Project 2_Data 624

Heleine Fouda

2024-12-10

Contents

As a team of data scientists at ABC Beverage, we are responding to new regulatory requirements by thoroughly analyzing our manufacturing process to identify key predictive factors influencing pH levels. Using the historical dataset provided to us, we aim to build a reliable predictive model that meets compliance standards while supporting operational excellence. Our approach will involve: 1. Conducting exploratory analysis to identify and understand the factors most strongly associated with pH variability, such as temperature, pressure,and material composition. 2. Developing and evaluating predictive models using statistical and machine learning methods, selecting the most accurate and interpretable approach for forecasting pH levels. To meet the needs of our diverse audience, we will provide both a Non-Technical Report (Section II) and a Technical Report(Section I).The Non-Technical Report will summarize our findings and recommendations in a clear and business - friendly language to our leadership. The Technical Report will be a more detailed document outlining our methodology, including the models tested, the performance metrics, and our rationale for selecting the final model(s). To ensure reproducibility, we converted the provided Excel files to CSV format and uploaded them to a publicly accessible GitHub repository. The predictions from our models will be delivered in an Excel-compatible format and as a pdf for easy review and integration. Both reports, along with RMarkdown (.rmd) files and published Rpubs links, will be submitted to ensure transparency and accessibility. By combining rigorous analysis with clear reporting, we aim to empower leadership at ABC Beverage to make informed decisions in compliance with the new regulations.

Set up the environment: Load all necessary libraries and set the seed for reproducibility.

Create and register a parallel processing cluster

```
library(parallel)
library(doParallel)

# Detect the number of cores
no_cores <- detectCores() - 1

# Create the cluster
cl <- makeCluster(no_cores)

# Register the cluster
registerDoParallel(cl)

# Properly stop the cluster at the end
stopCluster(cl)
```

Data loading and preparation:

```
test <- read.csv("https://raw.githubusercontent.com/MarjeteV/data624/refs/heads/main/test%20data.csv")
train <- read.csv("https://raw.githubusercontent.com/MarjeteV/data624/refs/heads/main/training%20data.cs
```

```
tidy_train <- train |>
  as_tibble()
tidy_train |>
  summarise(across(everything(), ~ sum(is.na(.)))) |>
  pivot_longer(cols = everything(), names_to = "Column", values_to = "Missing Entries") |>
  arrange(desc(`Missing Entries`))
```

```
## # A tibble: 33 x 2
##    Column         `Missing Entries`
##    <chr>                      <int>
##  1 MFR                          212
##  2 Filler.Speed                  57
##  3 PC.Volume                     39
##  4 PSC.CO2                       39
##  5 Fill.Ounces                   38
##  6 PSC                           33
##  7 Carb.Pressure1                32
##  8 Hyd.Pressure4                 30
##  9 Carb.Pressure                 27
## 10 Carb.Temp                     26
## # i 23 more rows
```

```
tidy_train |>
  summarise(n = n())
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  2571
```

```
tidy_train |>
  count(Brand.Code)
```

```
## # A tibble: 5 x 2
##   Brand.Code     n
##   <chr>      <int>
## 1 ""           120
## 2 "A"          293
## 3 "B"         1239
## 4 "C"          304
## 5 "D"          615
```

```
library(dplyr)

tidy_train |>
  nearZeroVar(saveMetrics = TRUE) |>
  dplyr::filter(zeroVar == TRUE | nzv == TRUE)
```

