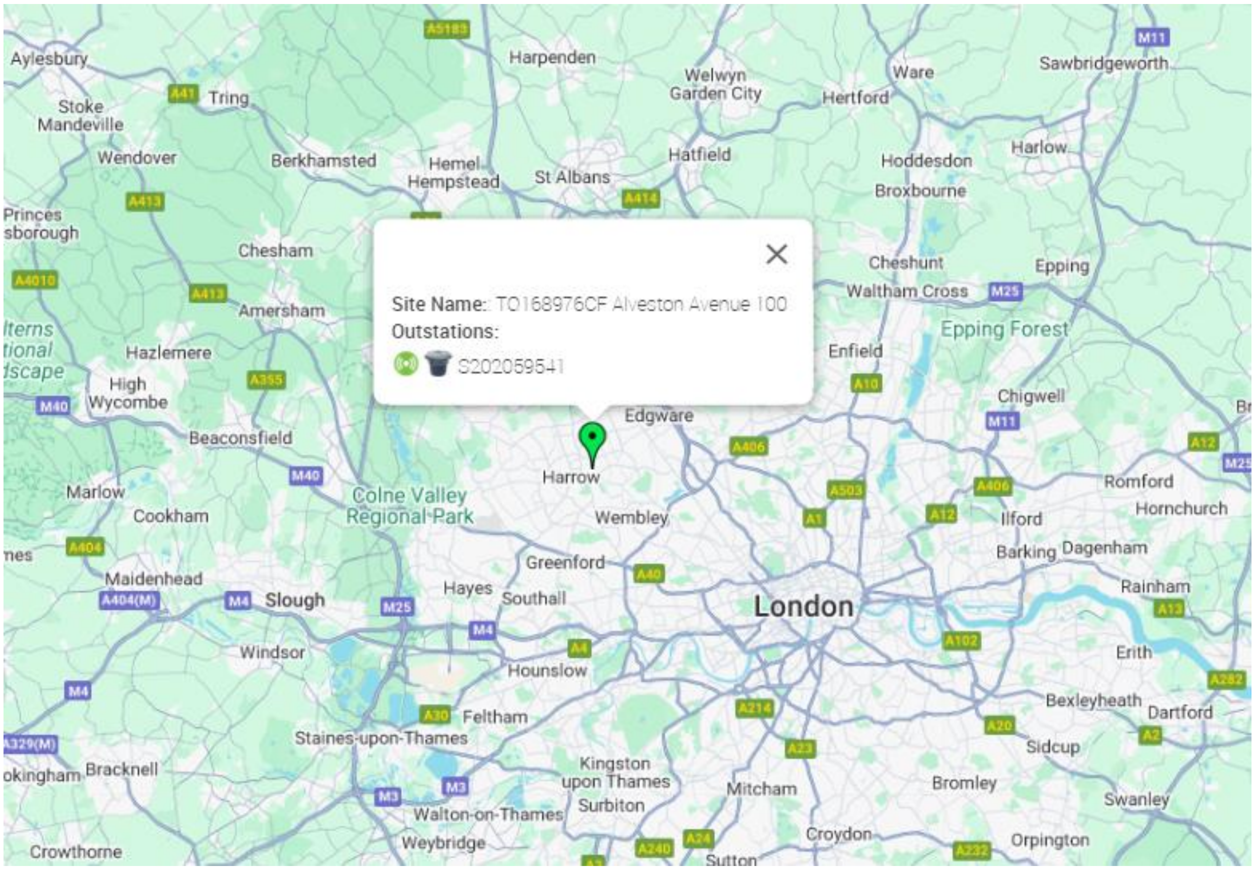
Remote telemetry loggers, equipped with level sensors to monitor flow in sewer systems, typically collect data every 15 minutes. The data is transmitted to the cloud once a day, except during alarm events when the collection frequency increases to every minute, and the system dials in every 30 minutes or every hour. However, sometimes, due to reflections or noise, some of the data points collected—out of, say, 40 in a day—may be incorrect due to sensor noise or false echoes. If we can use the previous and subsequent data points to average and predict missing or erroneous values, we can create a more reliable and accurate system. This would ensure that the analytics used by water utilities produce better insights, as the data would be smoothed and corrected based on the trend observed just before and after the missing point, ensuring it stays within reasonable limits.

