INDIAN INSTITUTE OF TECHNOLOGY KANPUR

DEPARTMENT OF MATHEMATICS AND STATISTICS

A REPORT SUBMITTED FOR FULFILLMENT OF THE COURSE UNDERGRADUATE PROJECT-II (MTH392A)

Construction of a Water Quality Index

Author:
Tejus Khandelwal

Supervisor:
Prof. Amit Mitra

December 6, 2024

Abstract

This study presents the development of a comprehensive Water Quality Index (WQI) tailored for India, utilizing an extensive dataset comprising various water quality parameters collected from multiple districts across all states over several years. The methodology commenced with data preprocessing and the creation of a labeled training dataset, where each parameter measurement was classified as indicative of healthy water based on the Bureau of Indian Standards (BIS) limits. A neural network was subsequently trained to predict the healthiness of water using these parameters. To ascertain the relative significance of each parameter in determining water quality, advanced model interpretation techniques such as Layer-wise Relevance Propagation (LRP) and Shapley values were employed to derive their respective weights for the additive WQI. Following this, sub-indices for individual parameters were formulated, and overall index values were calculated to reflect the water quality status. The resulting WQI was visualized through heatmaps, highlighting spatial and temporal variations in water quality across different states and years. This index provides a robust and scalable tool for monitoring water quality, facilitating informed decision-making for environmental management, policy formulation, and public health initiatives in India.

Contents

Chapter 1

Introduction

1.1 What is a Water Quality Index ?

A Water Quality Index (WQI) is a standardized tool designed to evaluate and communicate the overall quality of water bodies in a single, comprehensible numerical value. By integrating multiple water quality parameters into a unified index, the WQI simplifies the complex data associated with water quality assessments, making it accessible to policymakers, environmental managers, and the general public. Typically, the index is calculated by assigning weights to various parameters based on their relative importance and aggregating them to produce an overall score. This score categorizes the water quality into different classes, such as excellent, good, moderate, poor, or unsuitable for specific uses like drinking, irrigation, or aquatic life support.

1.2 Why do we need a WQI ?

1. **Simplification and Communication:** It breaks down complex datasets into an easily understandable format, allowing effective communication of water quality status to non-experts, including the public and decision-makers.
2. **Monitoring and Management:** A WQI serves as a valuable tool for continuous monitoring of water bodies, enabling the detection of trends, identification of pollution sources, and assessment of the effectiveness of management strategies.
3. **Policy Formulation and Enforcement:** Policymakers can rely on WQIs to develop regulations, set standards, and implement policies aimed at protecting and improving water resources.

1.3 Types of WQI

1.3.1 NSF-Additive Water Quality Index (AWQI) Model

The NSF-Additive Water Quality Index (AWQI) Model was developed by the National Sanitation Foundation (NSF) of USA to provide a straightforward method for assessing water quality based on multiple parameters. The AWQI operates on the principle of additive aggregation, where each water quality parameter is assigned a specific weight reflecting its relative importance. The overall water quality index is then calculated by summing the weighted scores of all individual parameters.

$$\text{AWQI} = \sum_{i=1}^{n} w_i \cdot S_i \tag{1.1}$$

Where:

- n is the total number of water quality parameters.

- w_i is the weight assigned to the i^{th} parameter.
- S_i is the sub-index score of the i^{th} parameter.

1.3.2 NSF-Multiplicative Water Quality Index (MWQI) Model

The NSF-Multiplicative Water Quality Index (MWQI) is an alternative model developed to assess water quality by incorporating a multiplicative approach rather than an additive one. Like the NSF-Additive model, it aims to combine various water quality parameters into a single index, but it differs in how it handles the interaction between parameters. In the multiplicative model, lower values for any one parameter can disproportionately lower the overall score, which makes it particularly sensitive to any single poor water quality indicator. This characteristic makes the MWQI suitable for identifying water bodies with critical pollution issues where even one severely impacted parameter can pose significant health and environmental risks. The general formula is:

$$\text{MWQI} = \prod_{i=1}^{n} (S_i)^{w_i} \tag{1.2}$$

Where:

- n is the total number of water quality parameters.
- w_i is the weight assigned to the i^{th} parameter.
- S_i is the sub-index score of the i^{th} parameter.

1.3.3 Oregon Water Quality Index (OWQI) Model

The Oregon Water Quality Index (OWQI) is a model developed by the Oregon Department of Environmental Quality (DEQ) to assess and communicate the status of water quality in Oregon's rivers and streams. This index integrates multiple water quality parameters into a single score, making it easier to track and communicate the condition of water bodies. The OWQI model is designed to be sensitive to changes in water quality over time and can help identify trends, pinpoint areas needing improvement, and evaluate the effectiveness of pollution control measures. The general formula is -

$$\text{OWQI} = \sqrt{\frac{n}{\sum_{i=1}^{n} \frac{1}{S_i^2}}}$$

Where:

- n is the total number of water quality parameters.
- S_i is the sub-index score of the i^{th} parameter.

1.4 Additive WQI

We have calculated the indices based on the AWQI model. Here are a few reasons-

1.4.1 Advantages of using AWQI

1. **Simplicity and Ease of Calculation:** Easily implementable and understandable.
2. **Flexibility in Weight Assignment:** Can accommodate regional variations and specific water quality concerns.
3. **Scalability:** The additive WQI can handle varying numbers of parameters without significant alterations to its structure.

1.4.2 Steps to construct the AWQI

Preparing the dataset:

In order to start with the index calculation one needs the value of various water quality parameters like pH, TDS, Total Hardness etc. for each district in each state in the country. Then we need to clean the dataset by dropping the null values, ensure that the values in the dataset are in correct format (for example, if the latitude and longitude are consistently in the same format). Depending on the dataset other operations might be needed.

Estimating weights:

One needs to have the weights in order to calculate the index. We can get it through experts or can use machine learning algorithms as we have used in this project.

Preparing the sub indices:

Different water quality parameters have different measurement units and scales (e.g., pH is unit less, while nitrate is measured in mg/L). Sub-indices standardize these values so they can be combined meaningfully. We need to consistently map the value of these parameters between 0 and 100 to ensure uniformity in in the index calculation.

Interpreting:

After calculating the index values one can plot heat maps of different states and view it as a time series to identify hot spots and make policy decisions.

We will be describing these steps in detail for our project in the following pages.

Chapter 2

Preparing the Dataset

The dataset we worked on was Ground Water Quality BIS dataset for years 2010 to 2018 for each district in each state of the country.

The key components of the dataset include-

1. Latitude
2. Longitude
3. District
4. State
5. Year
6. Value of different water quality parameters like pH, Total Hardness

2.1 Segregating year wise data:

The initial dataset contained data for each year from 2010 to 2018. We separated the data for each year to handle them separately.

2.2 Handling Missing Values:

In the dataset, some values are identified by the string 'ND' or 'BDL'. These entries are replaced with a numeric value of 0. The rows with null values were dropped. This replacement is critical because it transforms the dataset into a consistent numeric format, which is required for calculations and analysis. This ensures that string data doesn't interfere with the analytical workflow, allowing statistical or arithmetic operations to proceed smoothly.

2.3 Data Type Conversion:

After addressing missing values, we converted specific columns from string representations to numeric data types (e.g., integers or floats). This conversion allows for proper numeric processing, such as averaging, summing, or any operation requiring numerical values. Without this step, any attempt at numerical computation on these columns would raise errors or produce incorrect results, as strings cannot be directly used in arithmetic operations.

Chapter 3

Estimation of weights

3.1 Strategy:

In order to estimate weights for the water quality index model one can use the following strategies-

3.1.1 Asking experts

We can consult a multitude of water quality experts and ask them for relative importance of each water parameter. Then we can just take an average for each parameter.
The problem with this method is that unless we have access to a large number of experts the weights might be too subjective. So we have used a more objective way of getting weights using a machine learning algorithm.

3.1.2 Using machine learning algorithms

We have trained a neural network classifier to predict whether the water quality parameters correspond to fit/unfit for drinking water. Then we used model interpretation techniques to get the weights for each of the parameters.

3.2 Training the Neural Network

3.2.1 Getting the training data

We started with the raw dataset. We focused on nine critical parameters for water quality assessment: pH, Total Hardness (TH), Calcium (CA), Magnesium (MG), Chloride, Sulphate, Nitrate, Fluoride, and Total Dissolved Solids (TDS).

3.2.2 Cleaning the dataset

To handle missing values, non-detectable (ND) and below-detection-limit (BDL) entries were replaced with zeros. All selected parameters were converted to numeric types, with errors handled by setting non-convertible values to NaN. Subsequently, rows with any NaN values were removed to ensure a complete dataset for model training.

