Project Draft

Introduction

To safeguard agricultural productivity and maintain soil health in Telangana, our research team is conducting an analysis of irrigation water quality across various mandals within the state. Leveraging data obtained from the "Telangana Open Data Portal", we aim to identify potential correlations between water quality parameters and their implications for irrigation practices. This initiative seeks to provide actionable insights for farmers, agricultural scientists, and policymakers to optimize water resource management strategies. From previous pre-processing of the data, we found that the following features might have a bigger impact on our models.

Data Overview:

- **Mandal**: Administrative division in Telangana where the water sample was collected, serving as the primary geographical unit for our analysis.
- **Village**: Specific villages within each mandal from which samples were collected, offering granular data points for our study.
- **RSC** (Residual Sodium Carbonate): Indicates the excess of carbonate and bicarbonate ions over calcium and magnesium ions in water, crucial for evaluating the risk of soil sodicity.
- **SAR** (Sodium Adsorption Ratio): Measures the sodium hazard to crops and soil, a key factor in assessing water's suitability for irrigation.
- **Na** (Sodium content): The concentration of sodium in the water, which influences both SAR and the overall salinity hazard.
- **E.C** (Electrical Conductivity): Reflects the water's salinity level, directly impacting plant water uptake and soil structure.
- **TDS** (Total Dissolved Solids): The overall concentration of dissolved substances in water, indicating potential salinity issues.
- **HCO3** (Bicarbonate level): Concentration of bicarbonate in water, affecting soil pH and the precipitation of calcium and magnesium.

- **pH**: The acidity or alkalinity of the water, with significant implications for nutrient availability and microbial activity in the soil.
- **Classification.1:** Safety classification of the water (Safe P.S., Marginal MR, Unsafe U.S.). This variable is our response variable and has been changed to a binary response by changing MR to U.S.

The primary objective of this study is to establish a comprehensive understanding of irrigation water quality within Telangana's agricultural landscapes. By analyzing the relationships between the above water quality indicators and their collective impact on soil and plant health, we aim to categorize water sources into suitability classes for irrigation. This classification will help in formulating guidelines for water use in agriculture, thereby mitigating the risks associated with inappropriate water sources.

Methods

Random Forest Model: (Brayan Gutierrez, Tie Wang, Ulises Ramirez)

To mitigate issues such as overfitting, high variance, and poor generalization to unseen data, commonly associated with a single decision tree, we opted for an ensemble method employing decision trees. While the bagging ensemble model addresses the shortcomings of individual decision trees, it introduces a unique challenge. In the bagging model, variables tend to be highly correlated due to the consistent selection of the most important variables in each decision tree. To address these challenges and leverage the robustness of random forest models, we chose this particular ensemble method. However, by adopting the random forest model, we sacrifice some interpretability inherent in a single decision tree, as well as computational resources, and introduce the need for hyperparameter tuning, which is less significant in a single decision tree. Similar to the previous model, our response variable is "Classification.1", with the remaining variables serving as predictors. Below is the formula for our random forest model:

$$\text{Classification.1} \sim \text{Mandal} + \text{Village} + \text{RSC} + \text{SAR} + \text{Na} + \text{E.C} + \text{TDS} + \text{HCO3} + \text{pH}$$

Firstly, we aimed to evaluate the performance of the base random forest with default hyperparameters on our dataset. We read the data, removed NA observations, and converted the response variable to a binary format. Additionally, as the Mandal and Village variables contained multiple categories and to ensure proper functionality of the model, we employed one-hot encoding using the `dummyVars()` function from the *caret* library.

```
library(tree)
```

```
Warning: package 'tree' was built under R version 4.3.3
```

```
library(randomForest)
```

```
Warning: package 'randomForest' was built under R version 4.3.3
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

```
Loading required package: ggplot2


Attaching package: 'ggplot2'
The following object is masked from 'package:randomForest':

    margin
Loading required package: lattice
```

```
ground_water_quality_2022_post <- read.csv("ground_water_quality_2022_post.csv", header =

ground_water_quality_2022_post = na.omit(ground_water_quality_2022_post)

# Omitting CO3 and Season since they're the same
ground_water_quality_2022_post$CO3 = NULL
ground_water_quality_2022_post$season = NULL