Solution:

Various machine learning techniques could be used to address missing data, including simple mathematical averaging algorithms. However, basic averaging does not take into account historical data from previous months or years, which could provide more accurate predictions. As a result, I opted for a black-box function to process the data, predict the missing points, and ensure that the dataset sent from the remote telemetry logger is as accurate as possible. If the missing or noisy data continues to appear over time, we could also implement tuning mechanisms in the black-box function to account for persistent issues, which may be caused by other factors, such as sensor malfunctions.

Dataset:

For this test project, I am using data from a live site in Harrow.


Site Name:: TQ168976CF Alveston Avenue 100
Outstations:
S202059541
Aylesbury
Harpenden
Welwyn Garden City
Ware
Sawbridgeworth
Stoke Mandeville
Tring
Hertford
Wendover
Berkhamsted
Hemel Hempstead
St Albans
Hatfield
Hoddesdon
Harlow
Broxbourne
Chesham
Cheshunt
Epping
Amersham
Waltham Cross
Epping Forest
Hazlemere
Enfield
High Wycombe
Chigwell
Beaconsfield
Edgware
Harrow
Romford
Marlow
Colne Valley Regional Park
Wembley
Hornchurch
Cookham
Ilford
Barking
Dagenham
Greenford
Maidenhead
Hayes
Southall
Rainham
Slough
London
Windsor
Erith
Hounslow
Bexleyheath
Dartford
Feltham
Staines-upon-Thames
Sidcup
Bracknell
Kingston upon Thames
Bromley
Mitcham
Swanley
Surbiton
Walton-on-Thames
Croydon
Orpington
Weybridge
Crowthorne
Sutton

TQ168976CF Alveston Avenue 100

Summary Map Events Outstations Alarms Site Images Meter Report

Summary

14:04:48 22 Sep 2024

13:03:11 05 Feb 2024

3.51 V

-109 dBm

19.5 DegC

Site Images

The dataset includes level and timestamp data points collected by the remote telemetry logger at the site. Additionally, I have integrated rainfall forecast and historical data from the UK Met Office. In our usual workflow, an API pulls this rainfall data into our system, but for this project, I retrieved the data manually and included it in the project folder. These two data sources—the telemetry logger data and the rainfall data—will serve as the basis for predicting the noisy or missing level values. I specifically selected this site for the project because it contains instances of noisy data that need correction.

Code Function:

The goal of the code is to predict and correct negative or spiked data points using both previous and future data from the remote telemetry logger, along with rainfall data recorded during the same timeframe. The code will ingest data from both the logger and the Met Office's rainfall forecast and historical records. Using a black-box function, the code will predict and replace any negative or spiked data. The black-box function will rely on 80% of historical level readings and 20% of the forecasted rainfall data, while also incorporating 2-4 previous and forward data points to make more accurate predictions.

Additionally, if more than 4 spikes are detected in a minute, hyperparameter tuning has been implemented to ensure that the spike data is not predicted or averaged. Instead, the system assumes that multiple spikes in this time frame indicate a significant event, and the data will be recorded as is, flagging it as an important alarm event. This feature is essential for making the device more robust and ensures that critical data points related to system events are not altered. The telemetry system becomes more resilient and can handle edge cases where the spike represents an actual event rather than sensor noise.

My Code:

Step 1: First step, I added the dataset into the Jupyter Folder and labeled it as LevelData and ForecastAndHistoric. Once added as seen in the code below dataset has been loaded.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the main dataset from an Excel file with specific columns
def load_main_data(excel_path):
    """
    Load the main dataset from the Excel file.
    Only 'time' and 'Level' columns are read.
    """
    try:
        main_df = pd.read_excel("LevelData.xlsx", usecols=['time', 'Level'])
        main_df['time'] = pd.to_datetime(main_df['time'])  # Ensure 'time' column is datetime
        return main_df
    except Exception as e:
        print(f"Error loading main data: {e}")
        return None

# Load the rainfall forecast data from an Excel file
def load_forecast_data(excel_path):
    """
    Load the forecast data from an Excel file.
    The Excel file is expected to have 'date' and 'forecast_rainfall' columns.
    """
    try:
        forecast_df = pd.read_excel("ForecastAndHistoric.xlsx")
        forecast_df['date'] = pd.to_datetime(forecast_df['date'])  # Ensure 'date' column is datetime
        return forecast_df.set_index('date')['forecast_rainfall'].to_dict()  # Convert to a dictionary
    except Exception as e:
        print(f"Error loading forecast data: {e}")
        return None
```

```python
# Path to the Excel files
main_data_path = 'LevelData.xlsx'  # Main dataset (Excel file path)
rainfall_forecast_path = 'ForecastAndHistoric.xlsx'  # Rainfall forecast data (Excel file path)