3.2.3 Labeling the dataset

Acceptable ranges for each parameter were defined according to BIS standards -

Table 3.1: BIS Standard Ranges for Water Quality Parameters

Parameter	**Acceptable Range**	**Unit**
pH	6.5–8.5	–
Total Hardness (TH)	≤ 200	mg/L
Calcium (CA)	≤ 75	mg/L
Magnesium (MG)	≤ 30	mg/L
Chloride	≤ 250	mg/L
Sulphate	≤ 200	mg/L
Nitrate	≤ 45	mg/L
Fluoride	≤ 1	mg/L
Total Dissolved Solids (TDS)	≤ 500	mg/L

If all values for a sample were within the specified ranges, it was marked as "healthy" (1); otherwise, it was labeled "unhealthy" (0).

3.2.4 Training the neural network

The features (X) and target (y) were separated, with X containing water quality parameters and y the binary label (1 for healthy, 0 for unhealthy). Then the data was split into training, validation, and test sets in an 80/20 split, with further subdivision of the training set to ensure balanced classes across all sets. We used a deep neural network.

Deep Neural Network

A Deep Neural Network (DNN) is a type of artificial neural network that contains multiple layers of nodes (neurons) between the input and output layers. DNNs are capable of learning complex patterns in data and are widely used in various applications, including image recognition, natural language processing, and more. Here's a detailed overview of DNNs, including their architecture, training process, activation functions, and applications. Components of a DNN -

1. Input Layer: The first layer that receives the input data. Each node in this layer corresponds to one feature of the input.
2. Hidden Layers: These layers lie between the input and output layers. A DNN has multiple hidden layers, allowing it to learn increasingly abstract representations of the input data. Each hidden layer consists of several neurons that apply transformations to the data.
3. Output Layer: The final layer that produces the output. The structure of this layer depends on the specific task (e.g., classification, regression).

Model Architecture

A neural network was built using the Keras Sequential model with the following structure: Input Layer (64 neurons) -> Dense Layer 1 (32 neurons) -> Dense Layer 2 (16 neurons) -> Dense Layer 3 (8 neurons) -> Output Layer (1 neuron)

1. An input layer with 64 neurons and ReLU activation.
2. Multiple dense layers capture non-linear interactions in the data.
3. An output layer with a sigmoid activation function for binary classification.

The model was compiled with binary cross-entropy as the loss function and the Adam optimizer. The model was trained on the training set with validation against a separate validation set to monitor performance. After training, the model's accuracy was evaluated on the test set, providing insight into how well the classifier generalizes to unseen data. The accuracy achieved on test data was 94%.

3.2.5 Interpretation of model weights

We looked at two model interpretation techniques.

Layer-wise Relevance Propagation for Neural Network Interpretation

Layer-wise Relevance Propagation (LRP) is a technique used to interpret neural networks by identifying which parts of the input contribute most to the model's output. LRP works by backtracking through the layers of the neural network, propagating the "relevance" from the output back to the input features. This approach helps in making the model's decisions more transparent.

Key Concepts in LRP

1. **Relevance Scores**:

   - At the output layer, the network's prediction is assigned an initial relevance score. For classification, this score corresponds to the confidence or probability of the predicted class.
   - The goal is to distribute this relevance score backward through each layer, assigning relevance to each neuron in proportion to its contribution to the final prediction.

2. **Propagation Rules**:

   - LRP applies specific propagation rules to distribute the relevance score backward from layer to layer.
   - A commonly used rule is the **LRP-ϵ rule**, where relevance R_j at a neuron j in layer l is propagated to its input neurons i in the previous layer $l-1$ based on weights w_{ij} and activations:
   $$R_i = \sum_j \frac{x_i w_{ij}}{\sum_k x_k w_{kj} + \epsilon \,\mathrm{sign}\left(\sum_k x_k w_{kj}\right)} R_j$$
   - This rule helps to distribute relevance more evenly, especially in cases where some neurons contribute very little or zero to the output.

3. **Layer-by-Layer Relevance Backtracking**:

   - Each layer in the network receives the relevance score from the next layer, propagating it down to the input layer. This process attributes importance to individual input features by assigning them relevance scores.

4. **Interpretation of Results**:

   - After relevance scores are assigned to the input features, they can be visualized as a heatmap, highlighting which features are most important for the model's decision.
   - For instance, in a water quality classification model, LRP can show which parameters (e.g., pH, hardness) influenced the classification of a sample as healthy or unhealthy, providing insights into feature importance.

Shapley Values for Model Interpretation

Shapley values, originating from cooperative game theory, offer a principled method for attributing contributions of individual features to the output of a model. This approach calculates the importance of each feature by considering it as a "player" in a cooperative game, where the "payoff" is the model's prediction.

Key Concepts in Shapley Values

1. **Coalition of Features**: In the context of Shapley values, we treat the features of a model as cooperative players in a game where each subset (or coalition) of features can contribute to the model's output. By assessing each feature's role across various subsets, we gain a holistic view of its contribution.

2. **Marginal Contribution**: For any given feature, the Shapley value represents the feature's average contribution across all possible subsets of features in the model. Specifically, the marginal contribution of a feature x_i to a subset S is calculated as the difference in the model's prediction when x_i is added to S versus when it is excluded. Mathematically, for a model f with input features $\{x_1, x_2, \ldots, x_n\}$, the marginal contribution of feature x_i to subset S is:

$$f(S \cup \{i\}) - f(S)$$

   where $f(S)$ represents the model's output when only features in subset S are included.

Advantages of Shapley Values

1. **Game-Theoretic Foundation:** Shapley values are derived from cooperative game theory, providing a solid mathematical basis for fair value allocation among players (features). This foundation ensures a principled approach to interpretation.

2. **Robustness to Changes in the Model:** The Shapley value framework is robust to changes in the model or data distribution, meaning that the explanations derived remain reliable even when models are updated or retrained.

We have prepared the indices based on the weights given by Shapley values. The final weights that we got for each parameter were -

1. **pH -** 0.03176855
2. **TH -** 0.58346045
3. **CA -** 0.03082074
4. **MG -** 0.05622624
5. **Chloride -** 0.02997204
6. **Sulphate -** 0.01809105
7. **Nitrate -** 0.07043174
8. **Fluoride -** 0.05326117
9. **TDS -** 0.12596802

Chapter 4

Preparing Sub-Indices

4.1 Strategy

We first introduce **BIS** limits for various water parameters-

Table 4.1: BIS Limits for Water Quality Parameters

Parameter	**Acceptable Limit**	**Permissible Limit**
pH	6.5 - 8.5	No relaxation
Total Hardness (mg/L)	200	600
Calcium (mg/L)	75	200
Magnesium (mg/L)	30	100
Chloride (mg/L)	250	1000
Sulphate (mg/L)	200	400
Nitrate (mg/L)	45	No relaxation
Fluoride (mg/L)	1.0	1.5
Total Dissolved Solids (mg/L)	500	2000

We assign a score from 100 to 0 in the following way -

$$S_i = \begin{cases} 100 & \text{if } V \leq V_{\text{acceptable}} \\ \frac{V_{\text{permissible}} - V}{V_{\text{permissible}} - V_{\text{acceptable}}} \times 100 & \text{if } V_{\text{acceptable}} < V \leq V_{\text{permissible}} \\ 0 & \text{if } V > V_{\text{permissible}} \end{cases}$$

1. We assign a score of 100 if the concentration lies within the acceptable limits.

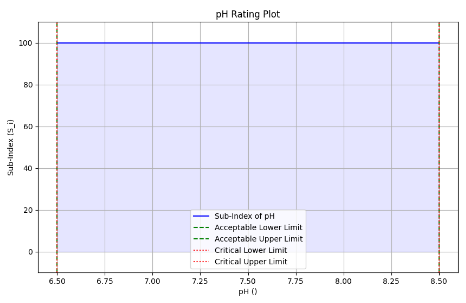
2. Otherwise we do a linear decrease from 100 to 0 such that outside the permissible limit the value of the sub index for that water parameter is 0.

