```
!pip install bayesian-optimization
```

```
Collecting bayesian-optimization
  Downloading bayesian_optimization-1.4.3-py3-none-any.whl (18 kB)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.25.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.11.4)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization)
Collecting colorama>=0.4.6 (from bayesian-optimization)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->b
Installing collected packages: colorama, bayesian-optimization
Successfully installed bayesian-optimization-1.4.3 colorama-0.4.6
```

```
!git clone https://github.com/808ss/thesis.git
```

```
Cloning into 'thesis'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (27/27), 311.32 KiB | 7.59 MiB/s, done.
```

```
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from bayes_opt import BayesianOptimization
```

```
random_seed = 808
np.random.seed(random_seed)
```

# MBBR

## Importing MBBR and Splitting

```
MBBR = pd.read_csv('thesis/MBBR-Chlorination.csv')
MBBR.drop(columns='Date',inplace=True)
```

```
X_orig_MBBR = MBBR.drop(columns='Total Coliform Effluent (MPN/100mL)')
y_orig_MBBR = MBBR['Total Coliform Effluent (MPN/100mL)']
X_train_orig_MBBR, X_test_orig_MBBR, y_train_orig_MBBR, y_test_orig_MBBR = train_test_split(X_orig_MBBR,
                                                                                        y_orig_MBBR,
                                                                                        test_size = 0.3,
                                                                                        random_state=808)
```

```
df_train_orig_MBBR = pd.concat([X_train_orig_MBBR,y_train_orig_MBBR], axis=1)
df_test_orig_MBBR = pd.concat([X_test_orig_MBBR,y_test_orig_MBBR], axis=1)
```

### Data Analysis for Raw Dataset

```
missing_rate_MBBR = [(MBBR.isnull().sum()[val]/MBBR.shape[0])*100 for val in range(0,MBBR.shape[1])]
```

```
pd.options.display.float_format = '{:,.2f}'.format
MBBR_transposed = MBBR.describe().T
MBBR_transposed['Missingness Rate'] = missing_rate_MBBR
```

```
MBBR_transposed
```

| | count | mean | std | min | 25% | 50% | 75% | max | Missingness Rate |
|---|---|---|---|---|---|---|---|---|---|
| **Flow Rate Influent (m3/d)** | 332.00 | 4,787.53 | 2,211.48 | 197.00 | 3,344.00 | 4,709.50 | 6,232.00 | 11,147.00 | 0.0 |
| **Total Coliform Influent (MPN/100mL)** | 270.00 | 290,896,939.26 | 733,941,441.61 | 1,600.00 | 17,250,000.00 | 40,500,000.00 | 160,000,000.00 | 5,200,000,000.00 | 18.6 |
| **Total Coliform Effluent (MPN/100mL)** | 329.00 | 733,375.06 | 9,402,984.69 | 0.00 | 2.00 | 10.00 | 471.00 | 143,900,000.00 | 0.9 |
| **Fecal Coliform Influent (MPN/100mL)** | 103.00 | 236,377,087.38 | 621,705,589.64 | 230,000.00 | 8,550,000.00 | 23,000,000.00 | 37,650,000.00 | 3,000,000,000.00 | 68.9 |
| **Fecal Coliform Effluent (MPN/100mL)** | 171.00 | 746.87 | 3,947.85 | 2.00 | 10.00 | 10.00 | 10.00 | 24,196.00 | 48.4 |
| **BOD Influent (ppm)** | 273.00 | 152.40 | 148.02 | 8.00 | 68.00 | 119.00 | 196.00 | 1,425.00 | 17.7 |
| **BOD Pre-chlorination\n(ppm)** | 274.00 | 11.28 | 12.82 | 1.00 | 4.00 | 8.00 | 14.00 | 119.00 | 17.4 |

## Data Analysis for Training Set (Pre-Imputation)

```
missing_rate_train_orig_MBBR = [(df_train_orig_MBBR.isnull().sum()[val]/df_train_orig_MBBR.shape[0])*100 for val in range(0,df_t
```

```
pd.options.display.float_format = '{:,.2f}'.format
#pd.set_option('display.float_format', '{:e}'.format)
df_train_orig_MBBR_transposed = df_train_orig_MBBR.describe().T
df_train_orig_MBBR_transposed['Missingness Rate'] = missing_rate_train_orig_MBBR
```

```
df_train_orig_MBBR_transposed
```

| | count | mean | std | min | 25% | 50% | 75% | max | Missingness Rate |
|---|---|---|---|---|---|---|---|---|---|
| **Flow Rate Influent (m3/d)** | 232.00 | 4,882.74 | 2,204.80 | 197.00 | 3,344.00 | 4,762.00 | 6,349.00 | 10,999.00 | 0.0 |
| **Total Coliform Influent (MPN/100mL)** | 185.00 | 315,522,010.81 | 790,769,090.66 | 16,000.00 | 18,000,000.00 | 41,000,000.00 | 160,000,000.00 | 5,200,000,000.00 | 20.2 |
| **Total Coliform Effluent (MPN/100mL)** | 230.00 | 1,045,242.88 | 11,238,969.87 | 0.00 | 2.25 | 10.00 | 1,280.75 | 143,900,000.00 | 0.8 |
| **Fecal Coliform Influent (MPN/100mL)** | 68.00 | 298,124,117.65 | 669,480,560.27 | 230,000.00 | 10,400,000.00 | 24,000,000.00 | 40,950,000.00 | 2,600,000,000.00 | 70.6 |
| **Fecal Coliform Effluent (MPN/100mL)** | 120.00 | 892.02 | 4,352.93 | 2.00 | 10.00 | 10.00 | 10.00 | 24,196.00 | 48.2 |
| **BOD Influent (ppm)** | 187.00 | 162.30 | 167.60 | 8.00 | 70.50 | 122.00 | 199.00 | 1,425.00 | 19.4 |
| **BOD Pre-chlorination\n(ppm)** | 188.00 | 11.12 | 12.31 | 1.00 | 5.00 | 8.00 | 14.00 | 119.00 | 18.9 |