# Selecting the most useful features
gw_df = ground_water_quality_2022_post[,c(23, 21, 16, 9, 10, 11, 8, 3, 4)]

# One-hot-encoding Mandal and Village
dummy = dummyVars(" ~ .", data = gw_df)
dum_df = data.frame(predict(dummy, newdata = gw_df))
```

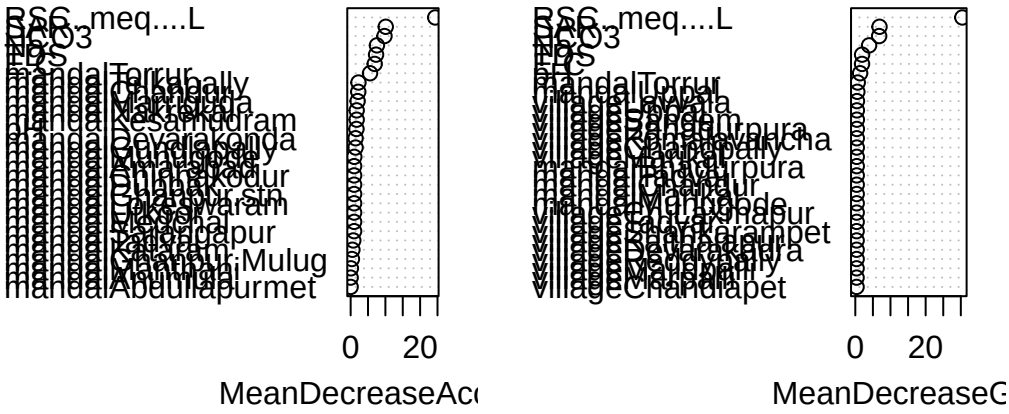
```r
# Changing Marginal Safe (MR) to Unsafe (U.S.)
Classification.1 = ground_water_quality_2022_post[,24]
final_gw_df = data.frame(dum_df, Classification.1)
final_gw_df$Classification.1 = gsub("MR", "U.S.", final_gw_df$Classification.1)
final_gw_df$Classification.1 = as.factor(final_gw_df$Classification.1)
```

To obtain the testing error for a classification random forest model, we would need to generate a confusion matrix for each test set. As we are testing the model 10 times with different 80% training and 20% testing splits, we decided to store all the confusion matrices for these splits in a list called "conf_mat". Additionally, we calculated and stored each model's out-of-bag score in a list named "OOB" before the loop. The model was trained on the 10 different training sets and then tested on 10 different testing sets within a for loop. We used a seed of 1 for reproducibility. Since the `cv.tree()` function only works for regression random forest models, we didn't use it here. Furthermore, we plotted the variable importance for each training/testing split. Below is the code for this process:

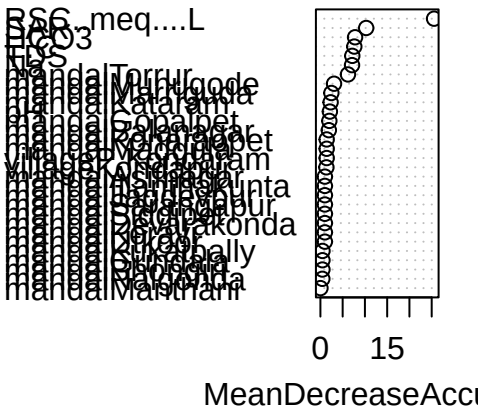
```r
conf_mat = list()
OOB = rep(0,10)

set.seed(1)

for (i in 1:10) {

  # Splitting data
  sample = sample(1:nrow(final_gw_df), 0.8 * nrow(final_gw_df))
  training = final_gw_df[sample,]
  testing = final_gw_df[-sample,]

  # Random Forest Model
  rf_gw = randomForest(Classification.1 ~., data = training, proximity = T, importance = T

  # OOB of Models
  conf = rf_gw$confusion[,-ncol(rf_gw$confusion)]
  OOB[i] = 1 - (sum(diag(conf)) / sum(conf))

  # Importance Plots
  importance(rf_gw)
  varImpPlot(rf_gw, main = paste('Ground Water Random Forest Split', i, sep = ' '))

  test.pred = predict(rf_gw, newdata = testing)
```

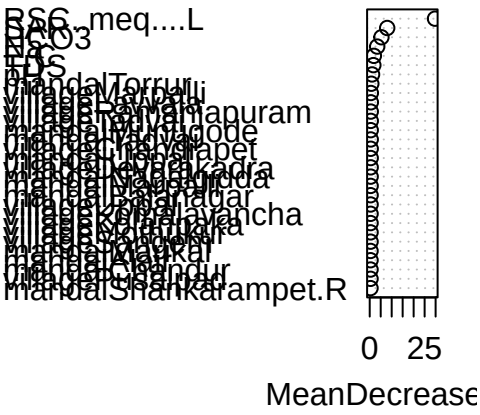
```
  # Confusion Matrix
  conf_mat[[i]] = table(testing$Classification.1, test.pred)

