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

October 30, 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) is calculated by assigning weights to each water quality parameter and summing their weighted scores. The general formula is given by:

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

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) aggregates water quality parameters by multiplying their weighted scores. This approach emphasizes the compounded effect of multiple parameters deviating from standards. 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 synonymous to some kind of harmonic averaging with equal weights (= 1/n) assigned to square of the each sub-index.

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

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

2.0.1 Dropping Null values:

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. A neural network was built using the Keras Sequential model with the following structure:

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:
$$R_i = \sum_j \frac{x_i w_{ij}}{\sum_k x_k w_{kj} + \epsilon \, \text{sign}\left(\sum_k x_k w_{kj}\right)} R_j$$
- 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.

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.

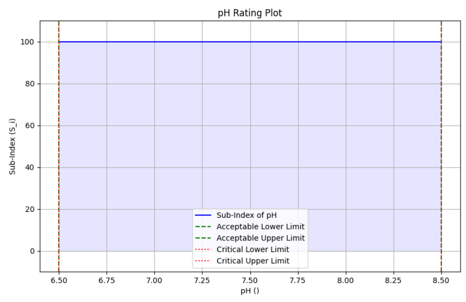
2. Otherwise we do a linear decrease from 100 to 0 such that outside the permissible limit the value of the sub index for that water parameter is 0.

4.2 Example for sub index calculation

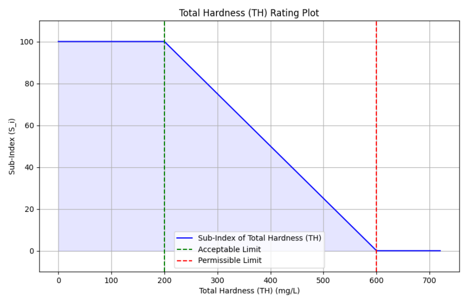
Take for example pH. Suppose the observed value of pH is 7.0. Then its subindex can be calculated as -

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

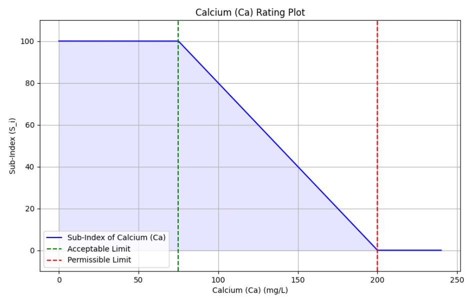
4.3 Plots for sub indices for various parameters



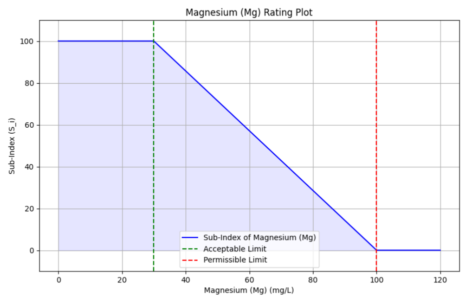
(a) pH



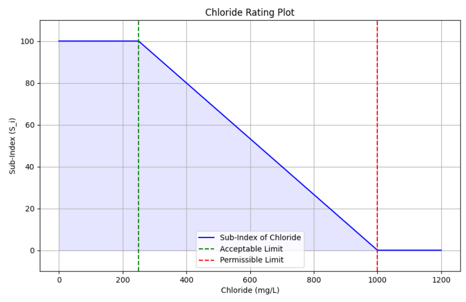
(b) Total Hardness (TH)



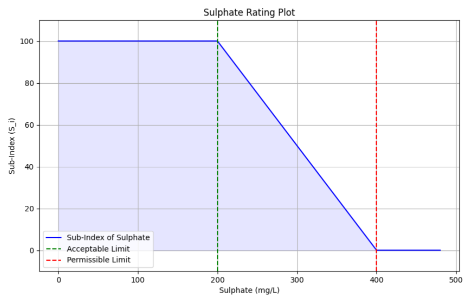
(c) Calcium (Ca)



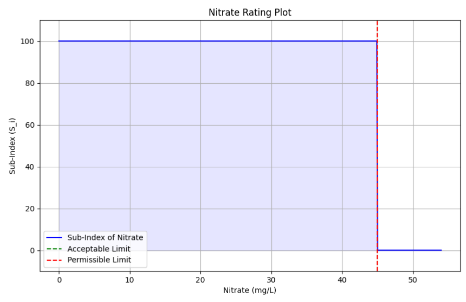
(d) Magnesium (Mg)