## Data Analysis for Testing Set (Pre-imputation)

```
missing_rate_test_orig_MBBR = [(df_test_orig_MBBR.isnull().sum()[val]/df_test_orig_MBBR.shape[0])*100 for val in range(0,df_test
```

```
#pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.float_format', '{:e}'.format)
df_test_orig_MBBR_transposed = df_test_orig_MBBR.describe().T
df_test_orig_MBBR_transposed['Missingness Rate'] = missing_rate_test_orig_MBBR
```

```
df_test_orig_MBBR_transposed
```

| | count | mean | std | min | 25% | 50% | 75% | max | Missing |
|---|---|---|---|---|---|---|---|---|---|
| **Flow Rate Influent (m3/d)** | 1.000000e+02 | 4.566650e+03 | 2.222244e+03 | 2.170000e+02 | 3.343750e+03 | 4.641500e+03 | 6.112250e+03 | 1.114700e+04 | 0.000000 |
| **Total Coliform Influent (MPN/100mL)** | 8.500000e+01 | 2.373012e+08 | 5.924907e+08 | 1.600000e+03 | 1.700000e+07 | 4.000000e+07 | 1.600000e+08 | 3.500000e+09 | 1.500000 |
| **Total Coliform Effluent (MPN/100mL)** | 9.900000e+01 | 8.833667e+03 | 3.764360e+04 | 0.000000e+00 | 2.000000e+00 | 1.000000e+01 | 9.000000e+01 | 2.419600e+05 | 1.000000 |
| **Fecal Coliform Influent (MPN/100mL)** | 3.500000e+01 | 1.164114e+08 | 5.038721e+08 | 1.000000e+06 | 7.450000e+06 | 1.700000e+07 | 3.045000e+07 | 3.000000e+09 | 6.500000 |
| **Fecal Coliform Effluent (MPN/100mL)** | 5.100000e+01 | 4.053529e+02 | 2.779390e+03 | 2.000000e+00 | 2.000000e+00 | 1.000000e+01 | 1.000000e+01 | 1.986300e+04 | 4.900000 |
| **BOD Influent (ppm)** | 8.600000e+01 | 1.308605e+02 | 8.921358e+01 | 1.900000e+01 | 6.600000e+01 | 1.060000e+02 | 1.830000e+02 | 4.090000e+02 | 1.400000 |
| **BOD Pre-chlorination\n(ppm)** | 8.600000e+01 | 1.162791e+01 | 1.393434e+01 | 1.000000e+00 | 4.000000e+00 | 8.000000e+00 | 1.400000e+01 | 1.080000e+02 | 1.400000 |

## Data Imputation

## Exporting Datasets to R

```
df_train_orig_MBBR.to_csv('MBBR_train_set.csv',index=False)
df_test_orig_MBBR.to_csv('MBBR_test_set.csv',index=False)

# Export to R for mixgb
```

## Mixgb imputation

```
library(mixgb)
library(openxlsx)
set.seed(808)

MBBR_train_set <- read.csv("C:/Users/nikko/PycharmProjects/Thesis/MBBR_train_set.csv")
MBBR_test_set <- read.csv("C:/Users/nikko/PycharmProjects/Thesis/MBBR_test_set.csv")

MBBR_train_set_df = as.data.frame(MBBR_train_set)
MBBR_test_set_df = as.data.frame(MBBR_test_set)

clean_MBBR_train_set_df <- data_clean(MBBR_train_set_df)
clean_MBBR_test_set_df <- data_clean(MBBR_test_set_df)

cv.results_1 <- mixgb_cv(data = clean_MBBR_train_set_df, nrounds = 5000, verbose = FALSE)
cv.results_1$evaluation.log
cv.results_1$best.nrounds

mixgb_obj <- mixgb(data = clean_MBBR_train_set_df, m = 5, nrounds = cv.results_1$best.nrounds, save.models = TRUE)
MBBR_train_imputed <- mixgb_obj$imputed.data

MBBR_test.imputed <- impute_new(object = mixgb_obj, newdata = clean_MBBR_test_set_df)

write.xlsx(MBBR_train_imputed[[1]], file = 'mbbr_m1_imputed_train.xlsx')
write.xlsx(MBBR_train_imputed[[2]], file = 'mbbr_m2_imputed_train.xlsx')
write.xlsx(MBBR_train_imputed[[3]], file = 'mbbr_m3_imputed_train.xlsx')
write.xlsx(MBBR_train_imputed[[4]], file = 'mbbr_m4_imputed_train.xlsx')
write.xlsx(MBBR_train_imputed[[5]], file = 'mbbr_m5_imputed_train.xlsx')

write.xlsx(MBBR_test.imputed[[1]], file = 'mbbr_m1_imputed_test.xlsx')
write.xlsx(MBBR_test.imputed[[2]], file = 'mbbr_m2_imputed_test.xlsx')
write.xlsx(MBBR_test.imputed[[3]], file = 'mbbr_m3_imputed_test.xlsx')
write.xlsx(MBBR_test.imputed[[4]], file = 'mbbr_m4_imputed_test.xlsx')
write.xlsx(MBBR_test.imputed[[5]], file = 'mbbr_m5_imputed_test.xlsx')
```

## Import imputed datasets from R

```
dfs = []
for val in range(1,6):
  source = f'thesis/mbbr_m{val}_imputed_train.xlsx'
```

```
  dfs.append(pd.read_excel(source))

average_MBBR_train = pd.concat(dfs).groupby(level=0).mean()


dfs = []
for val in range(1,6):
  source = f'thesis/mbbr_m{val}_imputed_test.xlsx'
  dfs.append(pd.read_excel(source))

average_MBBR_test = pd.concat(dfs).groupby(level=0).mean()
```

## Data Analysis for Training Set (Post-Imputation)

```
#pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.float_format', '{:e}'.format)
average_MBBR_train_transposed = average_MBBR_train.describe().T
```

```
average_MBBR_train_transposed
```