4.2 Example for sub index calculation

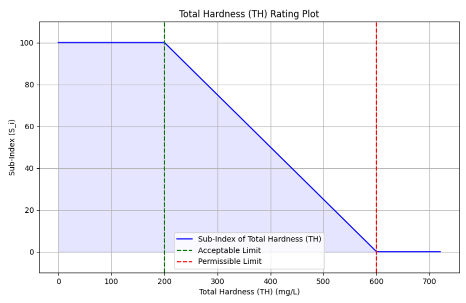
Take for example pH. Suppose the observed value of pH is 7.0. Then its subindex can be calculated as -

$$S_{\text{pH}} = \frac{V_{\text{permissible}} - V}{V_{\text{permissible}} - V_{\text{acceptable}}} \times 100 = \frac{8.5 - 7.0}{8.5 - 6.5} \times 100 = \frac{1.5}{2.0} \times 100 = 75$$

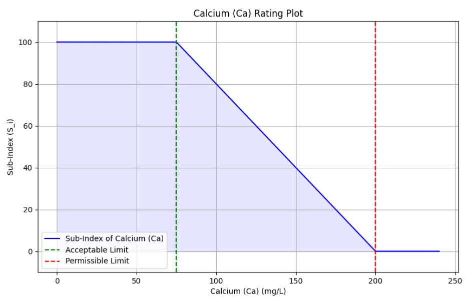
4.3 Plots for sub indices for various parameters



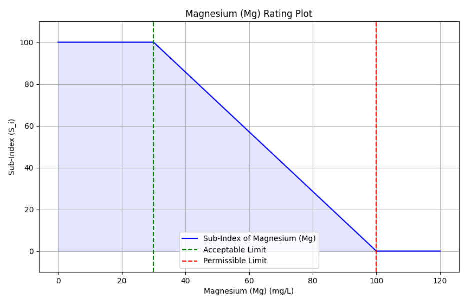
(a) pH



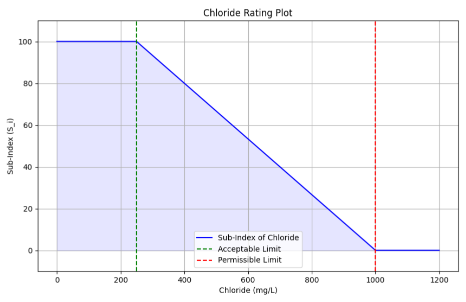
(b) Total Hardness (TH)



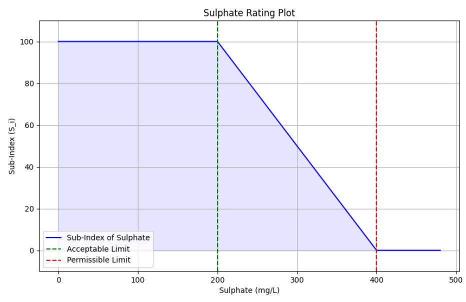
(c) Calcium (Ca)



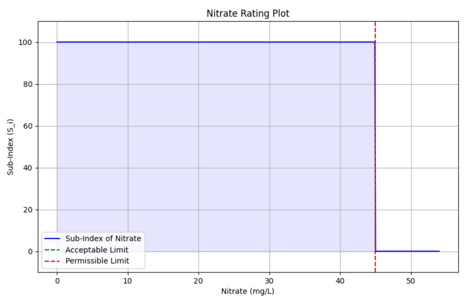
(d) Magnesium (Mg)



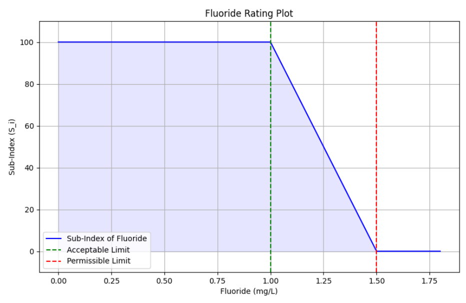
(e) Chloride



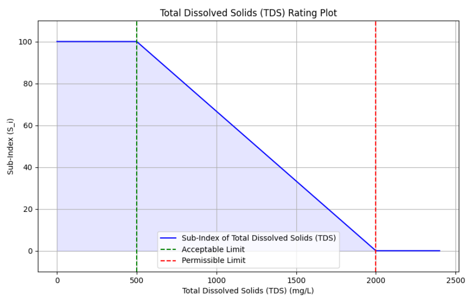
(f) Sulfate



(g) Nitrate



(h) Fluoride



(a) Total Dissolved Solids (TDS)

Figure 4.2: Rating Plots for Various Water Quality Parameters Based on BIS Limits

Chapter 5

Getting Final Indices

5.1 Calculating the final index

From the previous sections we have got the weights as well as the sub indices. Now using the additive model we calculate-

$$\text{AWQI} = \sum_{i=1}^{n} w_i \cdot S_i \tag{5.1}$$

This gives us the final index value for each location for each year.

5.2 Plotting the final indices on the Indian Map

We used the following python libraries for this purpose -

1. **Numpy and Pandas:** For Data Handling
2. **Geopandas:** For handling geographical data
3. **Matplotlib:** For Plotting
4. **re:** For converting coordinates into a consistent format

Step 1 **Coordinate Conversion - DMS to Decimal Degrees:** The given latitude and longitude were in mixed format. Some were in DMS others were in decimals. We converted all the coordinates into decimal format for efficient plotting.

Step 2 **Loading Shapefiles and Setting Coordinate System:** Loaded the shapefiles for country and state boundaries assuming the coordinate system to be **EPSG:4326**.

Step 3 **Plotting the Map:** Using matplotlib plotted a Scatter plot on the Indian Map. A color bar was also added for easy interpretation.

The final plot for the years can be seen in the following pages, but we make some remarks first -

5.3 Remarks

1. For years 2010 - 2016 we didn't have **TDS** data with us. So the weights were adjusted for the remaining 8 coordinates.
2. In maps it can be seen that there was no data available for certain states. This was because either there was no data for that place or the data had missing values and hence no index could be calculated for that place.
3. The data for the year 2017 was very inconsistent, with several fields being null. This resulted in very few data points and hence no plot is being shown for 2017.

5.4 Plots

5.4.1 Plot for 2010

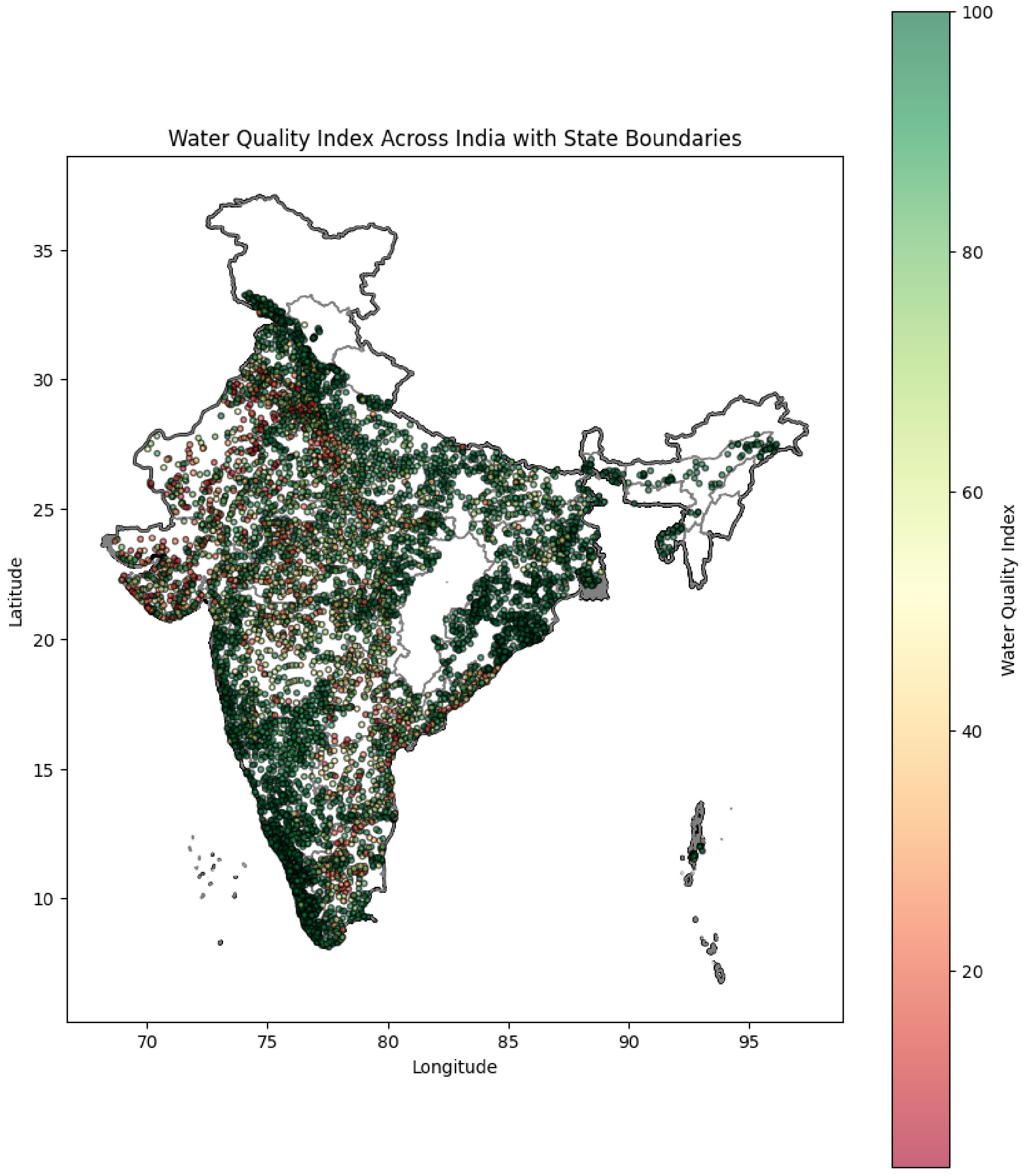


Figure 5.1: Water quality index in 2010.