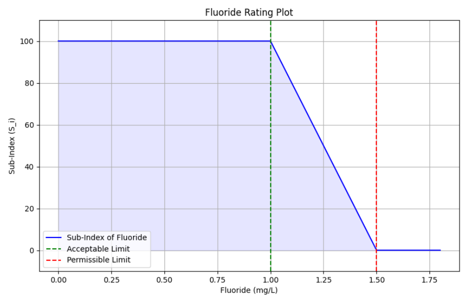
(e) Chloride



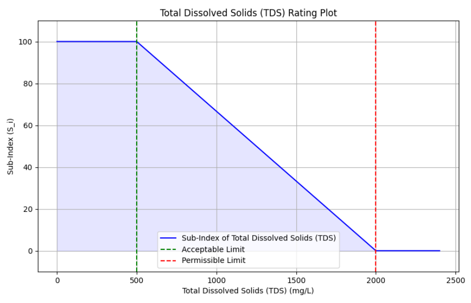
(f) Sulfate



(g) Nitrate



(h) Fluoride



(a) Total Dissolved Solids (TDS)

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

Bibliography

1. “Extreme Value Theory as a Risk Management Tool” By Paul Embrechts, Sidney I. Resnick, and Gennady Samorodnitsky North American Actuarial Journal, Volume 3, Number 2, April 1999
2. "Using Extreme Value Theory to Estimate Value-at-Risk" Martin Odening and Jan Hinrichs
3. A comparative study of VaR and ES using extreme value theory Klara Andersson, May 2020
4. Bonate, Peter. (2021). The application of extreme value theory to pharmacometrics. Journal of pharmacokinetics and pharmacodynamics. 48. 10.1007/s10928-020-09721-0.
5. "Fisher–Tippett–Gnedenko theorem" - Wikipedia
6. "RANDOM FORESTS" Leo Breiman Statistics Department University of California Berkeley, CA 94720 January 2001
7. "Machine Learning in Financial Market Risk", Wei XIONG Supervision: Antonella Bucciaglia J.P. Morgan, Quantitative Research Market Risk September 10, 2018

Appendix

- Code for NIFTY50 returns and EVT estimation
- Code for Kurtosis and Skewness of NIFTY50 returns
- Random Forest Regression model code

Snippets of Python Codes

Code for Kurtosis and Skewness

```
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Fetching data for NSE index
nse_data = yf.download('^NSEI', start='2004-01-01', end='2024-03-01')

# Calculating daily returns
nse_data['Returns'] = nse_data['Adj Close'].pct_change() * 100
nse_returns = nse_data['Returns'].dropna()

# Plotting histogram
plt.figure(figsize=(10, 6))
plt.hist(nse_returns, bins=50, density=True, alpha=0.6, color='b')

# Fitting a normal distribution
mu, sigma = stats.norm.fit(nse_returns)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = stats.norm.pdf(x, mu, sigma)
plt.plot(x, p, 'k', linewidth=2)

plt.title('NSE Index Returns Distribution (2007-2024)')
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.grid(True)
plt.legend(['Normal Distribution', 'NSE Index Returns'])

# Measuring kurtosis and skewness
kurtosis = stats.kurtosis(nse_returns)
skewness = stats.skew(nse_returns)
annotation_text = f"Kurtosis: {kurtosis:.2f}\nSkewness: {skewness:.2f}"
plt.annotate(annotation_text, xy=(0.05, 0.75), xycoords='axes fraction', fontsize=12)

plt.show()
```

Code for NIFTY50 returns plots

```
import yfinance as yf
import matplotlib.pyplot as plt

# Define the ticker symbol for NSE index
ticker_symbol = '^NSEI'

# Define start and end dates for the data
start_date = '2007-09-17'
end_date = '2024-03-01'

# Fetch the data
```

```
nse_data = yf.download(ticker_symbol, start=start_date, end=end_date)

# Calculate daily returns
nse_data['Returns'] = nse_data['Adj Close'].pct_change()

# Drop NaN values
nse_data.dropna(inplace=True)

# Plot the returns
plt.figure(figsize=(10, 6))
plt.plot(nse_data.index, nse_data['Returns'], color='blue')
plt.title('NSE Index Returns (2007-2024)')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.grid(True)
plt.show()
```

Code for using EVT in VaR

```
import yfinance as yf
from scipy.stats import genpareto

# Define the threshold level
threshold = -0.01  # 1% loss threshold

# Define the confidence level
confidence_level = 0.90  # 90% confidence level

# Define the ticker symbol for NSE index
ticker_symbol = '^NSEI'

# Define start and end dates for the data
start_date = '2007-09-17'
end_date = '2024-03-01'

# Fetch the data
nse_data = yf.download(ticker_symbol, start=start_date, end=end_date)

# Calculate daily returns
nse_data['Returns'] = nse_data['Adj Close'].pct_change()

# Drop NaN values
nse_data.dropna(inplace=True)

# Filter data where loss is more than or equal to threshold
loss_data = nse_data[nse_data['Returns'] <= threshold]

# Calculate exceedances
exceedances = threshold - loss_data['Returns']

# Fit Generalized Pareto Distribution (GPD) to negative exceedances
params = genpareto.fit(-exceedances)

# Calculate VaR using GPD quantile
var = genpareto.ppf(1 - confidence_level, *params)

print(f"Value at Risk (VaR) at {confidence_level*100:.2f}% confidence level: {var:.4f}")
```