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **Flow.Rate.Influent..m3.d.** | 2.320000e+02 | 4.882741e+03 | 2.204801e+03 | 1.970000e+02 | 3.344000e+03 | 4.762000e+03 | 6.349000e+03 | 1.09990 |
| **Total.Coliform.Influent..MPN.100mL.** | 2.320000e+02 | 3.480306e+08 | 7.784700e+08 | 1.600000e+04 | 2.175000e+07 | 5.400000e+07 | 2.200000e+08 | 5.20000 |
| **Total.Coliform.Effluent..MPN.100mL.** | 2.320000e+02 | 1.036336e+06 | 1.119062e+07 | 0.000000e+00 | 2.750000e+00 | 1.000000e+01 | 1.600000e+03 | 1.43900 |
| **Fecal.Coliform.Influent..MPN.100mL.** | 2.320000e+02 | 1.965840e+08 | 5.001635e+08 | 2.300000e+05 | 1.428500e+07 | 2.740000e+07 | 3.821000e+07 | 2.60000 |
| **Fecal.Coliform.Effluent..MPN.100mL.** | 2.320000e+02 | 4.182178e+03 | 8.256544e+03 | 2.000000e+00 | 8.800000e+00 | 1.000000e+01 | 2.282500e+02 | 2.41960 |
| **BOD.Influent..ppm.** | 2.320000e+02 | 1.585655e+02 | 1.533443e+02 | 8.000000e+00 | 7.585000e+01 | 1.250000e+02 | 1.950500e+02 | 1.42500 |
| **BOD.Pre.chlorination..ppm.** | 2.320000e+02 | 1.199138e+01 | 1.246310e+01 | 1.000000e+00 | 5.000000e+00 | 9.000000e+00 | 1.500000e+01 | 1.19000 |
| **COD.Influent..ppm.** | 2.320000e+02 | 3.458931e+02 | 5.439009e+02 | 1.300000e+01 | 1.750000e+02 | 2.510000e+02 | 3.745000e+02 | 7.73400 |
| **COD.Pre.chlorination..ppm.** | 2.320000e+02 | 4.940431e+01 | 3.795535e+01 | 5.000000e+00 | 2.475000e+01 | 4.150000e+01 | 6.300000e+01 | 3.43000 |
| **TSS.Pre.chlorination..ppm.** | 2.320000e+02 | 1.756897e+01 | 2.094736e+01 | 1.000000e+00 | 6.000000e+00 | 1.200000e+01 | 2.000000e+01 | 1.60000 |
| **pH.Pre.chlorination** | 2.320000e+02 | 7.185121e+00 | 3.010609e-01 | 6.120000e+00 | 7.000000e+00 | 7.200000e+00 | 7.362500e+00 | 8.38000 |
| **Chlorine.dosage..L.d.** | 2.320000e+02 | 8.721017e+02 | 4.852711e+02 | 0.000000e+00 | 6.000000e+02 | 8.500000e+02 | 1.160000e+03 | 2.80000 |
| **Residual.chlorine..ppm.** | 2.320000e+02 | 2.082141e+00 | 1.913163e+00 | 0.000000e+00 | 3.815000e-01 | 1.205700e+00 | 3.917150e+00 | 5.48000 |

## Data Analysis for Testing Set (Post-Imputation)

```
pd.options.display.float_format = '{:,.2f}'.format
#pd.set_option('display.float_format', '{:e}'.format)
average_MBBR_test_transposed = average_MBBR_test.describe().T
```

```
average_MBBR_test_transposed
```

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **Flow.Rate.Influent..m3.d.** | 100.00 | 4,566.65 | 2,222.24 | 217.00 | 3,343.75 | 4,641.50 | 6,112.25 | 11,147 |
| **Total.Coliform.Influent..MPN.100mL.** | 100.00 | 288,973,956.00 | 647,069,268.14 | 1,600.00 | 22,750,000.00 | 45,000,000.00 | 200,000,000.00 | 3,500,000,000 |
| **Total.Coliform.Effluent..MPN.100mL.** | 100.00 | 8,933.07 | 37,466.19 | 0.00 | 2.00 | 10.00 | 134.75 | 241,960 |
| **Fecal.Coliform.Influent..MPN.100mL.** | 100.00 | 147,284,740.00 | 427,084,482.81 | 1,000,000.00 | 12,000,000.00 | 25,320,000.00 | 41,560,000.00 | 3,000,000,000 |
| **Fecal.Coliform.Effluent..MPN.100mL.** | 100.00 | 2,363.05 | 6,219.15 | 2.00 | 10.00 | 10.00 | 107.30 | 24,196 |
| **BOD.Influent..ppm.** | 100.00 | 134.13 | 88.52 | 19.00 | 66.45 | 108.40 | 188.25 | 409 |
| **BOD.Pre.chlorination..ppm.** | 100.00 | 12.06 | 13.39 | 1.00 | 5.00 | 9.00 | 15.00 | 108 |
| **COD.Influent..ppm.** | 100.00 | 268.66 | 192.32 | 41.00 | 135.75 | 210.00 | 357.25 | 1,256 |
| **COD.Pre.chlorination..ppm.** | 100.00 | 45.03 | 47.09 | 5.00 | 20.00 | 29.70 | 53.55 | 314 |
| **TSS.Pre.chlorination..ppm.** | 100.00 | 18.00 | 24.39 | 1.00 | 5.00 | 10.00 | 19.25 | 164 |
| **pH.Pre.chlorination** | 100.00 | 7.21 | 0.37 | 5.28 | 7.10 | 7.21 | 7.40 | 8 |
| **Chlorine.dosage..L.d.** | 100.00 | 809.32 | 498.65 | 0.00 | 488.70 | 748.90 | 1,005.05 | 2,900 |
| **Residual.chlorine..ppm.** | 100.00 | 2.14 | 1.69 | 0.01 | 0.49 | 2.05 | 3.57 | 5 |