# Load the data
main_data = load_main_data(main_data_path)
forecast_rainfall = load_forecast_data(rainfall_forecast_path)

# Check only the first 5 rows of the data to avoid large output
if main_data is not None:
    print(main_data.head(5))  # Limiting output to 5 rows

if forecast_rainfall is not None:
    print({k: forecast_rainfall[k] for k in list(forecast_rainfall.keys())[:5]})  # Show only 5 entries
```

```
                time   Level
0 2024-02-05 09:45:00 -2600.0
1 2024-02-05 10:00:00   250.0
2 2024-02-05 10:15:00 -2600.0
3 2024-02-05 10:30:00 -2600.0
4 2024-02-05 10:45:00 -2600.0
{Timestamp('2023-03-01 00:00:00'): 0.0, Timestamp('2023-03-01 00:05:00'): 0.0, Timestamp('2023-03-01 00:10:00'): 0.0, Timestamp('2023-03-01 00:
15:00'): 0.0, Timestamp('2023-03-01 00:20:00'): 0.0}
```

I have added a print function to check if the dataset is correct or not. Once confirmed, move to the next step.

Step 2: Check if there are any negative values and then run the black box function using the Level Data and the Rainfall Data. This function predicts an average value for negative data

```python
# Define a function to check if a value is negative
def is_negative(value):
    return value < 0
```

```python
# Black-box function to estimate values using forecast and historical data, and round off the corrected value
def black_box_estimation(index, main_data, forecast_rainfall):
    """
    Estimate the value using a black-box function that averages
    the previous 2-4 readings, forward 2-4 readings, and uses forecast rainfall data.
    The corrected value is rounded off to the nearest integer.
    """
    num_rows = len(main_data)
    current_time = main_data.loc[index, 'time']

    # Get the forecasted rainfall value if available, otherwise use 0
    forecasted_rainfall = forecast_rainfall.get(current_time.date(), 0)

    # Get the previous 2-4 readings (within bounds)
    previous_readings = main_data['Level'].iloc[max(0, index-4):index].values
    # Get the forward 2-4 readings (within bounds)
    forward_readings = main_data['Level'].iloc[index+1:min(num_rows, index+5)].values

    # Combine previous and forward readings to compute average
    all_readings = np.concatenate((previous_readings, forward_readings))

    # Compute average of the readings
    if len(all_readings) > 0:
        average_readings = np.mean(all_readings)
    else:
        average_readings = 0  # Default to 0 if no readings are available

    # Incorporate forecasted rainfall into the final estimated value
    estimated_value = 0.8 * average_readings + 0.2 * forecasted_rainfall

    # Round off the corrected value to the nearest integer
    corrected_value = round(estimated_value)

    # Ensure that the corrected value is non-negative
    corrected_value = max(corrected_value, 0)

    # Print out the correction details
    print(f"Black-box correction at index {index}: Corrected value = {corrected_value} (was {estimated_value})")

    return corrected_value
```

```python
# Create a copy of the main data to store the corrected values
corrected_data = main_data.copy()

# Iterate through the main data and fix negative values
installation_issue = False
negative_start_time = None
consecutive_negatives = 0

for i in range(len(main_data)):
    level_value = main_data.loc[i, 'Level']

    if is_negative(level_value):
        if negative_start_time is None:
            negative_start_time = i
        consecutive_negatives += 1

        # Estimate the value using the black-box function
        corrected_data.loc[i, 'Level'] = black_box_estimation(i, main_data, forecast_rainfall)
    else:
        if consecutive_negatives > 0:
            if consecutive_negatives > 12:  # If more than 12 hours of negative values
                installation_issue = True
                print(f"Potential installation issue detected starting at index {negative_start_time}")
            consecutive_negatives = 0
            negative_start_time = None
```