5.4.2 Plot for 2011

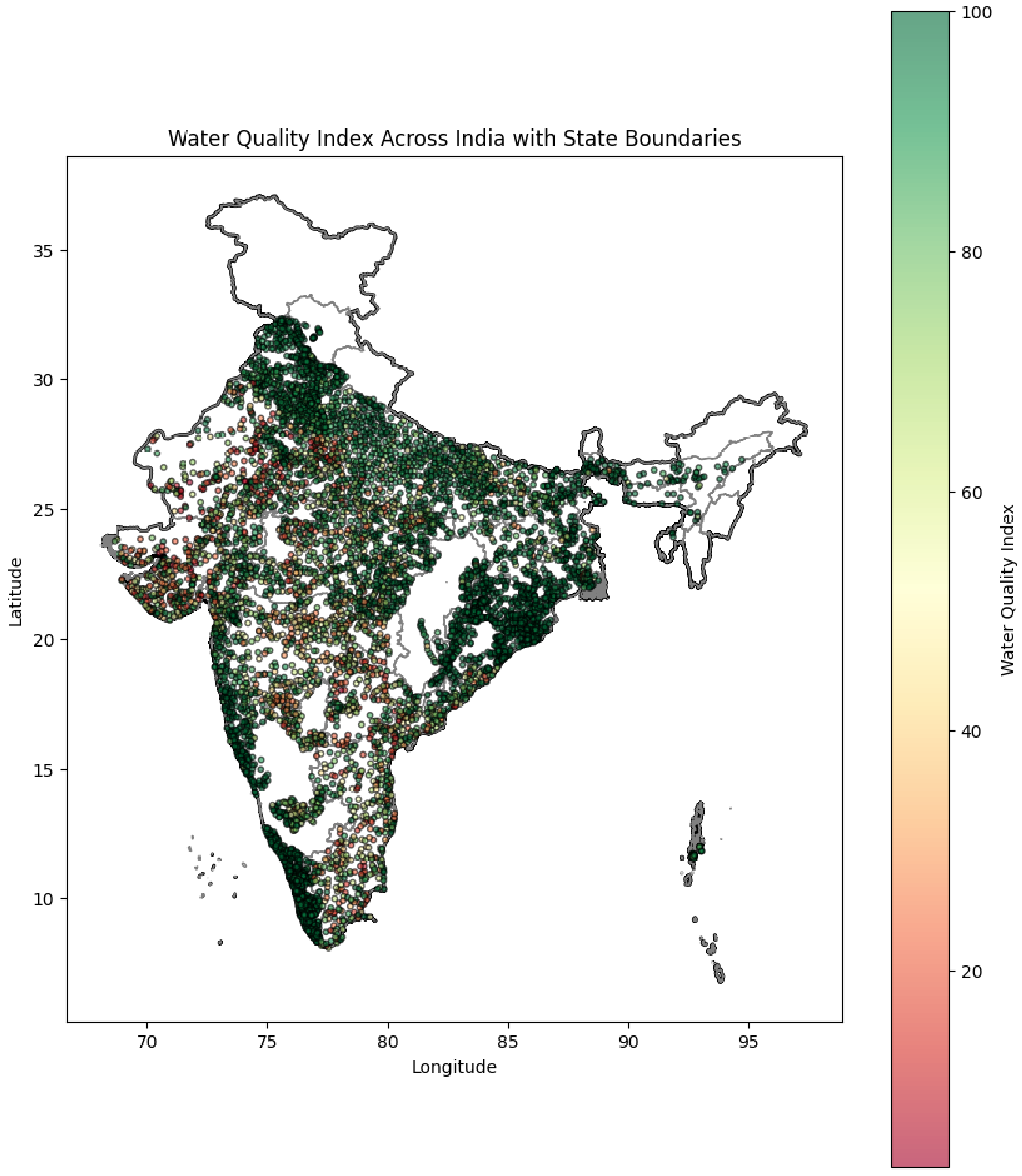


Figure 5.2: Water quality index in 2011.

5.4.3 Plot for 2012

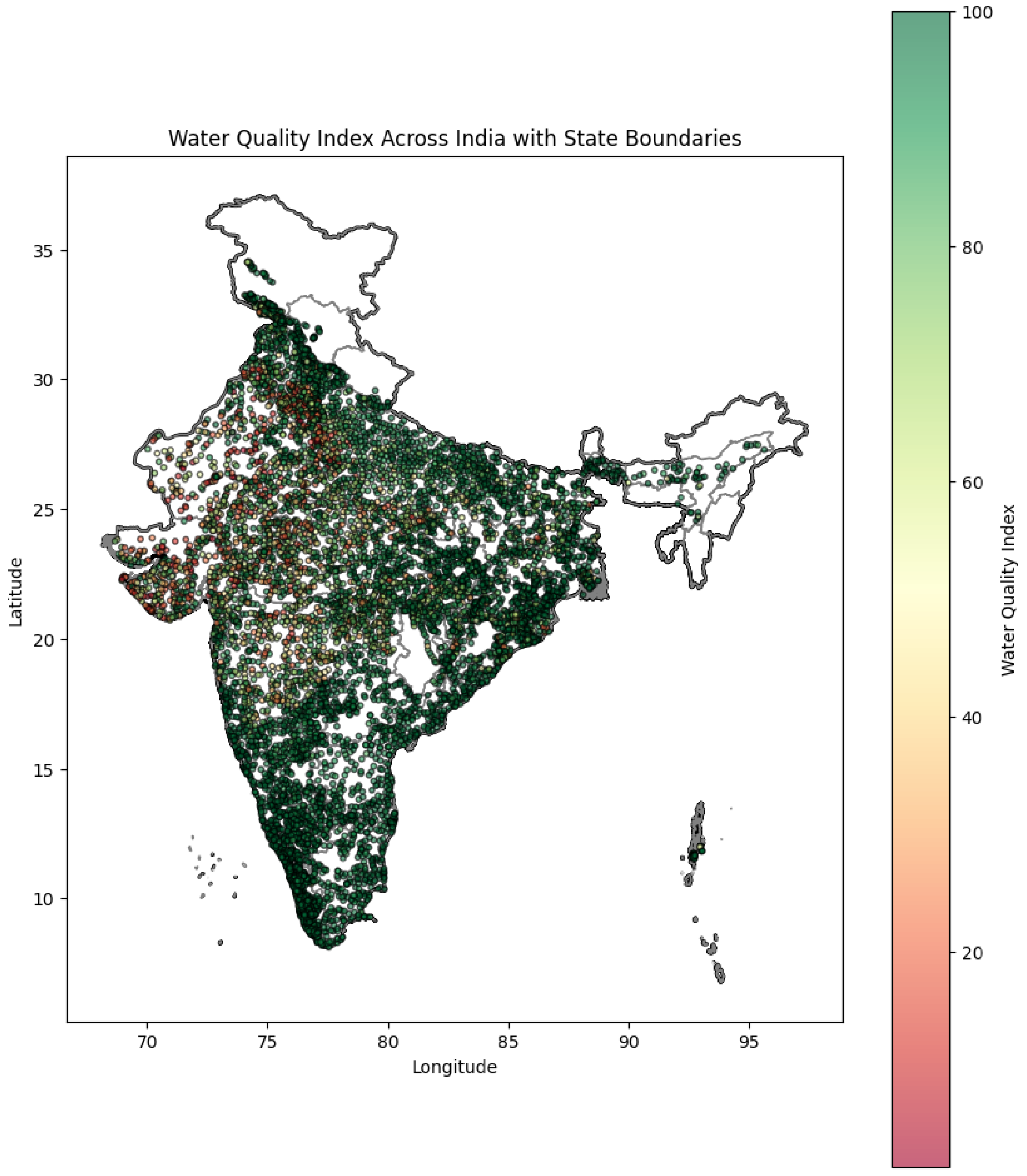


Figure 5.3: Water quality index in 2012.