## Exhaustive Feature Selection

### For Imputed Dataset

```
pd.reset_option('display.float_format')
```

```
X_train_MBBR = average_MBBR_train.drop(columns=['Residual.chlorine..ppm.','Total.Coliform.Effluent..MPN.100mL.','Fecal.Coliform.
y_train_MBBR = average_MBBR_train['Total.Coliform.Effluent..MPN.100mL.']
X_test_MBBR = average_MBBR_test.drop(columns=['Residual.chlorine..ppm.','Total.Coliform.Effluent..MPN.100mL.','Fecal.Coliform.Ef
y_test_MBBR = average_MBBR_test['Total.Coliform.Effluent..MPN.100mL.']

features_wo_chlorine_dosage = X_train_MBBR.columns[:-1]
features_wo_chlorine_dosage
```

```
Index(['Flow.Rate.Influent..m3.d.', 'Total.Coliform.Influent..MPN.100mL.',
       'Fecal.Coliform.Influent..MPN.100mL.', 'BOD.Influent..ppm.',
       'BOD.Pre.chlorination..ppm.', 'COD.Influent..ppm.',
       'COD.Pre.chlorination..ppm.', 'TSS.Pre.chlorination..ppm.',
       'pH.Pre.chlorination'],
      dtype='object')
```

```
# Generate all combinations of the other features
combinations = []
for r in range(1, len(features_wo_chlorine_dosage) + 1):
    combinations.extend(itertools.combinations(features_wo_chlorine_dosage, r))

# Add the first feature to each combination
combinations = [(X_train_MBBR.columns[-1],) + combo for combo in combinations]
```

```
params = {'objective': 'reg:squarederror'}

results = []
for combo in combinations:
    dtrain = xgb.DMatrix(X_train_MBBR[list(combo)], label=y_train_MBBR)
    cv_result = xgb.cv(params, dtrain, num_boost_round=10, nfold=5, metrics='rmse', seed=808)
    last_round_metrics = cv_result.iloc[-1]
    results.append([combo, last_round_metrics['train-rmse-mean'], last_round_metrics['test-rmse-mean'],
                    last_round_metrics['train-rmse-std'],last_round_metrics['test-rmse-std']])

results_df_MBBR = pd.DataFrame(results, columns=['Combination', 'Train RMSE', 'Validation RMSE', 'Train RMSE Std. Dev.', ' Valic
```

```
results_df_MBBR.sort_values(by='Validation RMSE')
```

| | Combination | Train RMSE | Validation RMSE | Train RMSE Std. Dev. | Validation RMSE Std. Dev |
|---|---|---|---|---|---|
| **22** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143679e+06 | 6.973836e+06 | 433716.475063 | 8.686581e+06 |
| **89** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143670e+06 | 6.974175e+06 | 433716.373885 | 8.686855e+06 |
| **77** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143647e+06 | 6.974453e+06 | 433716.834645 | 8.686765e+06 |
| **195** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143643e+06 | 6.974751e+06 | 433718.284515 | 8.686897e+06 |
| **82** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143649e+06 | 6.975425e+06 | 433719.035511 | 8.685778e+06 |
| **...** | ... | ... | ... | ... | ... |
| **496** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143576e+06 | 1.775114e+07 | 433675.592797 | 2.464120e+06 |
| **455** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143577e+06 | 1.775115e+07 | 433674.532288 | 2.464128e+06 |
| **458** | (Chlorine.dosage..L.d., Fecal.Coliform.Influen... | 2.143571e+06 | 1.775116e+07 | 433680.718785 | 2.464060e+06 |
| **452** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143572e+06 | 1.775118e+07 | 433679.779721 | 2.464035e+06 |
| **493** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143572e+06 | 1.775118e+07 | 433679.908052 | 2.464035e+06 |

511 rows × 5 columns

```
results_df_MBBR.sort_values(by='Validation RMSE').iloc[0:3]
```

| | Combination | Train RMSE | Validation RMSE | Train RMSE Std. Dev. | Validation RMSE Std. Dev |
|---|---|---|---|---|---|
| **22** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143679e+06 | 6.973836e+06 | 433716.475063 | 8.686581e+06 |
| **89** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143670e+06 | 6.974175e+06 | 433716.373885 | 8.686855e+06 |
| **77** | (Chlorine.dosage..L.d., Total.Coliform.Influen... | 2.143647e+06 | 6.974453e+06 | 433716.834645 | 8.686765e+06 |

```
results_df_MBBR.sort_values(by='Validation RMSE').iloc[0]['Combination']
```

```
('Chlorine.dosage..L.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'TSS.Pre.chlorination..ppm.')
```

```
results_df_MBBR.sort_values(by='Validation RMSE').iloc[1]['Combination']
```

```
('Chlorine.dosage..L.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'COD.Influent..ppm.',
 'TSS.Pre.chlorination..ppm.')
```

```
results_df_MBBR.sort_values(by='Validation RMSE').iloc[2]['Combination']
```

```
('Chlorine.dosage..L.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'Fecal.Coliform.Influent..MPN.100mL.',
 'TSS.Pre.chlorination..ppm.')
```

```
optimal_features_MBBR = results_df_MBBR.sort_values(by='Validation RMSE').iloc[0]['Combination']
optimal_features_MBBR
```

```
('Chlorine.dosage..L.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'TSS.Pre.chlorination..ppm.')
```

```
results_df_MBBR['count'] = results_df_MBBR['Combination'].apply(lambda x: len(x))
results_df_MBBR.to_csv('MBBR Exhaustive Feature Selection.csv', index=False)
```

## For Raw Dataset

```
non_imputed_mask_MBBR_train = ~np.isnan(y_train_orig_MBBR)
non_imputed_mask_MBBR_test = ~np.isnan(y_test_orig_MBBR)
```

```
X_train_MBBR_dropped = X_train_orig_MBBR[non_imputed_mask_MBBR_train]
y_train_MBBR_dropped = y_train_orig_MBBR[non_imputed_mask_MBBR_train]
X_test_MBBR_dropped = X_test_orig_MBBR[non_imputed_mask_MBBR_test]
y_test_MBBR_dropped = y_test_orig_MBBR[non_imputed_mask_MBBR_test]
```

```
features_wo_chlorine_dosage_dropped = X_train_MBBR_dropped.columns[:-1]
features_wo_chlorine_dosage_dropped
```

```
Index(['Flow Rate Influent (m3/d)', 'Total Coliform Influent (MPN/100mL)',
       'Fecal Coliform Influent (MPN/100mL)',
       'Fecal Coliform Effluent (MPN/100mL)', 'BOD Influent (ppm)',
       'BOD Pre-chlorination\n(ppm)', 'COD Influent (ppm)',
       'COD Pre-chlorination\n(ppm)', 'TSS Pre-chlorination (ppm)',
       'pH Pre-chlorination', 'Residual chlorine\n(ppm)'],
      dtype='object')
```

```
# Generate all combinations of the other features
combinations = []
for r in range(1, len(features_wo_chlorine_dosage_dropped) + 1):
    combinations.extend(itertools.combinations(features_wo_chlorine_dosage_dropped, r))

# Add the first feature to each combination
combinations = [(X_train_MBBR_dropped.columns[-1],) + combo for combo in combinations]


params = {'objective': 'reg:squarederror'}

results = []
for combo in combinations:
    dtrain = xgb.DMatrix(X_train_MBBR_dropped[list(combo)], label=y_train_MBBR_dropped)
    cv_result = xgb.cv(params, dtrain, num_boost_round=10, nfold=5, metrics='rmse', seed=808)
    last_round_metrics = cv_result.iloc[-1]
    results.append([combo, last_round_metrics['train-rmse-mean'], last_round_metrics['test-rmse-mean'],
                    last_round_metrics['train-rmse-std'],last_round_metrics['test-rmse-std']])

results_df_MBBR_dropped = pd.DataFrame(results, columns=['Combination', 'Train RMSE', 'Validation RMSE', 'Train RMSE Std. Dev.',
```

```
results_df_MBBR_dropped.sort_values(by='Validation RMSE')
```