```
Black-box correction at index 0: Corrected value = 0 (was -1510.0)
Black-box correction at index 2: Corrected value = 0 (was -1700.0)
Black-box correction at index 3: Corrected value = 0 (was -1754.2857142857142)
Black-box correction at index 4: Corrected value = 0 (was -1795.0)
Black-box correction at index 5: Corrected value = 0 (was -1795.0)
Black-box correction at index 6: Corrected value = 0 (was -2080.0)
Black-box correction at index 7: Corrected value = 0 (was -2080.0)
Black-box correction at index 8: Corrected value = 0 (was -1821.0)
Black-box correction at index 9: Corrected value = 0 (was -1560.5)
Black-box correction at index 10: Corrected value = 0 (was -1299.7)
Black-box correction at index 11: Corrected value = 0 (was -1039.5)
Black-box correction at index 12: Corrected value = 0 (was -1037.1000000000001)
Black-box correction at index 93: Corrected value = 4 (was 3.9000000000000004)
Black-box correction at index 94: Corrected value = 4 (was 4.0)
Black-box correction at index 95: Corrected value = 3 (was 3.1)
Black-box correction at index 106: Corrected value = 4 (was 3.8000000000000003)
Black-box correction at index 231: Corrected value = 6 (was 6.4)
Black-box correction at index 232: Corrected value = 6 (was 6.4)
```

The black box function has predicted the negative values using the 4 previous and forward data points along with the rainfall data during that period.

Step 3: Once the negative data is rectified this function checks for irregular spikes in the data caused from sensor issues. If a spike of more than 200 mm occurs and its less than 4 data points, we conclude this is due to sensor not being able to pick up the data correctly therefore we correct it. It uses the corrected data from the black box function and the forecast rainfall to predict the values.

Note: The target is to use for spikes large than 200 mm however for this project I have changed the threshold to 130 mm to ensure the code is working correctly.

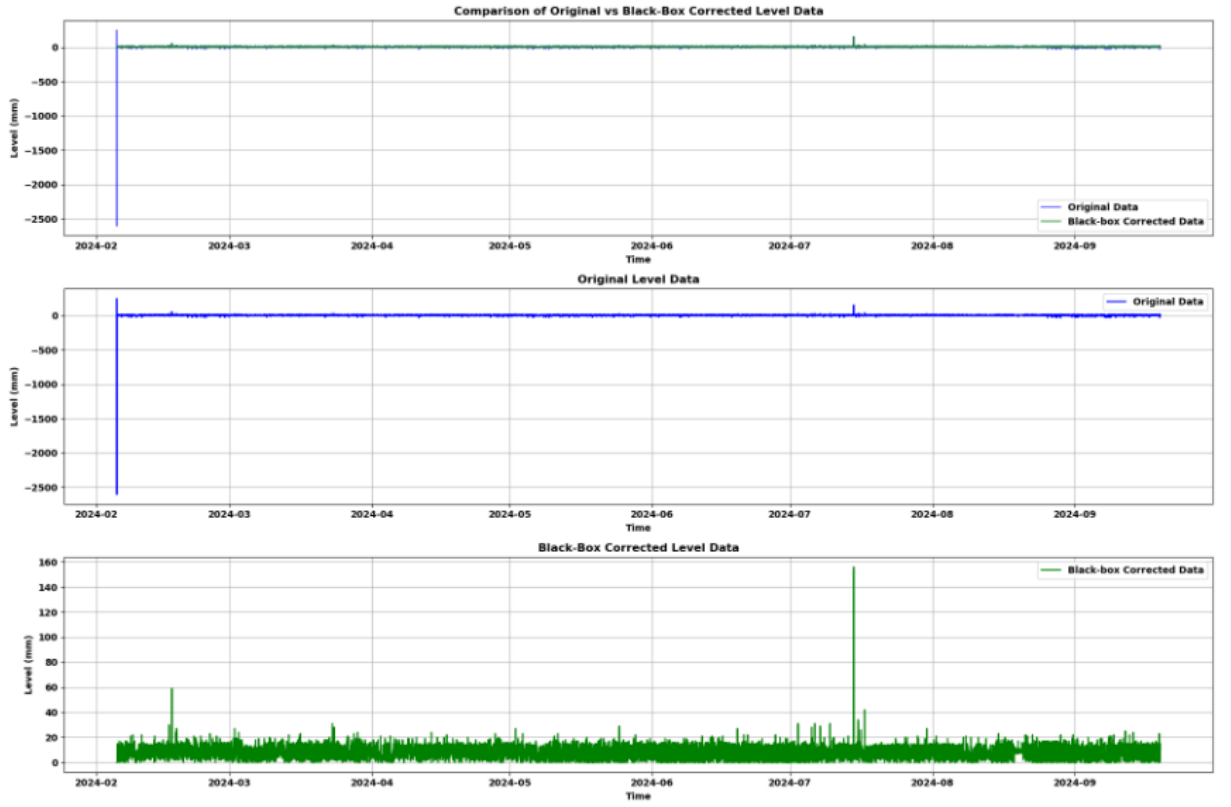
```python
def check_spikes(corrected_data, forecast_rainfall):
    """
    This function checks for spikes larger than 200 mm relative to the previous data
    and corrects them if found. If more than 4 consecutive spikes are on the same level,
    they are not averaged. The correction is done by averaging the previous 2-4 values
    and forward 2-4 values. Additionally, any negative values in the corrected data
    are set to 0.
    """
    spike_threshold = 130
    consecutive_spikes = 0  # To track consecutive spikes
    num_rows = len(corrected_data)

    for i in range(1, num_rows - 1):
        level_value = corrected_data.loc[i, 'Level']
        previous_value = corrected_data.loc[i-1, 'Level']