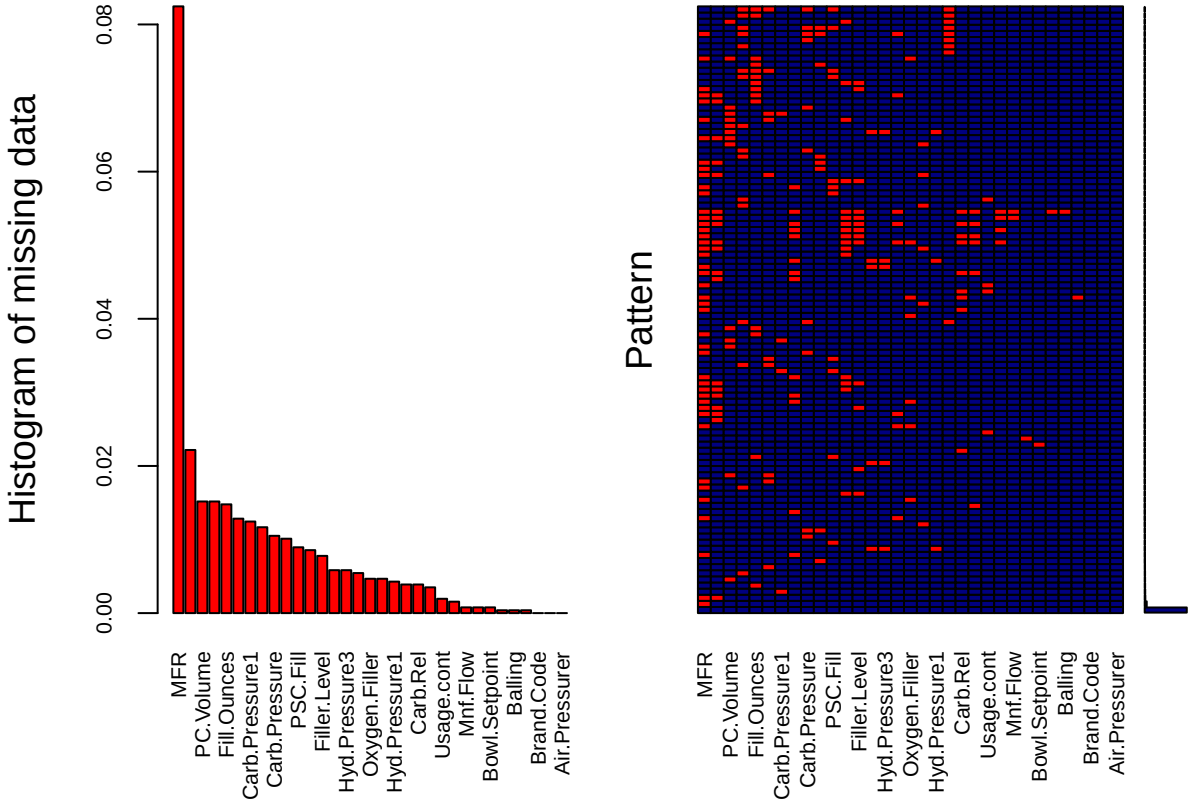
```
##               freqRatio percentUnique zeroVar  nzv
## Hyd.Pressure1  31.11111      9.529366   FALSE TRUE
```

```
sapply(tidy_train, class)
```

```
##         Brand.Code        Carb.Volume        Fill.Ounces          PC.Volume
##        "character"          "numeric"          "numeric"          "numeric"
##      Carb.Pressure          Carb.Temp                PSC           PSC.Fill
##          "numeric"          "numeric"          "numeric"          "numeric"
##            PSC.CO2           Mnf.Flow     Carb.Pressure1     Fill.Pressure
##          "numeric"          "numeric"          "numeric"          "numeric"
##      Hyd.Pressure1      Hyd.Pressure2      Hyd.Pressure3      Hyd.Pressure4
##          "numeric"          "numeric"          "numeric"          "integer"
##       Filler.Level       Filler.Speed        Temperature         Usage.cont
##          "numeric"          "integer"          "numeric"          "numeric"
##          Carb.Flow            Density                MFR            Balling
##          "integer"          "numeric"          "numeric"          "numeric"
##    Pressure.Vacuum                 PH      Oxygen.Filler      Bowl.Setpoint
##          "numeric"          "numeric"          "numeric"          "integer"
##  Pressure.Setpoint      Air.Pressurer           Alch.Rel           Carb.Rel
##          "numeric"          "numeric"          "numeric"          "numeric"
##        Balling.Lvl
##          "numeric"
```

```
aggr_plot <- aggr(tidy_train, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE, labels=names(data),
```

```
## Warning in plot.aggr(res, ...): not enough vertical space to display
## frequencies (too many combinations)
```




```
##
##  Variables sorted by number of missings:
##            Variable        Count
```

```
##                MFR 0.0824581875
##       Filler.Speed 0.0221703617
##          PC.Volume 0.0151691949
##            PSC.CO2 0.0151691949
##        Fill.Ounces 0.0147802412
##                PSC 0.0128354726
##     Carb.Pressure1 0.0124465189
##      Hyd.Pressure4 0.0116686114
##      Carb.Pressure 0.0105017503
##          Carb.Temp 0.0101127966
##           PSC.Fill 0.0089459354
##      Fill.Pressure 0.0085569817
##       Filler.Level 0.0077790743
##      Hyd.Pressure2 0.0058343057
##      Hyd.Pressure3 0.0058343057
##        Temperature 0.0054453520
##      Oxygen.Filler 0.0046674446
##  Pressure.Setpoint 0.0046674446
##      Hyd.Pressure1 0.0042784909
##        Carb.Volume 0.0038895371
##           Carb.Rel 0.0038895371
##           Alch.Rel 0.0035005834
##         Usage.cont 0.0019447686
##                 PH 0.0015558149
##           Mnf.Flow 0.0007779074
##          Carb.Flow 0.0007779074
##      Bowl.Setpoint 0.0007779074
##            Density 0.0003889537
##            Balling 0.0003889537
##        Balling.Lvl 0.0003889537
##         Brand.Code 0.0000000000
##    Pressure.Vacuum 0.0000000000
##      Air.Pressurer 0.0000000000
```

```r
library(dplyr)
library(caret)
# Convert integer columns to numeric to prevent errors
tidy_train <- tidy_train |>
  mutate(across(
    .cols = c('Filler.Speed', 'Hyd.Pressure4', 'Bowl.Setpoint', 'Carb.Flow'),
    .fns = as.numeric
  ))

# Drop rows with missing Brand.Code or PH
drop_missing_cat <- tidy_train |>
  dplyr::filter(Brand.Code != "" & !is.na(PH))

# Create a one-hot encoding tibble
dummies <- dummyVars(~ Brand.Code, data = drop_missing_cat)
one_hot_df <- predict(dummies, newdata = drop_missing_cat) %>%
  as_tibble()

# Add the one-hot encoded data to the original and remove the categorical column
one_hot_df <- drop_missing_cat |>
```

```
  cbind(one_hot_df) |>
  dplyr::select(-Brand.Code)

# Impute data
preprocessor <- preProcess(one_hot_df, method = "bagImpute")
imputed_data <- predict(preprocessor, newdata = one_hot_df)
```

```
# Verify no columns have missing data
imputed_data |>
  summarise(across(everything(), ~ sum(is.na(.)))) |>
  pivot_longer(cols = everything(), names_to = "Column", values_to = "Missing Entries") |>
  arrange(desc(`Missing Entries`))
```

```
## # A tibble: 36 x 2
##    Column          `Missing Entries`
##    <chr>                       <int>
##  1 Carb.Volume                     0
##  2 Fill.Ounces                     0
##  3 PC.Volume                       0
##  4 Carb.Pressure                   0
##  5 Carb.Temp                       0
##  6 PSC                             0
##  7 PSC.Fill                        0
##  8 PSC.CO2                         0
##  9 Mnf.Flow                        0
## 10 Carb.Pressure1                  0
## # i 26 more rows
```

```
# Total number of rows after dropping those outlined earlier
imputed_data |>
  summarise(n = n())
```

```
##      n
## 1 2447
```

```
imputed_data |>
  write_csv("imputed_test_data.csv")
```

Random Forest Model

```
# Setting the seed for reproducibility
set.seed(8675309)
# Load the imputed training data
imputed_data <- read_csv("imputed_test_data.csv")
```

```
# Split the data into training (75%) and testing (25%) sets, using PH as the target
data_split <- createDataPartition(imputed_data$PH, p = 0.75, list = FALSE)
training_data <- imputed_data[data_split, ]
testing_data <- imputed_data[-data_split, ]
```

```
# Separate predictors and response variable for both training and testing sets
train_x <- training_data %>% select(-PH)
train_y <- training_data$PH
test_x <- testing_data %>% select(-PH)
test_y <- testing_data$PH
```

```r
# Fit a baseline Random Forest model
rf_model_default <- randomForest(
  x = train_x,
  y = train_y,
  ntree = 1000,
  importance = TRUE
)
pred_default <- predict(rf_model_default, test_x)
rmse_default <- sqrt(mean((test_y - pred_default)^2))
cat("Default RMSE:", rmse_default, "\n")
```