}
```

Ground Water Random Forest Split 1

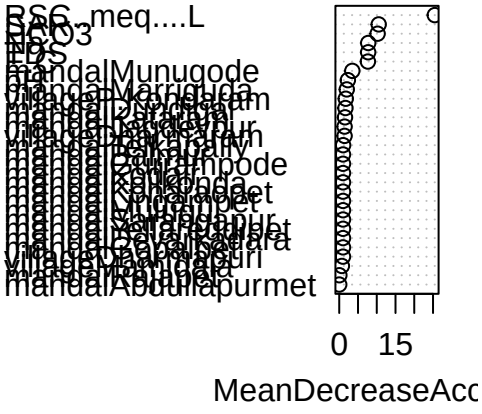


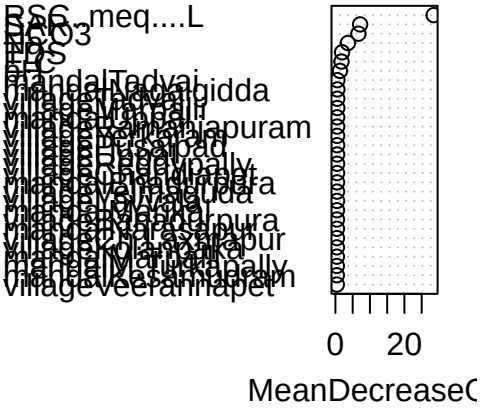
Ground Water Random Forest Split 2



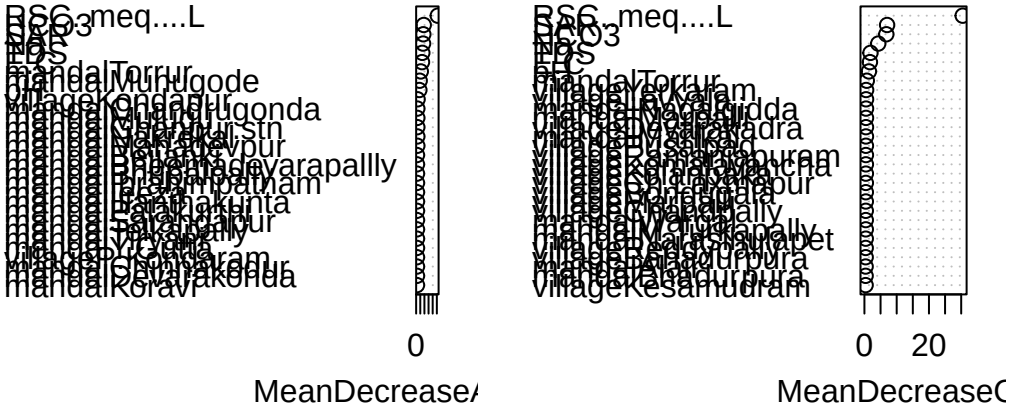


Ground Water Random Forest Split 3

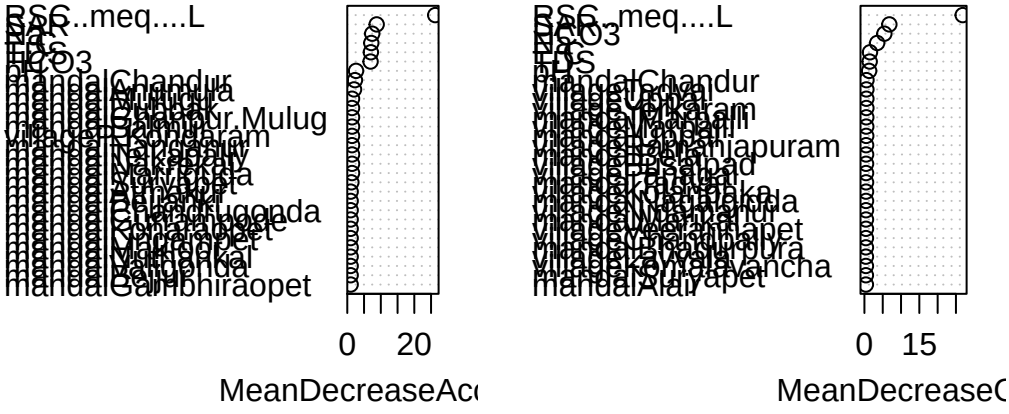




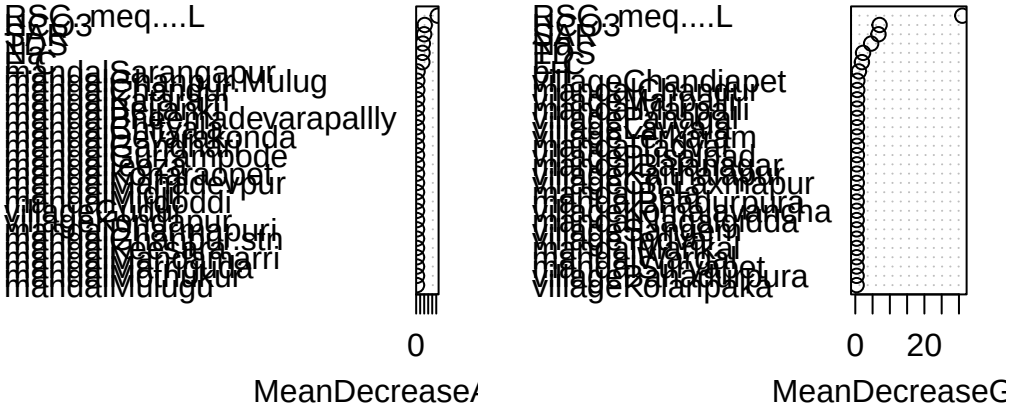
Ground Water Random Forest Split 4