| | Combination | Train RMSE | Validation RMSE | Train RMSE Std. Dev. | Validation RMSE Std. Dev |
|---|---|---|---|---|---|
| **464** | (Chlorine dosage (L/d), Fecal Coliform Influen... | 2.153240e+06 | 7.055569e+06 | 437515.494145 | 8.776780e+06 |
| **159** | (Chlorine dosage (L/d), Fecal Coliform Influen... | 2.153264e+06 | 7.055577e+06 | 437508.097017 | 8.776333e+06 |
| **89** | (Chlorine dosage (L/d), Flow Rate Influent (m3... | 2.153108e+06 | 7.109512e+06 | 437418.475857 | 8.734094e+06 |
| **273** | (Chlorine dosage (L/d), Flow Rate Influent (m3... | 2.153108e+06 | 7.109512e+06 | 437418.475857 | 8.734094e+06 |
| **20** | (Chlorine dosage (L/d), Flow Rate Influent (m3... | 2.153104e+06 | 7.109978e+06 | 437430.985738 | 8.734234e+06 |
| **...** | ... | ... | ... | ... | ... |
| **1710** | (Chlorine dosage (L/d), Total Coliform Influen... | 2.153240e+06 | 2.215344e+07 | 437550.571627 | 5.784395e+06 |
| **399** | (Chlorine dosage (L/d), Total Coliform Influen... | 2.153254e+06 | 2.215344e+07 | 437543.765876 | 5.784474e+06 |
| **791** | (Chlorine dosage (L/d), Total Coliform Influen... | 2.153254e+06 | 2.215344e+07 | 437543.765876 | 5.784471e+06 |
| **834** | (Chlorine dosage (L/d), Total Coliform Influen... | 2.153465e+06 | 2.215351e+07 | 437452.108605 | 5.784252e+06 |
| **1282** | (Chlorine dosage (L/d), Total Coliform Influen... | 2.153465e+06 | 2.215351e+07 | 437452.108605 | 5.784253e+06 |

2047 rows × 5 columns

```
results_df_MBBR_dropped.sort_values(by='Validation RMSE').iloc[0:3]
```

| | Combination | Train RMSE | Validation RMSE | Train RMSE Std. Dev. | Validation RMSE Std. Dev |
|---|---|---|---|---|---|
| **464** | (Chlorine dosage (L/d), Fecal Coliform Influen... | 2.153240e+06 | 7.055569e+06 | 437515.494145 | 8.776780e+06 |
| **159** | (Chlorine dosage (L/d), Fecal Coliform Influen... | 2.153264e+06 | 7.055577e+06 | 437508.097017 | 8.776333e+06 |
| **89** | (Chlorine dosage (L/d), Flow Rate Influent (m3... | 2.153108e+06 | 7.109512e+06 | 437418.475857 | 8.734094e+06 |

```
results_df_MBBR_dropped.sort_values(by='Validation RMSE').iloc[0]['Combination']
```

```
('Chlorine dosage (L/d)',
 'Fecal Coliform Influent (MPN/100mL)',
 'BOD Influent (ppm)',
 'COD Influent (ppm)',
 'Residual chlorine\n(ppm)')
```

```
results_df_MBBR_dropped.sort_values(by='Validation RMSE').iloc[1]['Combination']
```

```
('Chlorine dosage (L/d)',
 'Fecal Coliform Influent (MPN/100mL)',
 'BOD Influent (ppm)',
 'Residual chlorine\n(ppm)')
```

```
results_df_MBBR_dropped.sort_values(by='Validation RMSE').iloc[2]['Combination']
```

```
('Chlorine dosage (L/d)',
 'Flow Rate Influent (m3/d)',
 'Fecal Coliform Effluent (MPN/100mL)',
 'Residual chlorine\n(ppm)')
```

```
optimal_features_MBBR_dropped = results_df_MBBR_dropped.sort_values(by='Validation RMSE').iloc[0]['Combination']
optimal_features_MBBR_dropped
```

```
('Chlorine dosage (L/d)',
 'Fecal Coliform Influent (MPN/100mL)',
 'BOD Influent (ppm)',
 'COD Influent (ppm)',
 'Residual chlorine\n(ppm)')
```

```
results_df_MBBR_dropped['count'] = results_df_MBBR_dropped['Combination'].apply(lambda x: len(x))
results_df_MBBR_dropped.to_csv('MBBR Dropped Exhaustive Feature Selection.csv', index=False)
```