Fit a baseline Random Forest model

```
## Default RMSE: 0.09502031
```

```r
rf_model_default
```

```
##
## Call:
##  randomForest(x = train_x, y = train_y, ntree = 1000, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 11
##
##           Mean of squared residuals: 0.009558166
##                     % Var explained: 67.98
```

```r
# Tune mtry
mtry_values <- seq(2, ncol(train_x), by = 2)
mtry_results <- sapply(mtry_values, function(m) {
  rf_model <- randomForest(x = train_x, y = train_y, ntree = 1000, mtry = m, importance = TRUE)
  pred <- predict(rf_model, test_x)
  sqrt(mean((test_y - pred)^2))
})
best_mtry <- mtry_values[which.min(mtry_results)]
cat("Optimal mtry:", best_mtry, "\n")
```

Tune mtry

```
## Optimal mtry: 20
```

```r
# Tune nodesize
nodesize_values <- c(1, 5, 10, 20)
nodesize_results <- sapply(nodesize_values, function(n) {
  rf_model <- randomForest(x = train_x, y = train_y, ntree = 1000, mtry = best_mtry, nodesize = n, impo
  pred <- predict(rf_model, test_x)
  sqrt(mean((test_y - pred)^2))
})
best_nodesize <- nodesize_values[which.min(nodesize_results)]
cat("Optimal nodesize:", best_nodesize, "\n")
```

Tune nodesize

```
## Optimal nodesize: 1
```

```r
# Tune maxnodes
maxnodes_values <- seq(10, 50, by = 10)
maxnodes_results <- sapply(maxnodes_values, function(mn) {
  rf_model <- randomForest(x = train_x, y = train_y, ntree = 1000, mtry = best_mtry, nodesize = best_noc
  pred <- predict(rf_model, test_x)
  sqrt(mean((test_y - pred)^2))
})
best_maxnodes <- maxnodes_values[which.min(maxnodes_results)]
cat("Optimal maxnodes:", best_maxnodes, "\n")
```

Tune maxnodes

```
## Optimal maxnodes: 50
```

```r
# Fit the final optimized Random Forest model
rf_model_optimized <- randomForest(
  x = train_x,
  y = train_y,
  ntree = 1000,
  mtry = best_mtry,
  nodesize = best_nodesize,
  maxnodes = best_maxnodes,
  importance = TRUE
)
pred_optimized <- predict(rf_model_optimized, test_x)
rmse_optimized <- sqrt(mean((test_y - pred_optimized)^2))
cat("Optimized RMSE:", rmse_optimized, "\n")
```

Fit the final optimized Random Forest model

```
## Optimized RMSE: 0.1094477
```

```r
rf_model_optimized
```

```
##
## Call:
##  randomForest(x = train_x, y = train_y, ntree = 1000, mtry = best_mtry,      nodesize = best_nodesiz
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 20
##
##           Mean of squared residuals: 0.01243471
##                     % Var explained: 58.34
```

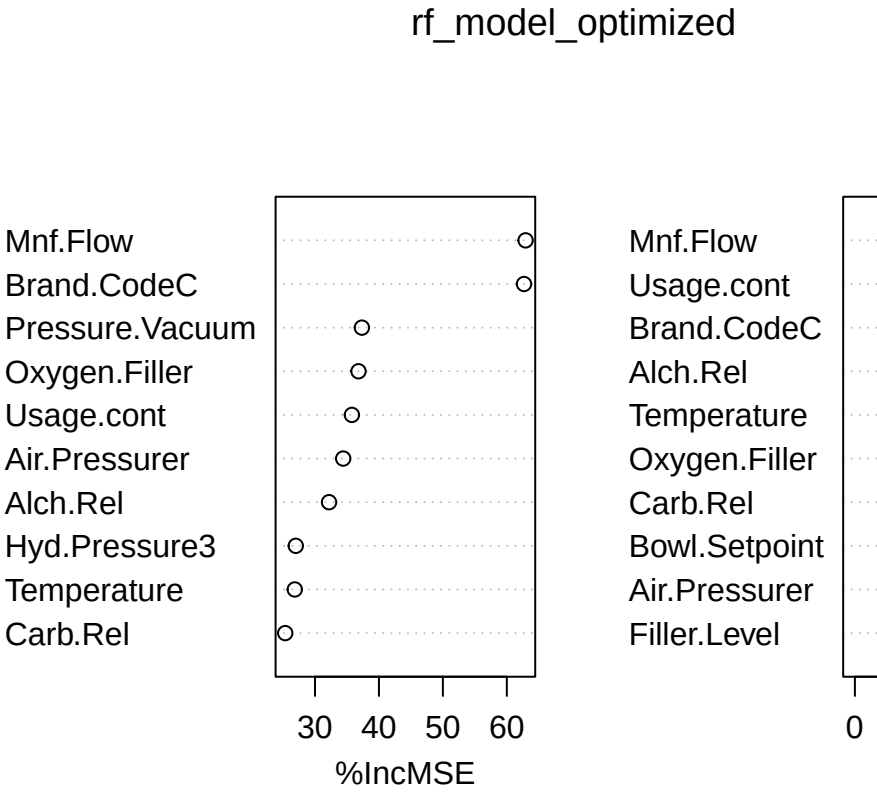
```r
# Compare default and optimized results
cat("Default RMSE:", rmse_default, "\n")
```

```
## Default RMSE: 0.09502031
```

```r
cat("Optimized RMSE:", rmse_optimized, "\n")
```

```
## Optimized RMSE: 0.1094477
```