5.4.4 Plot for 2013

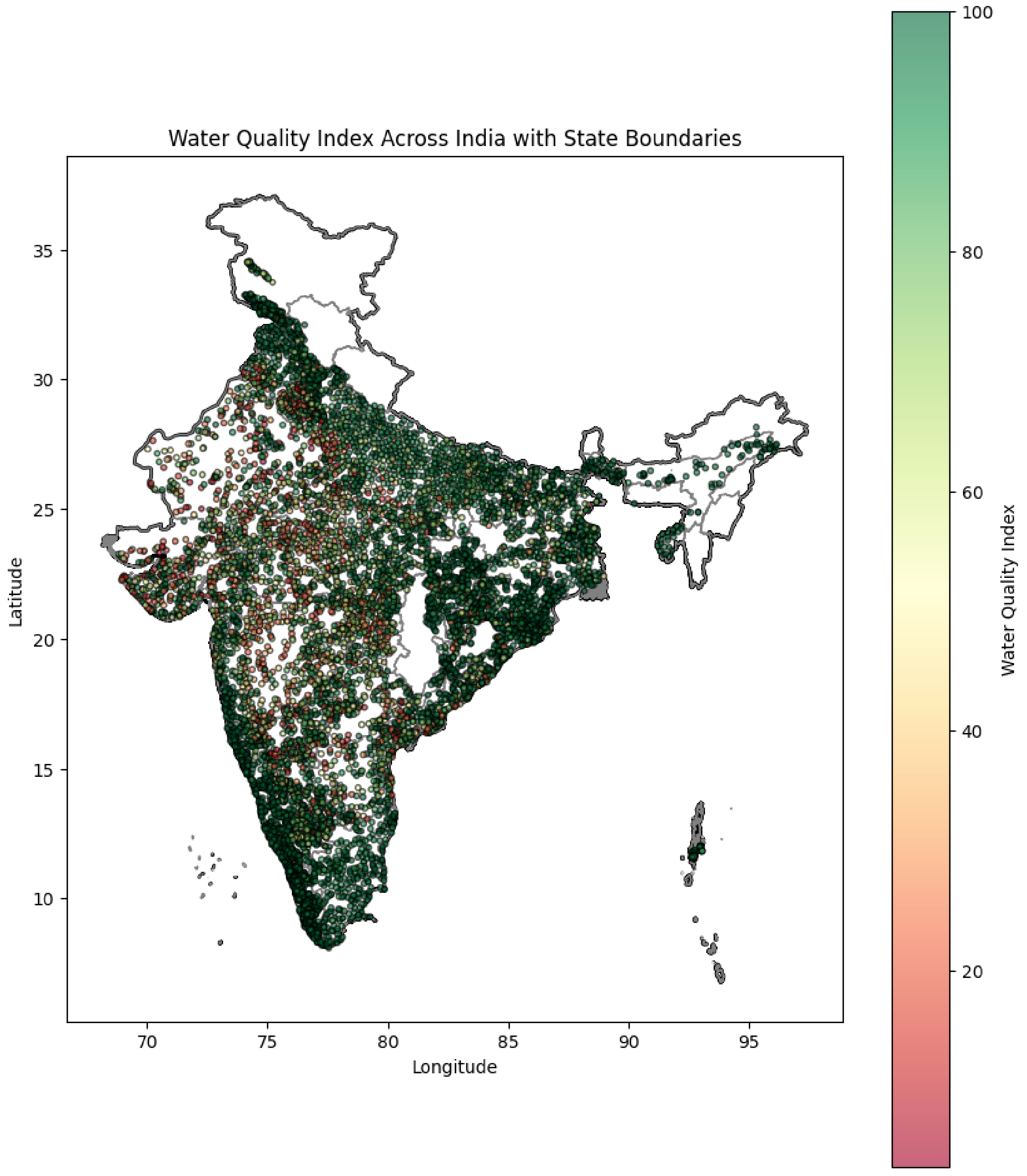


Figure 5.4: Water quality index in 2013.

5.4.5 Plot for 2014

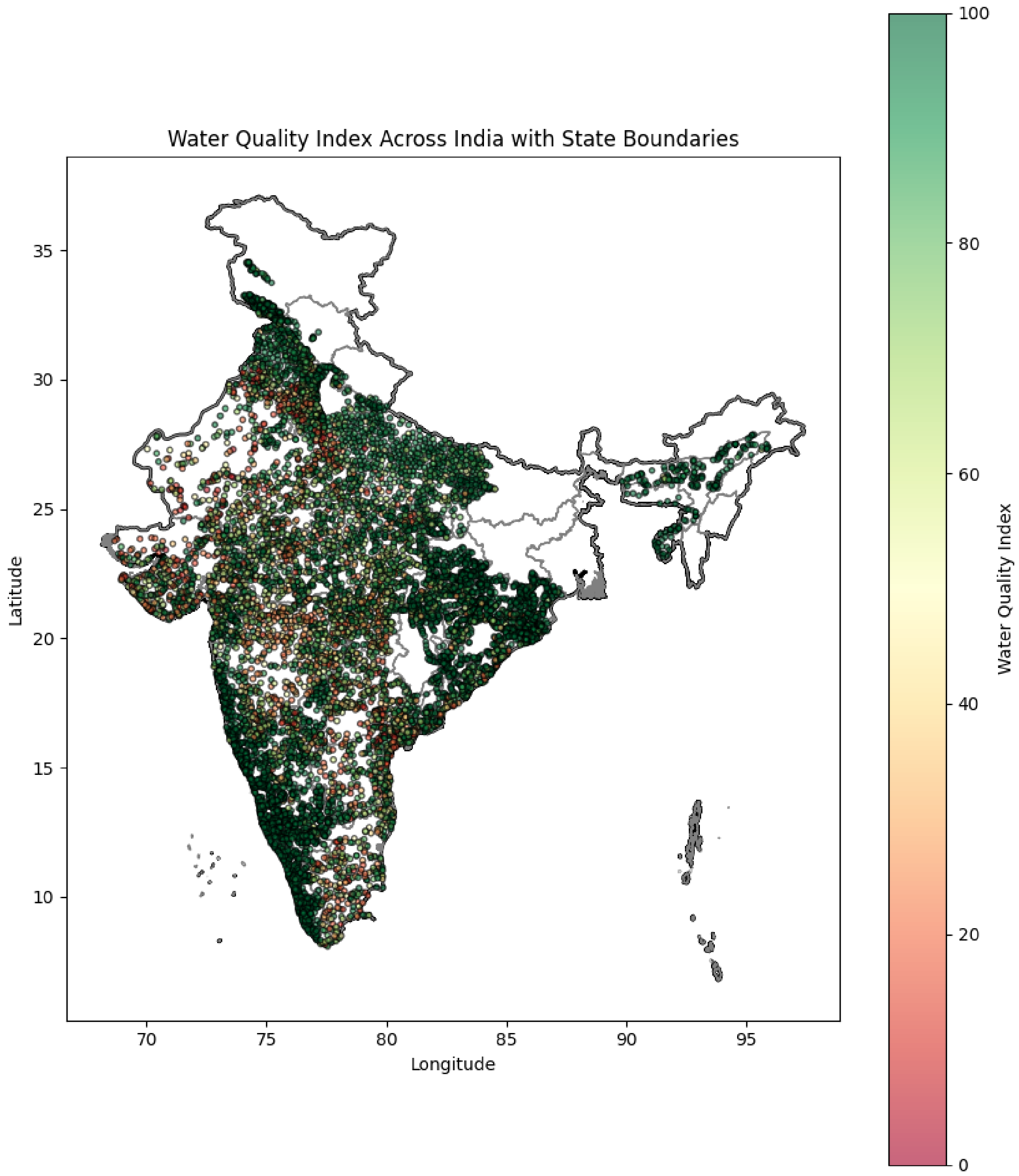


Figure 5.5: Water quality index in 2014.