Code plotting the cdf of GPD

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import genpareto
```

```
# Define the threshold level
threshold = -0.01  # 1% loss threshold

# Filter data where loss is more than or equal to threshold
extreme_data = nse_data[nse_data['Returns'] <= threshold]

# Calculate exceedances
exceedances = threshold - extreme_data['Returns']

# Fit Generalized Pareto Distribution (GPD) to negative exceedances
params = genpareto.fit(-exceedances)

# Generate data points for GPD CDF
x = np.linspace(min(-exceedances), max(-exceedances), 100)
y = genpareto.cdf(x, *params)

# Plot the GPD CDF
plt.figure(figsize=(10, 6))
plt.plot(x, y, 'k', linewidth=2, label='Fitted GPD (CDF)')

# Plot settings
plt.title('Fitted Generalized Pareto Distribution (GPD) CDF to Extreme Losses (Returns <= -1%)')
plt.xlabel('Losses')
plt.ylabel('Cumulative Probability')
plt.legend()
plt.grid(True)
plt.show()
```

Code for Random Forest Regression Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import yfinance as yf
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Function to calculate VaR
def calculate_var_percent(simulated_returns, confidence_level):
    sorted_returns = np.sort(simulated_returns)
    index = int((100 - confidence_level) / 100.0 * len(sorted_returns))
    return sorted_returns[index]

def download_data(symbol, start_date, end_date):
    # Download data from Yahoo Finance
    data = yf.download(symbol, start=start_date, end=end_date)
    return data

def preprocess_data(data):
    # Calculate daily returns
    data['Daily Return'] = data['Adj Close'].pct_change()
    # Drop 'Close' column
    data.drop(columns=['Close'], inplace=True)
    # Drop NaN values
    data.dropna(inplace=True)
    return data

def split_data(data):
    # Split data into features and target variable
    y = data['Daily Return']
    X = data.drop(columns=['Daily Return'])
    return X, y

def scale_data(X_train, X_test):
    # Scale the 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

def train_model(X_train, y_train):
    # Hyperparameter tuning
    param_grid = {
        'n_estimators': [50, 100, 150],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    rf = RandomForestRegressor(random_state=42)
    rf_random = RandomizedSearchCV(estimator=rf, param_distributions=param_grid, n_iter=100, cv=3, rand
    rf_random.fit(X_train, y_train)
    return rf_random.best_estimator_

def generate_simulated_returns(model, X_test_scaled, scaling_factor=1):
    # Generate simulated returns
    simulated_returns = model.predict(X_test_scaled)
    # Scale the simulated returns
    simulated_returns_scaled = simulated_returns * scaling_factor
    return simulated_returns_scaled

def visualize_actual_vs_predicted(y_test, simulated_returns_scaled):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, simulated_returns_scaled, alpha=0.5)
    plt.plot([-0.1, 0.1], [-0.1, 0.1], '--k')
    plt.xlabel('Actual␣Returns')
    plt.ylabel('Predicted␣Returns')
    plt.title('Actual␣vs␣Predicted␣Returns')
    plt.xticks([])  # Hide x-axis tick labels
    plt.yticks([])  # Hide y-axis tick labels
    plt.grid(True)
    plt.show()

def visualize_feature_importances(model, X_train):
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(10, 6))
    sns.barplot(x=importances[indices], y=X_train.columns[indices])
    plt.title('Feature␣Importances')
    plt.xlabel('Relative␣Importance')
    plt.xticks([])  # Hide x-axis tick labels
    plt.show()

def main():
    # Download NIFTY 50 data
    nifty_data = download_data('^NSEI', start_date='2007-09-17', end_date='2024-03-01')

    # Preprocess the data
    nifty_data = preprocess_data(nifty_data)

    # Split data into features and target variable
    X, y = split_data(nifty_data)

    # Split 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)

    # Scale the features
    X_train_scaled, X_test_scaled = scale_data(X_train, X_test)

    # Train the model with hyperparameter tuning
    model = train_model(X_train_scaled, y_train)

    # Generate simulated returns
    simulated_returns_scaled = generate_simulated_returns(model, X_test_scaled)
    # Make predictions on the test set
    y_pred = model.predict(X_test_scaled)

    # Visualize actual vs predicted returns
    visualize_actual_vs_predicted(y_test, simulated_returns_scaled)

    # Visualize feature importances
    visualize_feature_importances(model, X_train)

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