## Hyperparameter Optimization

### For Imputed Dataset

```
# Convert the data into DMatrix format
dtrain = xgb.DMatrix(X_train_MBBR[list(optimal_features_MBBR)], label=y_train_MBBR)

# Define the function to be optimized
def xgb_evaluate(eta, alpha, lambd, gamma, subsample, col_subsample, max_depth):
    eta = 10**eta
    alpha = 10**alpha
    lambd = 10**lambd
    gamma = 10**gamma
    max_depth = int(round(2**max_depth))

    params = {'eval_metric': 'rmse',
              'objective': 'reg:squarederror',
              'max_depth': max_depth,
              'eta': eta,
              'gamma': gamma,
              'subsample': subsample,
              'alpha': alpha,
              'lambda': lambd,
              'colsample_bytree': col_subsample,}

    cv_result = xgb.cv(params, dtrain, num_boost_round=1000, nfold=5, early_stopping_rounds=30, seed=808)
    return -1.0 * cv_result['test-rmse-mean'].iloc[-1]

# Specify the hyperparameters to be tuned
xgb_bo_MBBR = BayesianOptimization(xgb_evaluate, {'eta': (-3, 0),
                                              'alpha': (-6, 0.3),
                                              'lambd': (-6, 0.3),
                                              'gamma': (-6, 1.8),
                                              'subsample': (0.5, 1),
                                              'col_subsample': (0.3, 1),
                                              'max_depth': (1, 3)},
                              random_state=808)

# Optimize the hyperparameters
xgb_bo_MBBR.maximize(n_iter=1000, init_points=10)# Convert the data into DMatrix format
```

```
|   iter    |  target   |   alpha   | col_su... |    eta    |   gamma   |   lambd   | max_depth | subsample |
-------------------------------------------------------------------------------------------------------------
| 1         | -7.618e+0 | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -7.579e+0 | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -7.939e+0 | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -7.627e+0 | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
```

```
| 5         | -1.103e+0 | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -7.618e+0 | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -7.64e+06 | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -8.601e+0 | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -7.605e+0 | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -8.948e+0 | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -7.617e+0 | -0.6956   | 0.6578    | -2.8      | -1.29     | -2.763    | 1.304     | 0.8132    |
| 12        | -7.616e+0 | -2.266    | 0.6504    | -3.0      | -3.289    | -1.879    | 1.762     | 0.7004    |
| 13        | -7.61e+06 | -0.0363   | 0.9916    | -2.972    | -3.891    | -3.353    | 1.616     | 1.0       |
| 14        | -7.603e+0 | -4.428    | 1.0       | -3.0      | -4.197    | -3.358    | 1.0       | 1.0       |
| 15        | -7.606e+0 | 0.0059    | 1.0       | -3.0      | 1.8       | -1.461    | 1.0       | 0.5       |
| 16        | -7.588e+0 | -5.197    | 1.0       | -2.385    | -5.052    | -1.239    | 3.0       | 0.5       |
| 17        | -7.598e+0 | -4.257    | 1.0       | -0.3635   | -5.334    | -1.945    | 1.0       | 0.5       |
| 18        | -7.599e+0 | -6.0      | 1.0       | -3.0      | -3.836    | 0.3       | 1.0       | 1.0       |
| 19        | -9.984e+0 | -6.0      | 0.3       | 0.0       | -3.941    | 0.3       | 2.116     | 1.0       |
| 20        | -7.604e+0 | -3.85     | 1.0       | -3.0      | -5.723    | -1.056    | 1.0       | 0.5       |
| 21        | -7.597e+0 | -4.802    | 1.0       | -1.568    | -6.0      | -3.914    | 2.78      | 0.5       |
| 22        | -8.96e+06 | -2.328    | 1.0       | -0.1729   | -6.0      | -4.069    | 1.0       | 0.5       |
| 23        | -7.599e+0 | -6.0      | 1.0       | -2.64     | -6.0      | -2.477    | 1.003     | 0.5       |
| 24        | -7.605e+0 | -5.851    | 1.0       | -3.0      | -3.262    | -3.027    | 3.0       | 0.5       |
| 25        | -7.602e+0 | -5.339    | 1.0       | -3.0      | -0.259    | -0.312    | 1.0       | 1.0       |
| 26        | -7.595e+0 | -5.592    | 0.3       | -3.0      | 1.8       | 0.3       | 3.0       | 0.5       |
| 27        | -7.606e+0 | -5.816    | 0.7985    | -2.807    | 0.3534    | -1.984    | 2.959     | 0.6622    |