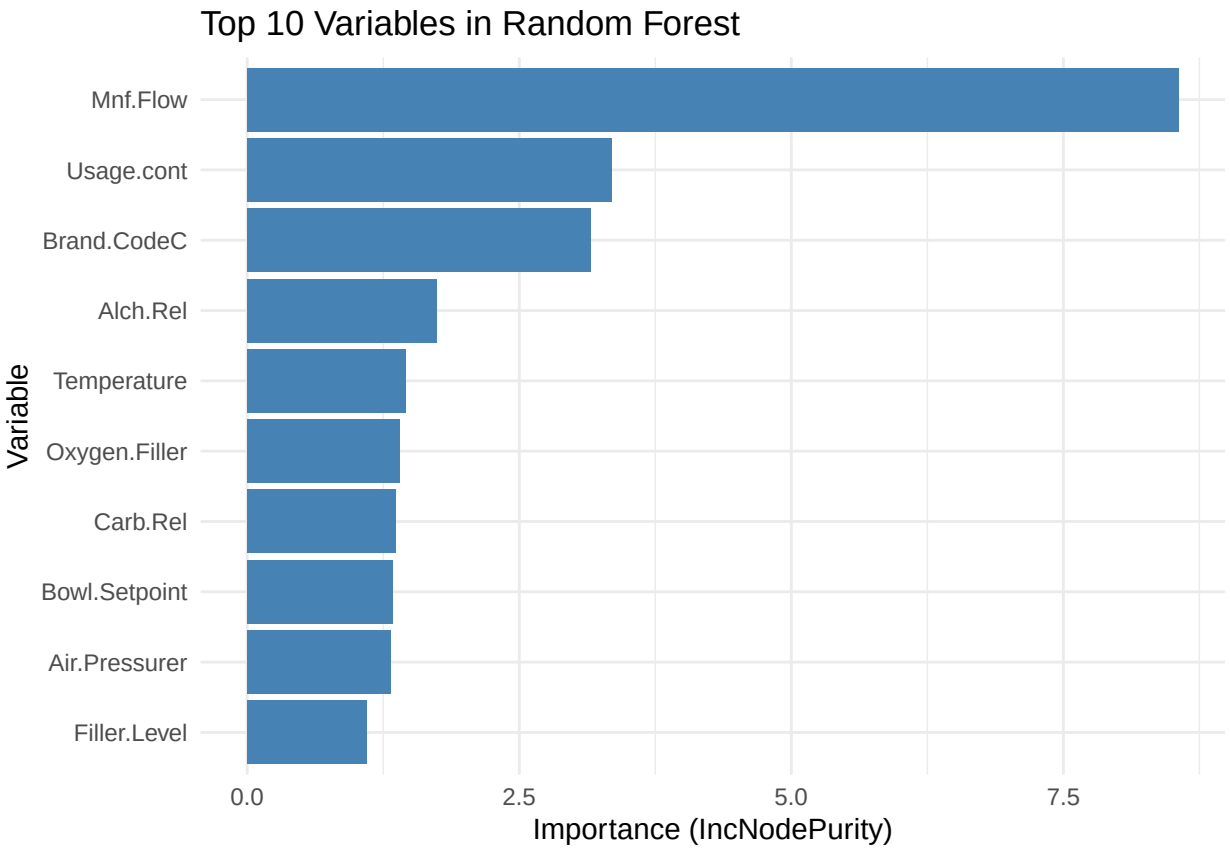
```r
# Variable importance from the final model
varImpPlot(rf_model_optimized, sort = TRUE, n.var = 10)
```



Variable importance from the final model

```r
# Visualize variable importance
importance_data <- as.data.frame(rf_model_optimized$importance)
importance_data <- importance_data %>%
  rownames_to_column(var = "Variable") %>%
  arrange(desc(IncNodePurity)) %>%
  slice_head(n = 10)

ggplot(importance_data, aes(x = reorder(Variable, IncNodePurity), y = IncNodePurity)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top 10 Variables in Random Forest", x = "Variable", y = "Importance (IncNodePurity)") +
  theme_minimal()
```




```
# Read the test data
test <- read.csv("https://raw.githubusercontent.com/MarjeteV/data624/refs/heads/main/test%20data.csv")

# Apply manual one-hot encoding to the Brand.Code variable
eval_data <- test %>%
  mutate(
    `Brand.Code.B` = ifelse(Brand.Code == "B", 1, 0),
    `Brand.Code.C` = ifelse(Brand.Code == "C", 1, 0),
    `Brand.Code.D` = ifelse(Brand.Code == "D", 1, 0)
  ) %>%
  select(-Brand.Code)  # Drop the original Brand.Code column

# Write the result to evaluation_data.csv
write.csv(eval_data, "evaluation_data.csv", row.names = FALSE)
eval_data <- read_csv("evaluation_data.csv")
```

```
# Predictions and performance on the evaluation dataset
eval_data <- read_csv("evaluation_data.csv") # Load evaluation data

# Ensure data types match between train_x and eval_data
train_x[] <- lapply(train_x, function(x) if(is.integer(x)) as.double(x) else x)
eval_data[] <- lapply(eval_data, function(x) if(is.integer(x)) as.double(x) else x)

# Recipe for imputation
recipe_obj <- recipe(~ ., data = train_x) %>%
```

```
  step_impute_bag(all_predictors())
recipe_prepped <- prep(recipe_obj)

# Align eval_data with train_x
missing_cols <- setdiff(names(train_x), names(eval_data))
for (col in missing_cols) eval_data[[col]] <- NA
eval_data_imputed <- bake(recipe_prepped, new_data = eval_data)
```

```
# Make predictions on evaluation dataset
rf_eval_predictions <- predict(rf_model_optimized, newdata = eval_data_imputed)
cat("Predictions on Evaluation Dataset:", rf_eval_predictions, "\n")
```

Predictions and performance on the evaluation dataset

```
## Predictions on Evaluation Dataset: 8.616627 8.515918 8.520494 8.581553 8.528347 8.563085 8.538249 8.5
```

```
# Convert predictions to a data frame
rf_predictions_df <- data.frame("Random Forest - Predicted PH" = rf_eval_predictions)

# Write the data frame to an Excel file
write_xlsx(rf_predictions_df, path = "Random_Forest_Predicted_PH.xlsx")

cat("Predictions saved to 'Random_Forest_Predicted_PH Values.xlsx'.\n")
```

Convert predicted PH values to an excel format

```
## Predictions saved to 'Random_Forest_Predicted_PH Values.xlsx'.
```