5.4.6 Plot for 2015

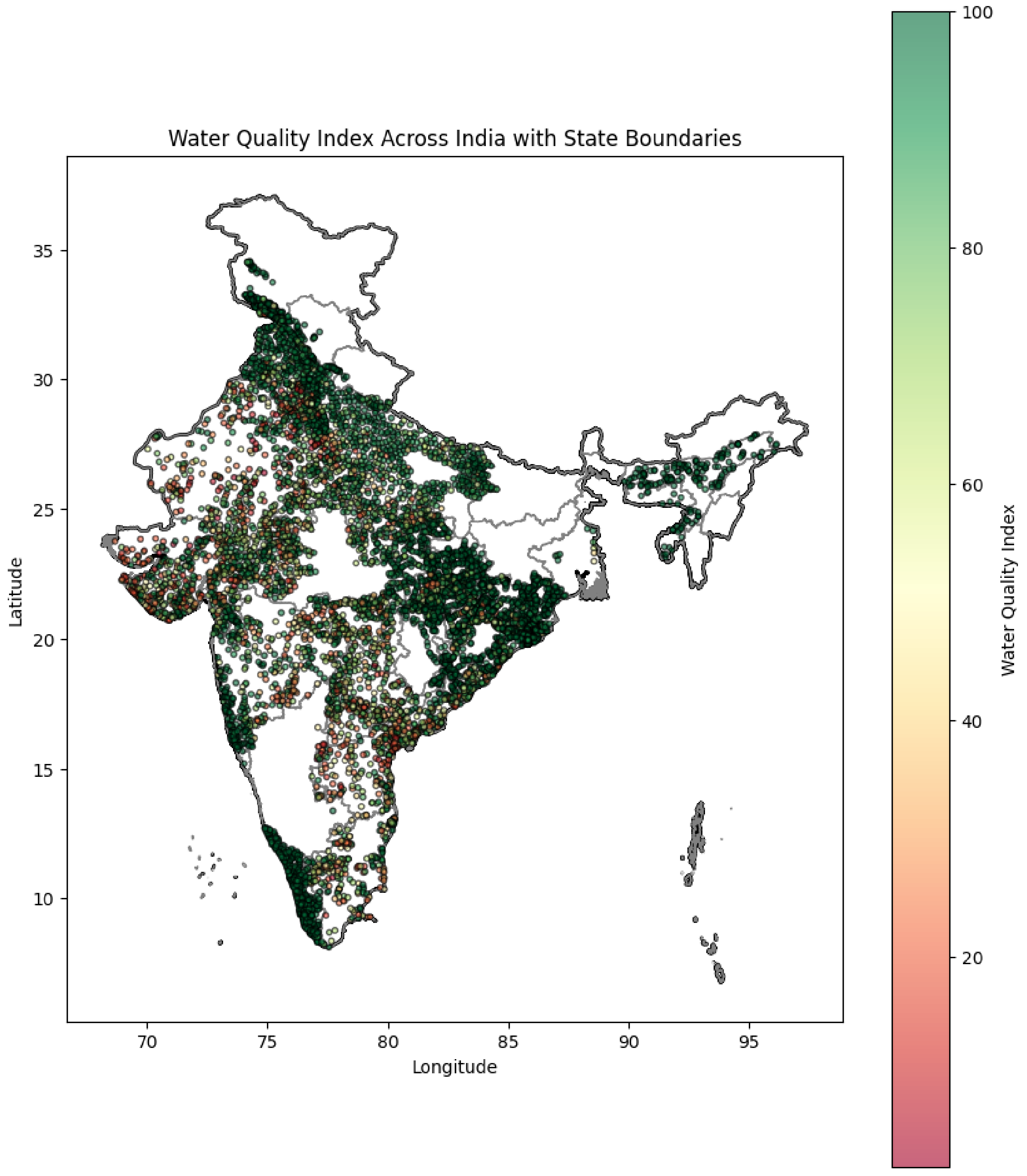


Figure 5.6: Water quality index in 2015.

5.4.7 Plot for 2016

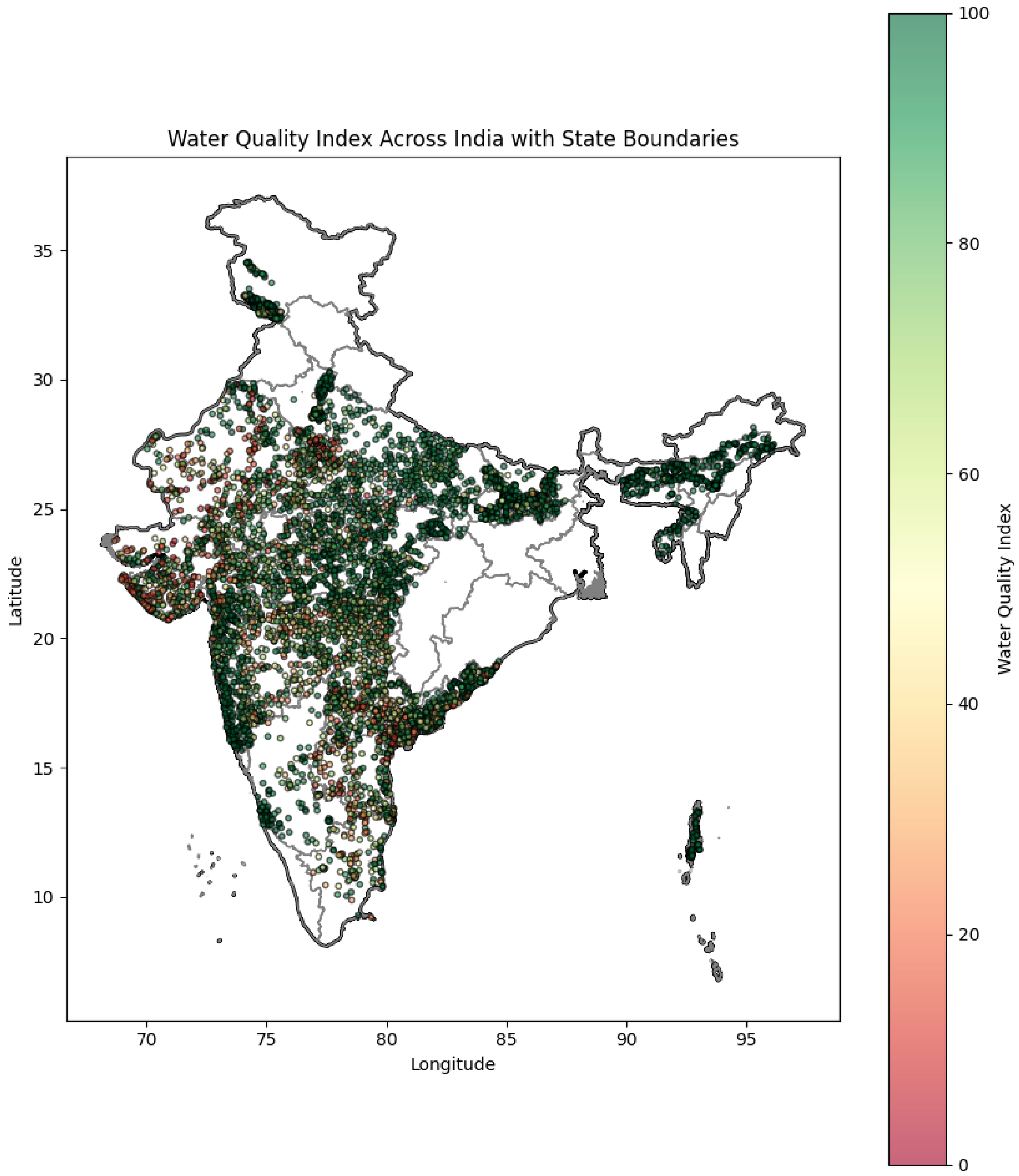


Figure 5.7: Water quality index in 2016.

5.4.8 Plot for 2018

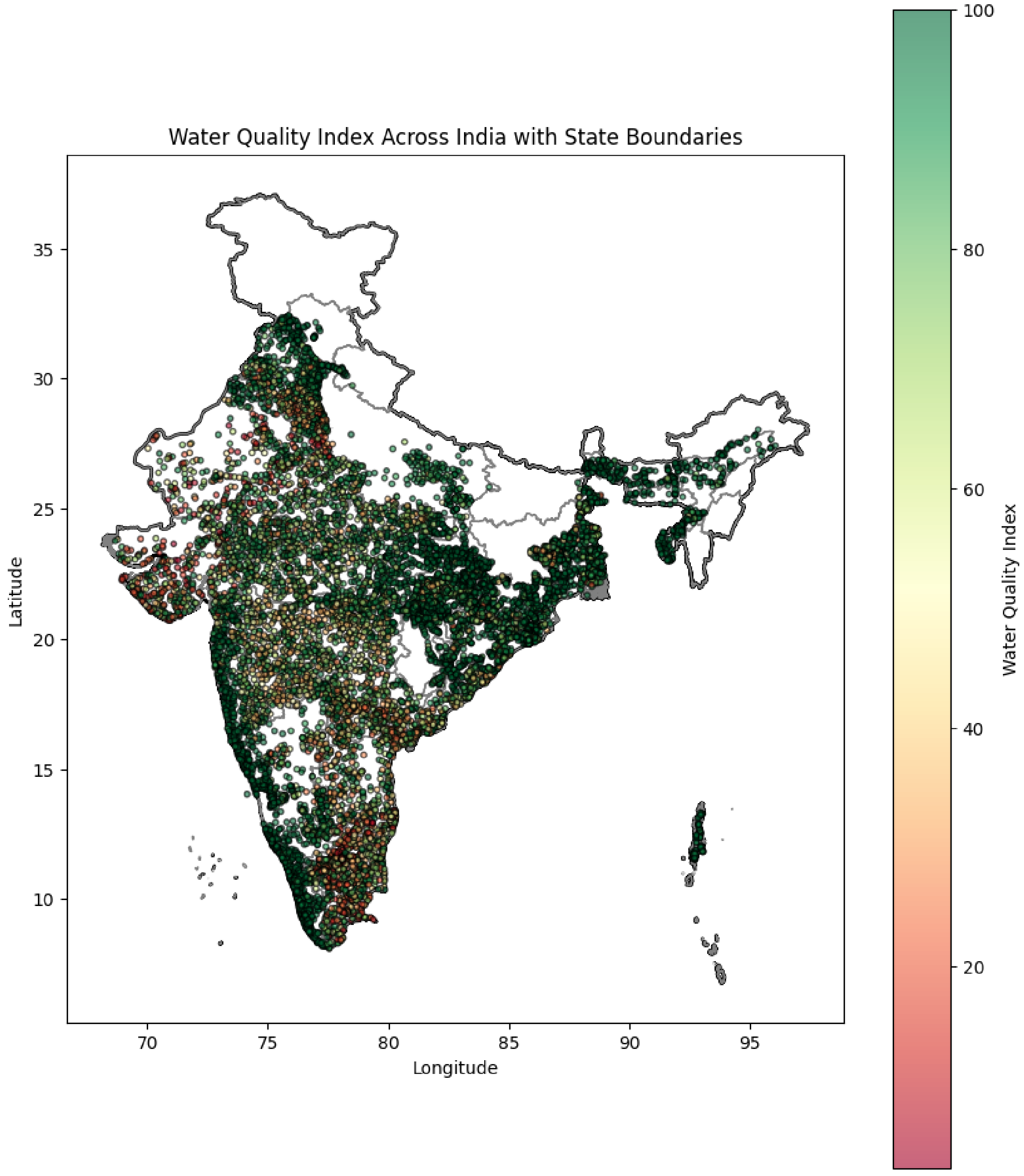


Figure 5.8: Water quality index in 2018.