| 28        | -7.615e+0 | -3.71     | 0.5931    | -2.877    | -0.3826   | -0.2186   | 2.975     | 0.9028    |
| 29        | -7.615e+0 | -6.0      | 0.3       | -3.0      | -5.482    | -6.0      | 3.0       | 1.0       |
| 30        | -7.554e+0 | -6.0      | 1.0       | -0.6008   | 1.8       | 0.3       | 1.0       | 0.5       |
| 31        | -7.605e+0 | -2.944    | 1.0       | -3.0      | -3.836    | -4.395    | 3.0       | 0.5       |
| 32        | -7.615e+0 | 0.3       | 0.3       | -3.0      | -5.54     | -6.0      | 3.0       | 1.0       |
| 33        | -7.605e+0 | 0.3       | 1.0       | -3.0      | 1.8       | -3.182    | 3.0       | 0.5       |
| 34        | -6.967e+0 | -6.0      | 1.0       | 0.0       | 1.8       | -1.196    | 3.0       | 1.0       |
| 35        | -9.565e+0 | -5.967    | 0.5339    | -0.5951   | 1.798     | -2.521    | 2.066     | 0.5897    |
| 36        | -6.97e+06 | -5.95     | 1.0       | 0.0       | 1.752     | -0.03868  | 3.0       | 1.0       |
| 37        | -7.879e+0 | -4.301    | 0.7841    | -0.4484   | 1.193     | 0.1425    | 2.867     | 0.6998    |
| 38        | -7.615e+0 | -4.13     | 0.3       | -3.0      | -4.956    | -2.786    | 3.0       | 1.0       |
| 39        | -7.652e+0 | -5.81     | 0.8819    | -0.8796   | -0.1237   | -0.3333   | 2.881     | 0.9823    |
| 40        | -7.605e+0 | -3.498    | 0.9565    | -2.818    | -1.856    | -2.613    | 2.917     | 0.5531    |
| 41        | -7.688e+0 | -6.0      | 0.3       | -1.732    | -4.536    | -4.968    | 1.0       | 0.5       |
| 42        | -7.615e+0 | -6.0      | 0.3       | -3.0      | -1.687    | -0.9669   | 3.0       | 1.0       |
| 43        | -1.19e+07 | -6.0      | 0.3       | 0.0       | -6.0      | -6.0      | 3.0       | 1.0       |
| 44        | -7.606e+0 | -4.548    | 1.0       | -3.0      | -6.0      | -5.27     | 1.0       | 0.5       |
| 45        | -7.603e+0 | -6.0      | 1.0       | -3.0      | -2.836    | -6.0      | 1.0       | 1.0       |
| 46        | -7.605e+0 | 0.3       | 1.0       | -3.0      | -2.24     | -5.132    | 3.0       | 0.5       |
| 47        | -7.614e+0 | -6.0      | 0.3       | -3.0      | -2.018    | -2.967    | 1.0       | 0.5       |
| 48        | -7.603e+0 | 0.3       | 0.3       | -3.0      | -6.0      | 0.3       | 1.0       | 1.0       |
| 49        | -7.566e+0 | -0.875    | 0.3       | -3.0      | -3.845    | -6.0      | 1.0       | 1.0       |
| 50        | -7.178e+0 | 0.3       | 1.0       | -3.0      | 1.8       | 0.3       | 3.0       | 1.0       |
| 51        | -7.596e+0 | -1.651    | 1.0       | -3.0      | 1.8       | -0.6719   | 3.0       | 0.5       |
| 52        | -7.165e+0 | 0.3       | 1.0       | -3.0      | 1.8       | -6.0      | 3.0       | 1.0       |
| 53        | -7.606e+0 | 0.3       | 1.0       | -3.0      | 1.8       | -6.0      | 1.0       | 0.5       |
| 54        | -7.601e+0 | -0.4824   | 0.564     | -2.975    | -3.071    | 0.2918    | 1.017     | 0.9226    |
| 55        | -7.526e+0 | -3.14     | 1.0       | -3.0      | -2.001    | 0.3       | 1.0       | 0.5       |
| 56        | -8.281e+0 | 0.3       | 1.0       | 0.0       | -6.0      | 0.3       | 1.0       | 0.5       |
```

```python
# Extract the optimal hyperparameters from the Bayesian Optimization object
best_params_MBBR = xgb_bo_MBBR.max['params']

# Transform the hyperparameters from log space to original space
best_params_MBBR['eta'] = 10 ** best_params_MBBR['eta']
best_params_MBBR['alpha'] = 10 ** best_params_MBBR['alpha']
best_params_MBBR['lambda'] = 10 ** best_params_MBBR['lambd']
best_params_MBBR['gamma'] = 10 ** best_params_MBBR['gamma']
best_params_MBBR['max_depth'] = int(round(2 ** best_params_MBBR['max_depth']))

# Define the remaining xgboost parameters
best_params_MBBR['objective'] = 'reg:squarederror'  # or 'binary:logistic' for classification
best_params_MBBR['eval_metric'] = 'rmse'  # or 'auc' for classification
best_params_MBBR['colsample_bytree'] = best_params_MBBR['col_subsample']
best_params_MBBR['subsample'] = best_params_MBBR['subsample']

del best_params_MBBR['col_subsample']
del best_params_MBBR['lambd']


best_params_MBBR
```

```
{'alpha': 7.087000588941134e-06,
 'eta': 1.0,
 'gamma': 0.008772055067787261,
 'max_depth': 2,
 'subsample': 0.5,
 'lambda': 0.0007115927569210631,
 'objective': 'reg:squarederror',
```

```
          'eval_metric': 'rmse',
          'colsample_bytree': 1.0}
```