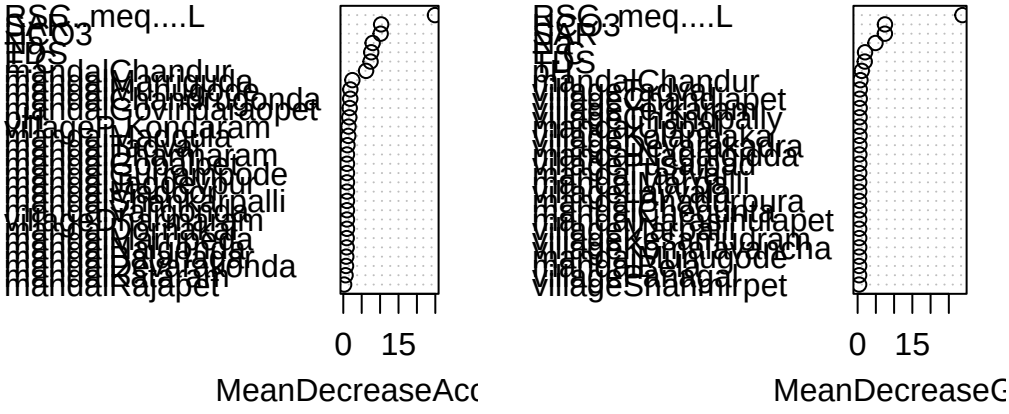
Ground Water Random Forest Split 5



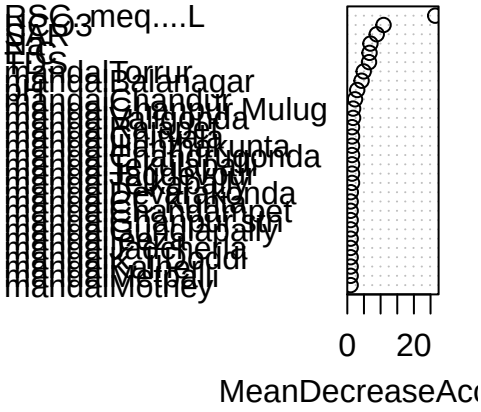
Ground Water Random Forest Split 6

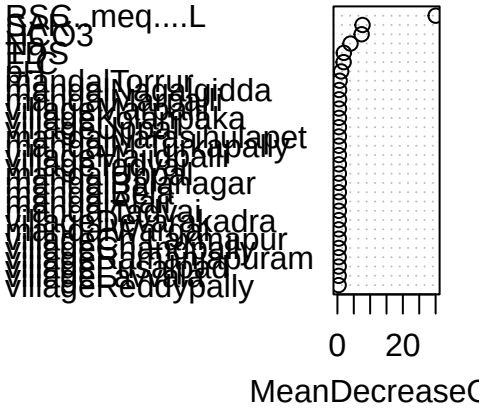


Ground Water Random Forest Split 7

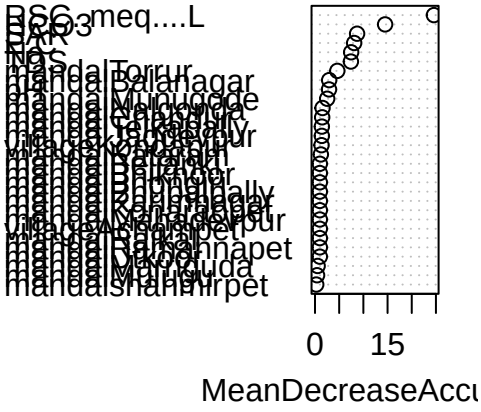


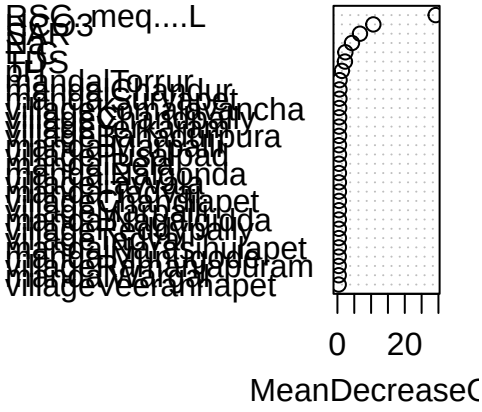
Ground Water Random Forest Split 8

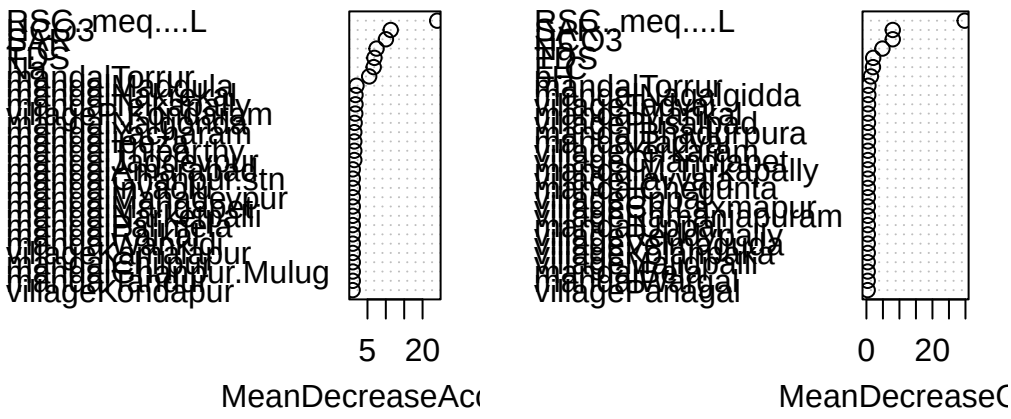




Ground Water Random Forest Split 9







We observed that the three most important variables in each split were RSC, SAR, and HCO3, with SAR and HCO3 competing for the second most important variable across the different splits. Now that we have saved the confusion matrices for each training/testing split, we can utilize them to calculate the testing error for each split.