Bibliography

1. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K., & Samek, W. (2015). On Pixel-Wise explanations for Non-Linear Classifier decisions by Layer-Wise relevance propagation. PLoS ONE, 10(7), e0130140. https://doi.org/10.1371/journal.pone.0130140 North American Actuarial Journal, Volume 3, Number 2, April 1999

2. Lumb, A., Sharma, T. C., Bibeault, J., & Klawunn, P. (2011). A Comparative Study of USA and Canadian Water Quality Index Models. Water Quality Exposure and Health, 3(3–4), 203–216. https://doi.org/10.1007/s12403-011-0056-5 Martin Odening and Jan Hinrichs

3. Molnar, C. (2024, July 31). 9.5 Shapley Values Interpretable Machine Learning. https:// christophm.github.io/ interpretabl-eml book/shapley.htmlKlara Andersson, May 2020

4. Ding, F., Zhang, W., Cao, S., Hao, S., Chen, L., Xie, X., Li, W., & Jiang, M. (2023). Optimization of water quality index models using machine learning approaches. Water Research, 243, 120337. https://doi.org/10.1016/j.watres.2023.120337

Appendix

- Code for training the classifier
- Code for getting weights using SHAP
- Code for getting sub indices
- Code for generating plots

Snippets of Python Codes

Code for training the network

```python
# Import necessary libraries
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

# Import Sequential model and layer components from Keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

# Load the training data from a CSV file
data = pd.read_csv('/content/training_data.csv')
# Display the first few rows of the dataset
print(data.head())

# Split the dataset into features (X) and target variable (y)
X = data.drop('Health_Status', axis=1)  # Features (all columns except 'Health_Status')
y = data['Health_Status']  # Target variable (Health_Status)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Further split the training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25,
random_state=42, stratify=y_train)

# Print the shapes of the training, validation, and test datasets
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)

# Define the neural network model architecture
model = Sequential([
    Dense(64, input_shape=(9,)),  # First hidden layer with 64 units and input shape of 9 features
    Activation('relu'),           # Activation function for the first layer
    Dense(32),                    # Second hidden layer with 32 units
    Activation('relu'),           # Activation function for the second layer
    Dense(16),                    # Third hidden layer with 16 units
    Activation('relu'),           # Activation function for the third layer
    Dense(8),                     # Fourth hidden layer with 8 units
    Activation('relu'),           # Activation function for the fourth layer
    Dense(1),                     # Output layer with 1 unit for binary classification
    Activation('sigmoid')         # Sigmoid activation function for binary output
])

# Compile the model with optimizer, loss function, and evaluation metric
model.compile(optimizer='adam',              # Optimizer used for training
              loss='binary_crossentropy',    # Loss function for binary classification
              metrics=['accuracy'])          # Metric to evaluate the model performance
```

```
# Train the model on the training data
history = model.fit(
    X_train, y_train,
    epochs=100,                          # Number of training epochs
    batch_size=32,                       # Number of samples per gradient update
    validation_data=(X_val, y_val),      # Validation data for monitoring performance
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True)]
# Early stopping callback
)

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
# Print the test loss and accuracy
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

Code for calculating weights using SHAP

```
import numpy as np
import shap

# Define the size of the background dataset for SHAP value computation
background_size = 100

# Randomly select indices from the training set to create a background dataset
random_indices = np.random.choice(X_train.shape[0], background_size, replace=False)
# Create the background dataset using the selected random indices
background = X_train.iloc[random_indices].values

# Initialize the DeepExplainer with the trained model and background data
explainer = shap.DeepExplainer(model, background)

# Convert the test features DataFrame to a NumPy array
X_test_np = X_test.values

# Compute SHAP values for the test dataset
shap_values = explainer.shap_values(X_test_np)

# Print the shape of the SHAP values array
print("SHAP values shape:", shap_values.shape)  # Expected shape: (num_samples, num_features)

# Print the SHAP values for the first test sample
print(shap_values[0])

# Calculate the average absolute SHAP values for each feature
avg_shap = np.mean(np.abs(shap_values), axis=0)

# Print the average absolute SHAP values
print(avg_shap)

# Normalize the average SHAP values to sum to 1 for better interpretability
normalized_relevance = avg_shap / np.sum(avg_shap)
print(normalized_relevance)

# Assuming `lrp_val` is defined elsewhere in your code; compute the absolute values
of LRP (Layer-wise Relevance Propagation)
lrp_val_abs = np.abs(lrp_val)
# Normalize the LRP values to sum to 1
normalized_lrp = lrp_val_abs / np.sum(lrp_val_abs)
# Print the normalized LRP values
print(normalized_lrp)
```

Code for getting sub indices

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

# Load the water quality data from a CSV file
data = pd.read_csv('/content/India_2017.csv')
# Display the first few rows of the dataset
print(data.head())

# Define Bureau of Indian Standards (BIS) limits for various water quality parameters
# pH: 6.5 to 8.5, no relaxation
# TH: 200 (acceptable) to 600 (permissible)
# CA: 75 (acceptable) to 200 (permissible)
# MG: 30 (acceptable) to 100 (permissible)
# Cl: 250 (acceptable) to 1000 (permissible)
# Sulphate: 200 (acceptable) to 400 (permissible)
# Nitrate: 75, no relaxation
# Fluoride: 1 (acceptable) to 1.5 (permissible)
# TDS: 500 (acceptable) to 2000 (permissible)

# Check for missing values in the dataset
print(np.sum(data.isna()))

# Replace 'ND' and 'BDL' with 0 in the dataset
data.replace(['ND', 'BDL'], 0, inplace=True)

# Exclude columns that should not be converted to numeric
exclude_columns = ['LATITUDE', 'LONGITUDE']
# Convert remaining columns to numeric, coercing errors to NaN
data[data.columns.difference(exclude_columns)] = data[data.columns.difference(exclude_columns)]
.apply(pd.to_numeric, errors='coerce')

# Check for missing values again after conversion
print(np.sum(data.isna()))

# Drop any remaining rows with missing values
data = data.dropna()

# Reset the index of the DataFrame after dropping rows
data = data.reset_index(drop=True)

# Function to generate sub-indices based on water quality parameters
def generate_sub_indices(data, sub_indices):
    # Define the columns to analyze
    columns = ['TH', 'CA', 'MG', 'CHLORIDE', 'SULPHATE', 'FLUORIDE', 'TDS']
    # Set acceptable and permissible limits for each parameter
    acceptable_limits = {'TH': 200, 'CA': 75, 'MG': 30, 'CHLORIDE': 250, 'SULPHATE': 200,
    'FLUORIDE': 1, 'TDS': 500}
    permissible_limits = {'TH': 600, 'CA': 200, 'MG': 100, 'CHLORIDE': 1000, 'SULPHATE':
    400, 'FLUORIDE': 1.5, 'TDS': 2000}

    # Initialize a DataFrame to hold sub-indices
    sub_indices = pd.DataFrame(index=data.index)

    # Calculate sub-index for pH
    sub_indices['PH'] = 0.0
    for i, value in enumerate(data['PH']):
        if value >= 6.5 and value <= 8.5:
            sub_indices['PH'].at[i] = 100  # Assign full score if within acceptable range
        else:
            sub_indices['PH'].at[i] = 0    # Assign zero score otherwise

    # Calculate sub-indices for each chemical parameter
    for col in columns:
        sub_indices[col] = 0.0
        for i, value in enumerate(data[col]):
            if value <= acceptable_limits[col]:
                sub_indices[col].at[i] = 100  # Full score if within acceptable limit
            elif value > permissible_limits[col]:
                sub_indices[col].at[i] = 0    # Zero score if above permissible limit
            else:
                # Calculate score proportionally between acceptable and permissible limits
                sub_indices[col].at[i] = (permissible_limits[col] - value) /
                (permissible_limits[col] - acceptable_limits[col]) * 100

    # Calculate sub-index for Nitrate
    sub_indices['NITRATE'] = 0.0
    for i, value in enumerate(data['NITRATE']):
        if value <= 75:
            sub_indices['NITRATE'].at[i] = 100  # Full score if within acceptable limit
        else:
            sub_indices['NITRATE'].at[i] = 0    # Zero score otherwise

    return sub_indices

# Generate sub-indices based on water quality data
sub_indices = generate_sub_indices(data, pd.DataFrame())
# Display the calculated sub-indices
print(sub_indices)

# Save the sub-indices DataFrame to a temporary CSV file
sub_indices.to_csv('temp2.csv', index=False)

# Display the shape of the sub-indices DataFrame
print(sub_indices.shape)

# Reorder the columns for better organization
new_order = ['PH', 'TH', 'CA', 'MG', 'CHLORIDE', 'SULPHATE', 'NITRATE', 'FLUORIDE']
sub_indices = sub_indices[new_order]
print(sub_indices)

# Add latitude and longitude to the sub-indices DataFrame
sub_indices['LATITUDE'] = data['LATITUDE']
sub_indices['LONGITUDE'] = data['LONGITUDE']
# Finalize the column order
new_order = ['LATITUDE', 'LONGITUDE', 'PH', 'TH', 'CA', 'MG', 'CHLORIDE', 'SULPHATE',
'NITRATE', 'FLUORIDE']
sub_indices = sub_indices[new_order]
print(sub_indices)

# Define columns to round to two decimal places
cols_to_round = ['PH', 'TH', 'CA', 'MG', 'CHLORIDE', 'SULPHATE', 'FLUORIDE', 'NITRATE']

# Round the specified columns to 2 decimal places
sub_indices[cols_to_round] = sub_indices[cols_to_round].round(2)

# Define shap weights for calculating the AWQI (Aggregate Water Quality Index)
shap_weights = {
    'PH': 0.03176855,
    'TH': 0.58346045,
    'CA': 0.03082074,
    'MG': 0.05622624,
    'CHLORIDE': 0.02997204,
    'SULPHATE': 0.01809105,
    'NITRATE': 0.07043174,
    'FLUORIDE': 0.05326117,
    'TDS': 0.12596802
}

# Define the columns to use for AWQI calculation
cols = ['PH', 'TH', 'CA', 'MG', 'CHLORIDE', 'SULPHATE', 'NITRATE', 'FLUORIDE']

# Calculate the AWQI by summing the weighted sub-indices
sub_indices['AWQI'] = sum(sub_indices[col] * shap_weights[col] for col in cols)
print(sub_indices)

# Round the AWQI to 2 decimal places
sub_indices['AWQI'] = sub_indices['AWQI'].round(2)
print(sub_indices)
```

```
# Save the final sub-indices DataFrame with AWQI to a CSV file
sub_indices.to_csv('AWQI_India_2017.csv', index=False)
```