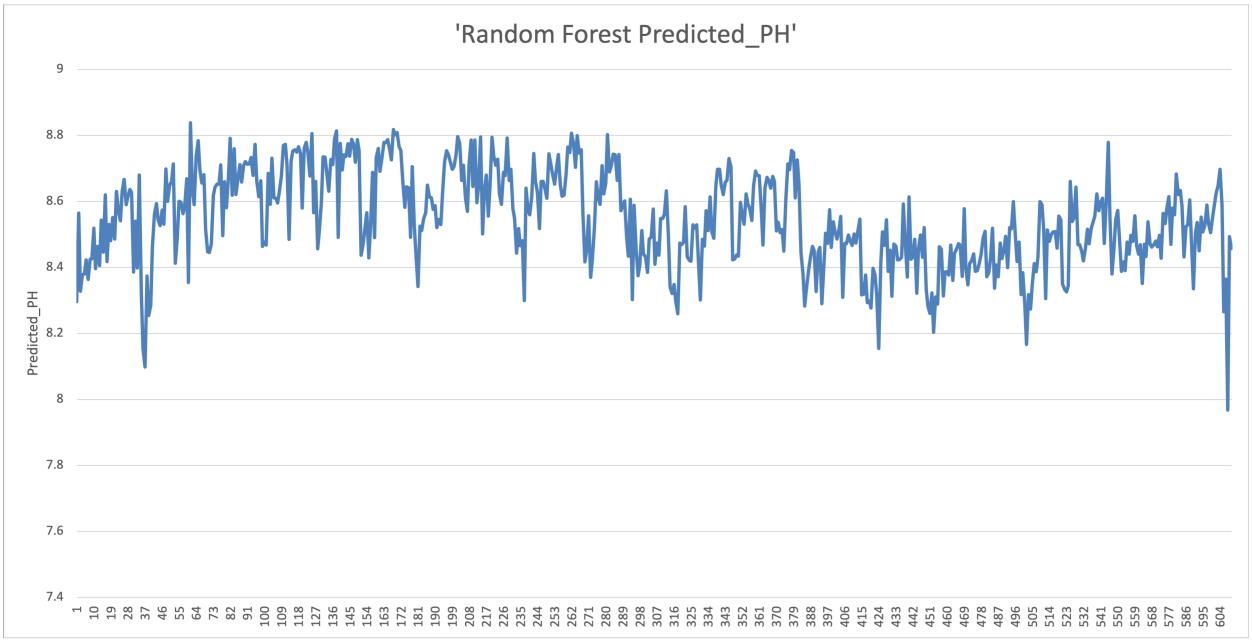


Figure 1: Random Forest - Predicted PH

XGBoost Model

```
# Prepare data for XGBoost
xgb_train <- xgb.DMatrix(data = as.matrix(training_data[,-which(names(training_data) == "PH")]),
                         label = training_data$PH)
xgb_test <- xgb.DMatrix(data = as.matrix(testing_data[,-which(names(testing_data) == "PH")]),
                        label = testing_data$PH)
```

```
# Fit XGBoost model
xgb_model <- xgboost(data = xgb_train, nrounds = 500, objective = "reg:squarederror", verbose = 0)
xgb_model
```

```
## ##### xgb.Booster
## raw: 905.6 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, objective = "reg:squarederror")
## params (as set within xgb.train):
##   objective = "reg:squarederror", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 35
## niter: 500
## nfeatures : 35
## evaluation_log:
##    iter   train_rmse
##   <num>        <num>
##       1 5.6373569433
##       2 3.9489972616
##     ---          ---
##     499 0.0009519344
##     500 0.0009519344
```

```
# Make predictions
xgb_predictions <- predict(xgb_model, xgb_test)
xgb_predictions
```

```
##   [1] 8.592402 8.534020 8.457472 8.577076 8.343072 8.468810 8.346246 8.457032
##   [9] 8.324239 8.470515 8.377457 8.539107 8.357828 8.758938 8.500212 8.475198
##  [17] 8.468724 8.405086 8.573372 8.571882 8.541825 8.568162 8.453789 8.612017
##  [25] 8.505932 8.608367 8.413998 8.621545 8.410470 8.537630 8.445471 8.518950
##  [33] 8.524765 8.660248 8.586647 8.738572 8.494659 8.422452 8.702604 8.556795
##  [41] 8.177881 8.293600 8.200117 8.284523 8.628872 8.347286 8.522160 8.440908
##  [49] 8.580902 8.767773 8.588437 8.608055 8.610805 8.540968 8.589284 8.648047
##  [57] 8.454151 8.445867 8.627739 8.688824 8.680476 8.646350 8.686100 8.724051
##  [65] 8.558842 8.652841 8.614025 8.728535 8.356428 8.529671 8.374063 8.397600
##  [73] 8.700023 8.596678 8.693190 8.624619 8.762877 8.548558 8.510328 8.805477
##  [81] 8.767357 8.775337 8.800529 8.656734 8.685970 8.844659 8.641129 8.756485
##  [89] 8.747045 8.536738 8.344983 8.273726 8.414019 8.594782 8.387517 8.613090
##  [97] 8.550943 8.643679 8.609205 8.702062 8.585046 8.578405 8.681227 8.677619
```

```
## [105] 8.736797 8.559057 8.583166 8.759194 8.745945 8.846068 8.786994 8.721261
## [113] 8.679956 8.783422 8.764991 8.720554 8.751206 8.709379 8.841465 8.573899
## [121] 8.732268 8.810301 8.697824 8.634059 8.458941 8.511786 8.690284 8.697989
## [129] 8.642604 8.756805 8.791809 8.637851 8.694446 8.755496 8.724286 8.757756
## [137] 8.666483 8.805989 8.642118 8.754937 8.647607 8.665210 8.766759 8.811372
## [145] 8.730283 8.783634 8.732708 8.758806 8.450335 8.574905 8.442653 8.621440
## [153] 8.710849 8.777111 8.718662 8.831800 8.782733 8.667342 8.666788 8.835305
## [161] 8.734479 8.697742 8.732264 8.770856 8.804313 8.741576 8.725571 8.652596
## [169] 8.665236 8.671348 8.640707 8.674062 8.705344 8.735470 8.629435 8.678043
## [177] 8.676818 8.591331 8.512016 8.489941 8.599134 8.580836 8.602056 8.510142
## [185] 8.538130 8.544350 8.567759 8.510412 8.476708 8.517720 8.587961 8.535630
## [193] 8.555132 8.590504 8.572080 8.514374 8.673258 8.691992 8.711905 8.747276
## [201] 8.712370 8.664030 8.658363 8.650095 8.670672 8.641890 8.775694 8.586211
## [209] 8.729497 8.677130 8.704222 8.754478 8.779437 8.645726 8.755598 8.810540
## [217] 8.729863 8.618113 8.754446 8.834400 8.796199 8.705998 8.799598 8.675386
## [225] 8.759608 8.736118 8.716408 8.646978 8.692834 8.638817 8.708001 8.659103
## [233] 8.675441 8.709978 8.709524 8.687164 8.657585 8.763477 8.743210 8.642066
## [241] 8.651530 8.708512 8.706908 8.473242 8.549557 8.503082 8.552217 8.442463
## [249] 8.576976 8.494738 8.636503 8.606617 8.598934 8.627021 8.603673 8.707008
## [257] 8.634945 8.663008 8.662742 8.695671 8.651863 8.713789 8.651999 8.710345
## [265] 8.652513 8.648350 8.779241 8.780798 8.749857 8.751452 8.448312 8.541691
## [273] 8.429336 8.578076 8.564515 8.555603 8.698176 8.616688 8.615564 8.637692
## [281] 8.717384 8.685679 8.602375 8.683269 8.703856 8.723024 8.687304 8.660220
## [289] 8.685849 8.674367 8.721226 8.680543 8.672932 8.607268 8.726758 8.651386