        # Check if the spike is more than 200 mm larger than the previous data
        if level_value > previous_value + spike_threshold:
            consecutive_spikes += 1

            # Check if it's a single spike (not part of a sequence longer than 4)
            if consecutive_spikes <= 4:
                # Get the previous 2-4 readings (within bounds)
                previous_readings = corrected_data['Level'].iloc[max(0, i-4):i].values
                # Get the forward 2-4 readings (within bounds)
                forward_readings = corrected_data['Level'].iloc[i+1:min(num_rows, i+5)].values

                # Combine previous and forward readings to compute average
                all_readings = np.concatenate((previous_readings, forward_readings))

                if len(all_readings) > 0:
                    corrected_value = np.mean(all_readings)
                else:
                    corrected_value = level_value  # If no readings are available, leave the spike as it is

                # Ensure the corrected value is not negative
                corrected_value = max(0, corrected_value)

                # Apply the correction
                corrected_data.loc[i, 'Level'] = round(corrected_value)
                print(f"Spike detected at index {i} with value {level_value}. Corrected to: {corrected_value}")

        else:
            # If no spike is detected, reset consecutive spikes
            if consecutive_spikes > 4:
                print(f"More than 4 consecutive spikes detected starting at index {i - consecutive_spikes}. No correction applied.")

            consecutive_spikes = 0  # Reset the consecutive spike count when no spike is detected

    return corrected_data
```

```python
# Apply the spike checking and correction
corrected_data = check_spikes(corrected_data, forecast_rainfall)

# Save the cleaned and corrected data back to a CSV file
corrected_data.to_csv('corrected_level_data.csv', index=False)
```

```
Spike detected at index 1 with value 250.0. Corrected to: 0
Spike detected at index 15393 with value 150.0. Corrected to: 78.75
```

Step 4: Plotting a graph to show how the data looked like with the predicting function vs when using the predicting function.

```python
import matplotlib.pyplot as plt

# Create a figure with subplots
plt.figure(figsize=(18, 12))

# Plot 1: Comparison of original and corrected data
plt.subplot(3, 1, 1)
plt.plot(main_data['time'], main_data['Level'], label='Original Data', color='blue', alpha=0.6)
plt.plot(corrected_data['time'], corrected_data['Level'], label='Black-box Corrected Data', color='green', alpha=0.7)
plt.title('Comparison of Original vs Black-Box Corrected Level Data')
plt.xlabel('Time')
plt.ylabel('Level (mm)')
plt.legend()
plt.grid(True)

# Plot 2: Original data only
plt.subplot(3, 1, 2)
plt.plot(main_data['time'], main_data['Level'], label='Original Data', color='blue')
plt.title('Original Level Data')
plt.xlabel('Time')
plt.ylabel('Level (mm)')
plt.legend()
plt.grid(True)

# Plot 3: Black-box corrected data only
plt.subplot(3, 1, 3)
plt.plot(corrected_data['time'], corrected_data['Level'], label='Black-box Corrected Data', color='green')
plt.title('Black-Box Corrected Level Data')
plt.xlabel('Time')
plt.ylabel('Level (mm)')
plt.legend()
plt.grid(True)

# Adjust layout to prevent overlapping of subplots
plt.tight_layout()

# Show the plots
plt.show()
```



As the green plot shows using the machine learning function the data has been predicted and producing a data quality of 100%

Conclusion:

In conclusion, this project illustrates how machine learning can be used to enhance the accuracy and robustness of data collection in sewer systems. By integrating a black-box function that incorporates both historical and forecasted rainfall data, we can predict and correct missing or erroneous data points from remote telemetry loggers. This approach enhances the quality of the data being used for analytics, ensuring that water utilities can make more informed decisions. Furthermore, the addition of hyperparameter tuning to avoid predicting crucial spike events ensures that important alarm signals are not missed. This system represents a significant step toward creating more reliable, edge-based data collection systems for the water industry.