Code for plotting the map

```
# Import necessary libraries
import pandas as pd
import re
import numpy as np
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib import colors

# Load the water quality index data from a CSV file
data = pd.read_csv('/content/AWQI_India_2017.csv')
print(data.head())  # Display the first few rows of the dataset

def dms_to_decimal(dms_str, default_direction=None):
    """
    Convert DMS (Degrees, Minutes, Seconds) format to decimal degrees.

    Parameters:
    dms_str (str): The DMS string to convert.
    default_direction (str): Default hemisphere if not provided in the string.

    Returns:
    float: Decimal degree representation of the input DMS string.
    """
    # Regex pattern to match DMS format
    dms_regex = re.match(r"(\d+)(\d+)'([\d.]+)\"? ?([NSEW]?)",␣dms_str.strip())

␣␣␣␣if␣dms_regex:
␣␣␣␣␣␣␣␣degrees␣=␣int(dms_regex.group(1))␣␣␣#␣Extract␣degrees
␣␣␣␣␣␣␣␣minutes␣=␣int(dms_regex.group(2))␣␣␣#␣Extract␣minutes
␣␣␣␣␣␣␣␣seconds␣=␣float(dms_regex.group(3))␣␣#␣Extract␣seconds

␣␣␣␣␣␣␣␣#␣Determine␣the␣direction␣(N,␣S,␣E,␣W)
␣␣␣␣␣␣␣␣direction␣=␣dms_regex.group(4)␣if␣dms_regex.group(4)␣else␣default_direction

␣␣␣␣␣␣␣␣#␣Calculate␣decimal␣degrees
␣␣␣␣␣␣␣␣decimal␣=␣degrees␣+␣minutes␣/␣60␣+␣seconds␣/␣3600

␣␣␣␣␣␣␣␣#␣Negate␣for␣Southern␣and␣Western␣coordinates
␣␣␣␣␣␣␣␣if␣direction␣in␣['S',␣'W']:
␣␣␣␣␣␣␣␣␣␣␣␣decimal␣*=␣-1

␣␣␣␣␣␣␣␣return␣decimal
␣␣␣␣else:
␣␣␣␣␣␣␣␣raise␣ValueError(f"Invalid DMS format: {dms_str}")␣␣#␣Raise␣error␣if␣DMS␣format␣is␣invalid

def␣convert_coordinate(coord_str,␣default_direction):
␣␣␣␣"""
    Convert coordinates to decimal format, checking if it's␣already␣in␣decimal.

␣␣␣␣Parameters:
␣␣␣␣coord_str␣(str):␣The␣coordinate␣string␣to␣convert.
␣␣␣␣default_direction␣(str):␣Default␣hemisphere␣if␣needed.

␣␣␣␣Returns:
␣␣␣␣float:␣Decimal␣representation␣of␣the␣input␣coordinate␣string.
␣␣␣␣"""
␣␣␣␣try:
␣␣␣␣␣␣␣␣#␣If␣it␣can␣be␣converted␣to␣float,␣it's already in decimal
        return float(coord_str)
    except ValueError:
        try:
            # If it's not a float, assume it's in DMS format and convert
            return dms_to_decimal(coord_str, default_direction)
        except ValueError:
            return None  # Return None if conversion fails

# Apply the coordinate conversion functions to latitude and longitude columns
data['Latitude_Decimal'] = data['LATITUDE'].apply(lambda x: convert_coordinate(x, 'N'))
data['Longitude_Decimal'] = data['LONGITUDE'].apply(lambda x: convert_coordinate(x, 'E'))

# Drop rows with NaN values in decimal latitude and longitude
data = data.dropna(subset=['Latitude_Decimal', 'Longitude_Decimal'])

# Display the converted latitude and longitude values
print(data['Latitude_Decimal'])
print(data['Longitude_Decimal'])

# Reset the index of the DataFrame
data = data.reset_index(drop=True)

# Load shapefiles for country and state boundaries
country_boundary_path = '/content/IndiaBoundary.shp'
state_boundary_path = '/content/India-States.shp'

# Read the shapefiles and convert them to the same Coordinate Reference System (CRS)
country_boundary = gpd.read_file(country_boundary_path).to_crs(epsg=4326)
state_boundary = gpd.read_file(state_boundary_path).to_crs(epsg=4326)

# Create a figure and axis for plotting
fig, ax = plt.subplots(figsize=(10, 12))

# Plot the country boundary with specified style
country_boundary.plot(ax=ax, edgecolor="black", linewidth=2, facecolor="none")

# Plot the state boundaries
state_boundary.plot(ax=ax, edgecolor="grey", linewidth=1, facecolor="none")

# Normalize the water quality index values for proportional circle sizing
max_index = data['AWQI'].max()  # Maximum AWQI value
min_index = data['AWQI'].min()  # Minimum AWQI value

# Plot points using scatter and color them based on the AWQI index
scatter = ax.scatter(
    data['Longitude_Decimal'], data['Latitude_Decimal'],
    c=data['AWQI'],  # Color based on water quality index
    s=10,  # Size of the scatter points
    cmap='RdYlGn', alpha=0.6, edgecolor="k"  # Color map and transparency
)

# Add a color bar to the plot for reference
cbar = plt.colorbar(scatter, ax=ax, orientation="vertical")
cbar.set_label("Water Quality Index")  # Label for the color bar

# Set axis labels and title for the plot
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")
ax.set_title("Water Quality Index Across India with State Boundaries")

# Display the plot
plt.show()

# Create a new figure for a simpler map view (country and state boundaries)
fig, ax = plt.subplots(figsize=(10, 12))
country_boundary.plot(ax=ax, edgecolor="black", linewidth=2, facecolor="none")  # Country boundary
state_boundary.plot(ax=ax, edgecolor="grey", linewidth=1, facecolor="none")  # State boundaries
plt.show()  # Display the simpler map
```