```
# Test Error Calculations
test_errors = rep(0, 10)
accuracies = rep(0, 10)

for (i in 1:10) {
    TP = conf_mat[[i]][2, 2]  # True Positives
    FP = conf_mat[[i]][1, 2]  # False Positives
    FN = conf_mat[[i]][2, 1]  # False Negatives
    TN = conf_mat[[i]][1, 1]  # True Negatives

    # Calculating test error rate
    test_errors[i] = (FP + FN) / sum(conf_mat[[i]])

    # Calculating Accuracies
    accuracies[i] = (TP + TN) / sum(conf_mat[[i]])
}
```

```r
scores = list(test_error = test_errors, OOB_score = OOB, accuracy = accuracies)

scores
```

```
$test_error
 [1] 0.024390244 0.006097561 0.006097561 0.006097561 0.006097561 0.000000000
 [7] 0.012195122 0.000000000 0.012195122 0.012195122

$OOB_score
 [1] 0.015313936 0.006125574 0.015313936 0.007656968 0.009188361 0.007656968
 [7] 0.015313936 0.006125574 0.016845329 0.021439510

$accuracy
 [1] 0.9756098 0.9939024 0.9939024 0.9939024 0.9939024 1.0000000 0.9878049
 [8] 1.0000000 0.9878049 0.9878049
```

```r
(avg_test_error = sum(test_errors) / 10)
```

```
[1] 0.008536585
```

We discovered that the base random forest model, evaluated across 10 different training/testing splits, exhibited test errors ranging from 0% to 2.4%, with an average testing error of 0.8%. Comparing these testing errors with the out-of-bag (OOB) scores, we observed minimal disparity, indicating that the base random forest model performs admirably with unseen data, affirming its ability to generalize well beyond the training set. This suggests a robust learning of underlying patterns within the data. However, it's worth noting the limitation posed by the fewer U.S. observations in our dataset, which might affect the model's performance on U.S. data. With this promising result, we proceeded to fine-tune the hyperparameters of the random forest model on our dataset.

Model Optimization Results:

The model optimization process involved an exhaustive search for the optimal combination of hyperparameters to maximize prediction accuracy while preventing overfitting. This was achieved through grid search, where various values of hyperparameters such as the number of estimators (trees), maximum depth, minimum samples split, minimum samples leaf, and maximum features were explored. Each hyperparameter combination was evaluated using cross-validation, which split the data into training and validation sets to assess model performance.

The optimization process was repeated across 10 different 80/20 splits of the dataset to ensure robustness and generalizability of the results. This approach allowed for a comprehensive

exploration of the hyperparameter space and provided insight into the sensitivity of the model to different parameter settings. By leveraging multiple splits, the final model selection was based on the mean cross-validation score across all splits, ensuring that the chosen model performed well on diverse subsets of the data.

Impact of Hyperparameters on Model Performance:

The hyperparameters explored during grid search play a crucial role in determining the performance of the Random Forest model. The number of estimators (trees) determines the complexity of the model and its ability to capture intricate relationships within the data. Increasing the number of trees can improve predictive accuracy up to a certain point, beyond which additional trees may lead to diminishing returns or increased computational costs.

The maximum depth of the trees controls the depth of individual decision trees in the Random Forest ensemble. Deeper trees can capture complex interactions but may also be prone to overfitting, especially with small datasets. By limiting the maximum depth, the model can generalize better to unseen data while still capturing important patterns in the training data.

Similarly, the minimum samples split and minimum samples leaf parameters regulate the minimum number of samples required to split an internal node and the minimum number of samples required to be a leaf node, respectively. These parameters help control the granularity of the decision tree structure and prevent overfitting by imposing constraints on tree growth.

The choice of maximum features determines the number of features considered for each split in the decision trees. By randomly selecting a subset of features, Random Forest mitigates the risk of feature dominance and enhances model diversity, leading to improved generalization performance.

```
# Define parameter grid for grid search
param_grid <- expand.grid(
  n_estimators = c(50, 100, 200),
  max_depth = c(5, 10, 15),
  min_samples_split = c(2, 5, 10),
  min_samples_leaf = c(1, 3, 5),
  max_features = c("sqrt", "log2"),  # Ensure max_features is specified as characters
  stringsAsFactors = FALSE  # Ensure strings are not converted to factors
)

# Initialize empty list to store results
results <- list()
```

Robustness and Generalizability of the Model:

By performing grid search and cross-validation across multiple data splits, the optimized Random Forest model demonstrates robustness and generalizability. The model's ability to consistently perform well across diverse subsets of the data indicates its stability and reliability.

This ensures that the selected hyperparameters are not overly tailored to a specific subset of the data but rather capture underlying patterns that generalize to unseen instances.

```
# Perform 10 different 80/20 splits and grid search on each split
for (split in 1:10) {
  # Set seed for reproducibility
  set.seed(1)

  # Split data into training and testing sets (80/20 split)
  sample <- sample(1:nrow(final_gw_df), 0.8 * nrow(final_gw_df))
  training <- final_gw_df[sample, ]
  testing <- final_gw_df[-sample, ]

  # Initialize cross-validation folds for grid search
  folds <- createFolds(training$Classification.1, k = 5, list = TRUE, returnTrain = FALSE)

  # Perform grid search
  for (i in 1:nrow(param_grid)) {
    params <- param_grid[i, ]

    # Initialize vector to store cross-validation scores
    cv_scores <- c()

    # Perform cross-validation
    for (fold in folds) {
      # Split training data into training and validation sets
      train_data <- training[-fold, ]
      val_data <- training[fold, ]