## [297] 8.661322 8.535290 8.513633 8.640612 8.398734 8.577394 8.444281 8.404604
## [305] 8.334756 8.413235 8.438358 8.432864 8.473073 8.481536 8.518168 8.503369
## [313] 8.496973 8.357733 8.510107 8.561962 8.323474 8.503790 8.555604 8.460287
## [321] 8.469098 8.490405 8.532889 8.629313 8.639193 8.557250 8.586759 8.461070
## [329] 8.452248 8.396910 8.381871 8.280540 8.411855 8.301216 8.414903 8.374217
## [337] 8.416240 8.523371 8.355029 8.529370 8.483225 8.476218 8.398775 8.510120
## [345] 8.444843 8.760103 8.714952 8.759775 8.657004 8.561294 8.650041 8.700640
## [353] 8.706162 8.701166 8.378780 8.381066 8.391579 8.317519 8.270967 8.455338
## [361] 8.440930 8.429550 8.536361 8.576667 8.500702 8.523929 8.524295 8.516997
## [369] 8.494056 8.560059 8.396033 8.633164 8.535918 8.526412 8.646472 8.688425
## [377] 8.666813 8.630137 8.658564 8.498184 8.442780 8.682314 8.552838 8.412932
## [385] 8.514787 8.530970 8.759764 8.745756 8.426727 8.478672 8.511013 8.422435
## [393] 8.538171 8.363786 8.311730 8.358886 8.452560 8.476179 8.454851 8.330366
## [401] 8.399204 8.524537 8.444817 8.456761 8.490196 8.611359 8.487227 8.440279
## [409] 8.417133 8.353301 8.531933 8.407683 8.521173 8.464212 8.469161 8.476653
## [417] 8.574114 8.373041 8.390111 8.427264 8.414803 8.276647 8.290631 8.315729
## [425] 8.276102 8.297026 8.330823 8.497059 8.346272 8.354200 8.491852 8.406491
## [433] 8.472630 8.434302 8.335213 8.364285 8.381289 8.457654 8.478552 8.428665
## [441] 8.445371 8.454531 8.435315 8.500250 8.616705 8.552369 8.442701 8.409670
## [449] 8.426731 8.359142 8.302433 8.310319 8.332380 8.206539 8.237105 8.352432
## [457] 8.335531 8.414830 8.322713 8.435411 8.438834 8.369884 8.366048 8.447397
## [465] 8.375874 8.325540 8.510240 8.365207 8.381170 8.406398 8.442830 8.379371
## [473] 8.438607 8.367685 8.405327 8.445025 8.428358 8.351132 8.376490 8.309277
## [481] 8.313497 8.440209 8.380155 8.285865 8.400068 8.374394 8.351134 8.380867
## [489] 8.391260 8.295679 8.395641 8.503159 8.394486 8.417079 8.506419 8.383339
## [497] 8.467154 8.429454 8.524978 8.626737 8.596740 8.478098 8.414742 8.213340
## [505] 8.305010 8.228884 8.180663 8.273080 8.440749 8.490028 8.526553 8.584715
## [513] 8.437732 8.393534 8.539525 8.490855 8.484344 8.291188 8.462782 8.428580
## [521] 8.290031 8.231206 8.453824 8.469494 8.526025 8.585400 8.567489 8.554195
## [529] 8.607200 8.519147 8.505195 8.478319 8.527300 8.517376 8.500285 8.513966
```

```
## [537] 8.642831 8.625364 8.577547 8.636528 8.592650 8.542228 8.594906 8.651990
## [545] 8.652433 8.607001 8.523657 8.531874 8.570465 8.500517 8.473552 8.478510
## [553] 8.435325 8.369467 8.411760 8.437175 8.455964 8.464563 8.530659 8.538727
## [561] 8.540906 8.526175 8.576984 8.581587 8.497831 8.559254 8.545195 8.552254
## [569] 8.494636 8.439826 8.508763 8.500803 8.381076 8.380809 8.431460 8.400235
## [577] 8.430225 8.418038 8.469668 8.472517 8.622595 8.518661 8.497220 8.637694
## [585] 8.504848 8.674034 8.602595 8.610171 8.651216 8.636060 8.604527 8.511970
## [593] 8.635962 8.542208 8.592838 8.540125 8.446125 8.540160 8.470771 8.459112
## [601] 8.482958 8.591929 8.504214 8.477065 8.469966 8.451881 8.600605 8.690752
## [609] 8.409744 8.342747
```

```
# Evaluate model performance
xgb_performance <- postResample(pred = xgb_predictions, obs = testing_data$PH)
print(xgb_performance)
```

```
##       RMSE   Rsquared        MAE
## 0.09949995 0.65392554 0.07329683
```

```
# Make predictions using your XGBoost model
xgb_predictions_ph <- predict(xgb_model, xgb_test)

# Convert predictions to a DataFrame
df_predictions <- data.frame(Predicted_PH = xgb_predictions_ph)
```

```
# Save the predictions to an Excel file
library(writexl)
write_xlsx(df_predictions, path = "XGBoost_Predicted_PH.xlsx")  # Specify a valid file name

cat("Predictions saved to XGBoost_Predicted_PH.xlsx\n")
```

Convert XGBoost prediction to an Excel format

```
## Predictions saved to XGBoost_Predicted_PH.xlsx
```

```
# Get feature importance
feature_importance <- xgb.importance(model = xgb_model, feature_names = colnames(training_data[,-which(

# Print the feature importance
print(feature_importance)
```