      # Determine the value of mtry based on max_features
      if (params$max_features == "sqrt") {
        mtry_value <- sqrt(ncol(train_data) - 1)  # Subtract 1 for the target variable
      } else if (params$max_features == "log2") {
        mtry_value <- log2(ncol(train_data) - 1)
      } else {
        mtry_value <- params$max_features
      }

      # Fit random forest model
      rf_gw <- randomForest(Classification.1 ~ ., data = train_data,
                            ntree = params$n_estimators,
                            mtry = round(mtry_value),  # Round to nearest integer
```

```
                            maxdepth = params$max_depth,
                            nodesize = params$min_samples_leaf)

      # Make predictions on validation set
      val_pred <- predict(rf_gw, newdata = val_data)

      # Calculate accuracy
      accuracy <- sum(val_pred == val_data$Classification.1) / length(val_data$Classificat

      # Append accuracy to vector
      cv_scores <- c(cv_scores, accuracy)
    }

    # Store mean cross-validation score in results list
    results[[paste("Split", split, "Model", i, sep = "_")]] <- mean(cv_scores)
  }
}


print(results)
results_df <- data.frame(Model = names(results), Score = unlist(results))
# Find the row with the highest score
best_model <- results_df[which.max(results_df$Score), ]
print(best_model)
best_model_name <- best_model$Model
# Find the corresponding parameters in the param_grid
best_model_params <- param_grid[grep(best_model_name, rownames(param_grid)), ]
print(best_model_params)


best_model_name <- best_model$Model
best_model_hyperparams <- unlist(strsplit(best_model_name, "_"))
# Extract the values of the best hyperparameters
best_model_values <- best_model_hyperparams[grep("^[0-9]+$", best_model_hyperparams)]

# Check if any values were found
if (length(best_model_values) > 0) {
  print(best_model_values)
} else {
  print("No values found for the best hyperparameters.")
}
```

```r
# Extract split number and model number
split_number <- as.integer(best_model_values[1])
model_number <- as.integer(best_model_values[2])
# Get the corresponding row in param_grid
best_model_params <- param_grid[model_number, ]
print(best_model_params)
```

The results of the model evaluation indicate that the model labeled as "Split_2_Model_46" achieved a mean cross-validation score of approximately 0.9954 for Split 2 of the data. This score suggests that the model performed exceptionally well on the validation data for this particular split. Further analysis reveals that this model was trained with a specific combination of hyperparameters, which can be found below the Score.

In conclusion, the machine learning model developed using Random Forests and optimized through cross-validation demonstrates high predictive accuracy for the task of groundwater quality prediction. The results of the evaluation process provide valuable insights into the performance of the model and offer guidance for future model development and refinement.

Future Directions:

While the optimized Random Forest model shows promising results in predicting ground water quality, there are opportunities for further refinement and enhancement. Future research may explore advanced techniques for hyperparameter tuning, such as Bayesian optimization or genetic algorithms, to more efficiently navigate the hyperparameter space. Additionally, incorporating domain knowledge and additional features, such as meteorological data or land use information, could further improve the model's predictive capabilities and provide valuable insights for environmental monitoring and management. Finally, ongoing validation and evaluation of the model on independent datasets are essential to ensure its applicability and reliability in real-world scenarios.

Results

Random Forest:

In general, we observed that the random forest model, whether base or tuned, performed well on the testing data. This indicates that our models are robust and generalizable. Starting with the base model, among the 10 different splits, we observed testing errors between 0% to 2.4% and accuracies between 97.6% to 100%. This suggests that our variables positively contributed to our base model's classification of the test observations, with RSC, SAR, and HCO3 being the most important across all models. It is safe to assume that these top variables, along

with the other important variables, will be present during the tuning process of the random forest model. This addresses our objective of identifying correlations between water quality parameters and their implications for soil health. By tuning the model, we were able to find the optimal hyperparameters that minimized the model's test error and maximized accuracy on unseen data. We observed that the parameters that led to the best results were 46 estimators, a max depth of 50, a minimum sample split of 5, a minimum leaf sample of 3, and a maximum number of features of $\sqrt{p}$ where p is the total number of features. With these hyperparameters, and the previously identified water quality parameters, we ensure that our model would have a classification accuracy of approximately 99%.

Conclusion