Important variable

```
##                Feature        Gain        Cover    Frequency
##                 <char>       <num>        <num>        <num>
##  1:          Mnf.Flow 0.138308917 2.823578e-02 0.0265882353
##  2:        Usage.cont 0.072899904 5.433195e-02 0.0444705882
##  3:          Alch.Rel 0.067016030 2.780176e-02 0.0223529412
##  4:     Oxygen.Filler 0.066814014 4.702940e-02 0.0401176471
##  5:       Temperature 0.047848062 2.734975e-02 0.0298823529
##  6:     Air.Pressurer 0.044609621 2.455323e-02 0.0250588235
##  7:   Pressure.Vacuum 0.039823809 2.373484e-02 0.0231764706
##  8:         Carb.Flow 0.039741424 4.332808e-02 0.0415294118
##  9:          Carb.Rel 0.037230076 2.373992e-02 0.0210588235
```

```
## 10:           Density 0.037132590 2.377590e-02 0.0181176471
## 11:       Brand.CodeC 0.035551031 2.857127e-03 0.0030588235
## 12:    Carb.Pressure1 0.029214934 5.345412e-02 0.0423529412
## 13:     Bowl.Setpoint 0.028904912 2.502870e-03 0.0020000000
## 14:      Filler.Speed 0.028768489 3.970339e-02 0.0318823529
## 15:               MFR 0.028072778 5.945343e-02 0.0398823529
## 16:       Fill.Ounces 0.026407999 4.859541e-02 0.0631764706
## 17:      Filler.Level 0.025448352 4.143871e-02 0.0356470588
## 18:       Carb.Volume 0.020823332 2.929386e-02 0.0885882353
## 19:           Balling 0.020705277 3.223388e-02 0.0176470588
## 20:     Hyd.Pressure3 0.018802664 2.081632e-02 0.0182352941
## 21:     Hyd.Pressure2 0.017189851 2.923521e-02 0.0238823529
## 22:               PSC 0.016760235 4.312475e-02 0.0475294118
## 23:     Carb.Pressure 0.014880526 3.671019e-02 0.0447058824
## 24:         PC.Volume 0.014378534 5.202497e-02 0.0608235294
## 25:       Balling.Lvl 0.013370916 2.900178e-02 0.0220000000
## 26:         Carb.Temp 0.013327348 3.583628e-02 0.0361176471
## 27:          PSC.Fill 0.012985824 2.814937e-02 0.0351764706
## 28:     Hyd.Pressure1 0.011727615 2.970012e-02 0.0257647059
## 29:     Fill.Pressure 0.007626644 2.812865e-02 0.0255294118
## 30:     Hyd.Pressure4 0.006393501 1.424536e-02 0.0205882353
## 31:           PSC.CO2 0.005867658 1.292335e-02 0.0170588235
## 32: Pressure.Setpoint 0.004075923 4.765658e-03 0.0028235294
## 33:       Brand.CodeA 0.003786009 1.532768e-03 0.0011764706
## 34:       Brand.CodeD 0.001870507 6.099793e-05 0.0001176471
## 35:       Brand.CodeB 0.001634697 3.307964e-04 0.0018823529
##               Feature        Gain        Cover    Frequency
```

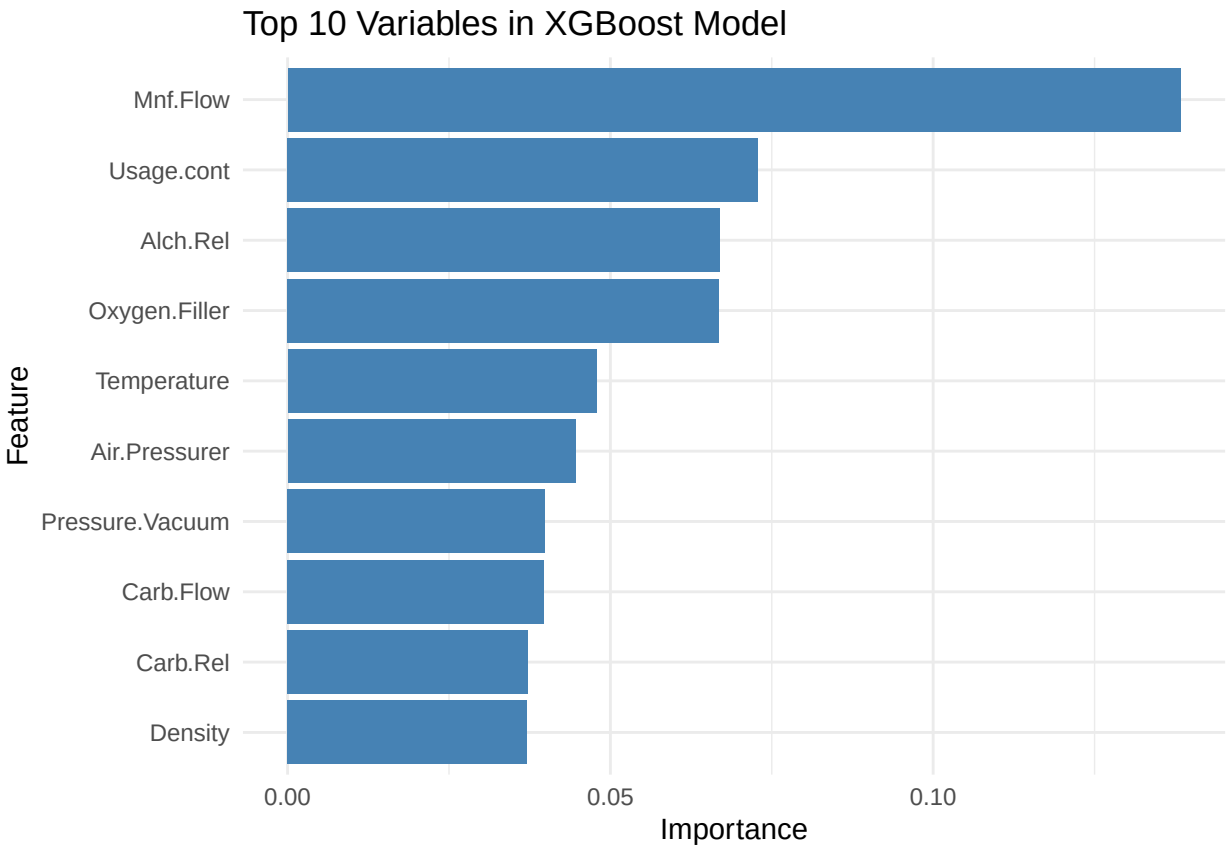
```r
# Extract the most important variable
most_important_variable <- feature_importance$Feature[which.max(feature_importance$Gain)]
cat("Most Important Variable:", most_important_variable, "\n")
```

```
## Most Important Variable: Mnf.Flow
```

```r
# Visualize the top 10 variables in XGBoost
# Compute feature importance
feature_importance <- xgb.importance(model = xgb_model, feature_names = colnames(training_data[,-which(

# Select top 10 most important variables
top_10_features <- head(feature_importance, 10)

# Create a bar plot using ggplot2
ggplot(top_10_features, aes(x = reorder(Feature, Gain), y = Gain)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Top 10 Variables in XGBoost Model",
    x = "Feature",
    y = "Importance"
  ) +
  theme_minimal()
```



XGBoost Performance on the test/evaluation dataset We begin by loading the evaluation data

```
# Load the evaluation data
eval_data <- read_csv("evaluation_data.csv")  # Load evaluation data
```

```
## Rows: 267 Columns: 35
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (34): Carb.Volume, Fill.Ounces, PC.Volume, Carb.Pressure, Carb.Temp, PSC...
## lgl  (1): PH
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Then, we ensure there is consistency between train_x and eval_data

```
# Ensure consistency between train_x and eval_data
# Assuming train_x already has the columns: Brand.CodeA, Brand.CodeB, Brand.CodeC, Brand.CodeD

# Check the columns of both datasets
train_columns <- colnames(train_x)
eval_columns <- colnames(eval_data)

# Add missing columns to eval_data with default 0 values
missing_columns <- setdiff(train_columns, eval_columns)
for (col in missing_columns) {
  eval_data[[col]] <- 0  # Add the missing columns and set to 0
}
```

```r
# Reorder eval_data columns to match train_x column order
eval_data <- eval_data[, train_columns]
```

Next, we prepare the recipe for both training and evaluation datasets

```r
library(recipes)
# Ensure `train_x` exists and contains all necessary predictors
recipe_obj_xgb <- recipe(~ ., data = train_x) |>
  step_impute_bag(all_predictors())  # Add other preprocessing steps if needed

# Prepare the recipe for both training and evaluation datasets
recipe_prepped_xgb <- prep(recipe_obj_xgb)
```

Next, we Impute missing values in both train_x and eval_data

```r
# Impute missing values in both train_x and eval_data
train_x_imputed_xgb <- bake(recipe_prepped_xgb, new_data = train_x)
eval_data_imputed_xgb <- bake(recipe_prepped_xgb, new_data = eval_data)
```

Then, we convert imputed evaluation data to XGBoost DMatrix

```r
# Convert imputed evaluation data to XGBoost DMatrix
eval_matrix <- xgb.DMatrix(data = as.matrix(eval_data_imputed_xgb))
```

We make predictions using the XGBoost model

```r
# Make predictions using the XGBoost model
xgb_eval_predictions <- predict(xgb_model, newdata = eval_matrix)
```

As a final step, we check if the number of rows in our eval data matches the length of predictions

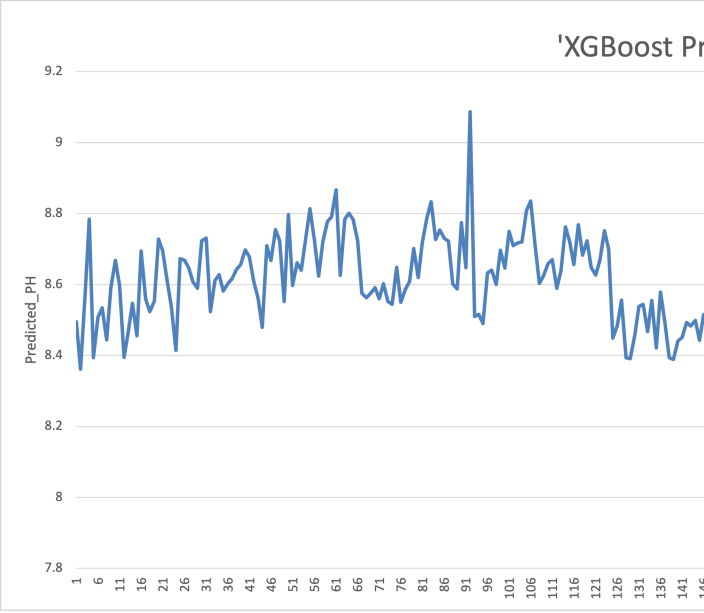
```r
# Check if the number of rows in your eval data matches the length of predictions
if (nrow(eval_data) != length(xgb_eval_predictions)) {
  # If they don't match, subset test_y to match the number of predictions
  test_y <- test_y[1:length(xgb_eval_predictions)]
}

# Now, both xgb_eval_predictions and test_y should have the same length
xgb_eval_performance <- postResample(pred = xgb_eval_predictions, obs = test_y)
```

```
## Warning in pred - obs: longer object length is not a multiple of shorter object
## length
## Warning in pred - obs: longer object length is not a multiple of shorter object
## length
```

```r
# Print performance
print(xgb_eval_performance)
```

```
##      RMSE  Rsquared       MAE
## 0.2083318        NA 0.1657437
```



Save XGboost predictions of PH values to an Excel file

```
library(openxlsx)

# Create a data frame for predictions
predictions_df <- data.frame(Predicted_PH = xgb_eval_predictions)

# Save predictions to an Excel file
write.xlsx(predictions_df, file = "XGBoost_Predicted_PH.xlsx", sheetName = "Predictions", rowNames = FA

cat("Predictions saved to XGBoost_Predicted_PH.xlsx\n")
```

```
## Predictions saved to XGBoost_Predicted_PH.xlsx
```

```
# Read the Excel file into R
predictions_read <- read.xlsx("XGBoost_Predicted_PH.xlsx", sheet = "Predictions")

# Display the imported data
head(predictions_read)
```

```
##   Predicted_PH
## 1     8.560360
## 2     8.454763
## 3     8.442389
## 4     8.689316
## 5     8.466005
## 6     8.556049
```