## For Raw Dataset

```
# Convert the data into DMatrix format
dtrain = xgb.DMatrix(X_train_MBBR_dropped[list(optimal_features_MBBR_dropped)], label=y_train_MBBR_dropped)

# Define the function to be optimized
def xgb_evaluate(eta, alpha, lambd, gamma, subsample, col_subsample, max_depth):
    eta = 10**eta
    alpha = 10**alpha
    lambd = 10**lambd
    gamma = 10**gamma
    max_depth = int(round(2**max_depth))

    params = {'eval_metric': 'rmse',
              'objective': 'reg:squarederror',
              'max_depth': max_depth,
              'eta': eta,
              'gamma': gamma,
              'subsample': subsample,
              'alpha': alpha,
              'lambda': lambd,
              'colsample_bytree': col_subsample,}

    cv_result = xgb.cv(params, dtrain, num_boost_round=1000, nfold=5, early_stopping_rounds=30, seed=808)
    return -1.0 * cv_result['test-rmse-mean'].iloc[-1]

# Specify the hyperparameters to be tuned
xgb_bo_MBBR_dropped = BayesianOptimization(xgb_evaluate, {'eta': (-3, 0),
                                             'alpha': (-6, 0.3),
                                             'lambd': (-6, 0.3),
                                             'gamma': (-6, 1.8),
                                             'subsample': (0.5, 1),
                                             'col_subsample': (0.3, 1),
                                             'max_depth': (1, 3)},
                              random_state=808)

# Optimize the hyperparameters
xgb_bo_MBBR_dropped.maximize(n_iter=1000, init_points=10)# Convert the data into DMatrix format
```

```
|   iter    |  target   |   alpha   | col_su... |    eta    |   gamma   |   lambd   | max_depth | subsample |
-------------------------------------------------------------------------------------------------------------
| 1         | -7.694e+0 | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -7.673e+0 | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -7.71e+06 | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -7.682e+0 | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
| 5         | -7.401e+0 | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -7.694e+0 | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -7.687e+0 | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -7.251e+0 | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -7.696e+0 | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -7.455e+0 | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -6.93e+06 | -1.479    | 0.634     | -0.04167  | -0.3922   | -4.084    | 2.21      | 0.535     |
| 12        | -6.926e+0 | -1.608    | 0.6554    | 0.0       | -0.4209   | -4.208    | 2.847     | 0.5       |
| 13        | -7.05e+06 | -1.024    | 1.0       | 0.0       | -0.8683   | -4.933    | 2.526     | 0.5       |
| 14        | -7.05e+06 | -1.108    | 1.0       | 0.0       | 0.7515    | -4.525    | 2.851     | 0.5       |
| 15        | -7.049e+0 | 0.3       | 0.3       | 0.0       | 0.3571    | -6.0      | 3.0       | 0.5       |
| 16        | -7.049e+0 | 0.3       | 1.0       | 0.0       | 1.8       | -6.0      | 1.656     | 0.5       |
| 17        | -7.044e+0 | 0.3       | 1.0       | 0.0       | 1.8       | -6.0      | 3.0       | 1.0       |
| 18        | -1.152e+0 | -6.0      | 0.3       | 0.0       | -6.0      | -6.0      | 3.0       | 1.0       |
| 19        | -8.536e+0 | -6.0      | 0.3       | 0.0       | 1.8       | 0.3       | 3.0       | 1.0       |
| 20        | -7.691e+0 | -4.345    | 1.0       | -3.0      | -6.0      | 0.3       | 1.0       | 1.0       |
| 21        | -7.697e+0 | 0.3       | 0.3       | -3.0      | -6.0      | -6.0      | 1.0       | 1.0       |
| 22        | -7.691e+0 | 0.3       | 1.0       | -3.0      | -6.0      | 0.3       | 1.0       | 1.0       |
| 23        | -7.042e+0 | 0.3       | 0.3       | 0.0       | 1.8       | 0.3       | 1.0       | 0.5       |
| 24        | -7.691e+0 | 0.3       | 1.0       | -3.0      | 1.8       | 0.3       | 1.0       | 0.5       |
| 25        | -7.045e+0 | 0.3       | 1.0       | 0.0       | -6.0      | -3.169    | 1.0       | 0.5       |
| 26        | -7.045e+0 | 0.3       | 0.3       | 0.0       | 1.8       | -2.701    | 1.0       | 0.5       |
| 27        | -7.254e+0 | 0.1712    | 0.3294    | -0.02565  | -3.823    | -3.298    | 1.115     | 0.7572    |
| 28        | -7.042e+0 | -2.457    | 1.0       | 0.0       | -1.493    | 0.3       | 1.0       | 0.5       |
| 29        | -7.05e+06 | 0.3       | 1.0       | 0.0       | -6.0      | -5.048    | 3.0       | 0.5       |
| 30        | -7.693e+0 | 0.3       | 1.0       | -3.0      | -6.0      | -3.287    | 3.0       | 0.5       |
| 31        | -1.186e+0 | -2.242    | 0.3       | 0.0       | 0.2528    | -1.682    | 1.0       | 1.0       |
| 32        | -7.042e+0 | -2.843    | 1.0       | 0.0       | -3.558    | 0.3       | 1.0       | 0.5       |
| 33        | -7.045e+0 | 0.3       | 1.0       | 0.0       | -4.614    | -5.838    | 1.0       | 0.5       |
| 34        | -7.049e+0 | 0.3       | 0.3       | 0.0       | -3.379    | -5.408    | 3.0       | 0.5       |
```

```
| 35        | -7.696e+0 | 0.3       | 0.3       | -2.336    | -1.61     | -6.0      | 3.0       | 1.0       |
| 36        | -7.05e+06 | -1.424    | 1.0       | 0.0       | -4.386    | -4.42     | 2.4       | 0.5       |
| 37        | -7.042e+0 | -5.845    | 1.0       | 0.0       | -3.701    | 0.3       | 1.0       | 0.5       |
| 38        | -7.045e+0 | -1.191    | 0.3       | 0.0       | -6.0      | -5.015    | 1.0       | 0.5       |
| 39        | -7.042e+0 | -4.051    | 1.0       | 0.0       | -2.893    | 0.3       | 3.0       | 0.5       |
| 40        | -7.042e+0 | -5.607    | 0.3       | 0.0       | -5.41     | 0.3       | 3.0       | 0.5       |
| 41        | -7.048e+0 | -6.0      | 1.0       | -1.942    | -3.776    | 0.3       | 3.0       | 1.0       |
| 42        | -7.693e+0 | 0.3       | 1.0       | -2.908    | 1.8       | -6.0      | 3.0       | 0.5       |
| 43        | -7.04e+06 | -6.0      | 1.0       | 0.0       | -3.619    | -1.196    | 3.0       | 1.0       |
| 44        | -7.693e+0 | -6.0      | 1.0       | -3.0      | 1.8       | -6.0      | 3.0       | 0.5       |
| 45        | -1.195e+0 | -1.474    | 0.3       | 0.0       | -3.018    | -6.0      | 1.0       | 1.0       |
| 46        | -7.05e+06 | 0.3       | 1.0       | 0.0       | -4.38     | -3.575    | 3.0       | 0.5       |
| 47        | -7.05e+06 | -1.44     | 1.0       | 0.0       | -6.0      | -3.292    | 2.628     | 0.5       |
| 48        | -7.04e+06 | -4.222    | 1.0       | 0.0       | -4.914    | -1.106    | 1.944     | 0.5       |
| 49        | -8.936e+0 | -6.0      | 1.0       | 0.0       | -6.0      | -0.4586   | 1.0       | 1.0       |
| 50        | -7.569e+0 | -4.084    | 0.3       | -1.299    | -4.48     | 0.3       | 3.0       | 0.5       |
| 51        | -7.042e+0 | -6.0      | 1.0       | 0.0       | -2.182    | 0.3       | 2.487     | 0.5       |
| 52        | -7.073e+0 | -2.607    | 0.8347    | -0.04126  | -4.009    | -2.697    | 2.738     | 0.7911    |
| 53        | -7.05e+06 | 0.3       | 1.0       | 0.0       | -2.04     | -3.484    | 3.0       | 0.5       |
| 54        | -7.049e+0 | 0.3       | 0.3       | 0.0       | 1.8       | -3.669    | 3.0       | 0.5       |
| 55        | -7.643e+0 | -6.0      | 1.0       | -3.0      | -1.389    | 0.3       | 3.0       | 0.5       |
| 56        | -7.042e+0 | 0.3       | 1.0       | 0.0       | 1.8       | 0.3       | 3.0       | 0.5       |
```

```
# Extract the optimal hyperparameters from the Bayesian Optimization object
best_params_MBBR_dropped = xgb_bo_MBBR_dropped.max['params']

# Transform the hyperparameters from log space to original space
best_params_MBBR_dropped['eta'] = 10 ** best_params_MBBR_dropped['eta']
best_params_MBBR_dropped['alpha'] = 10 ** best_params_MBBR_dropped['alpha']
best_params_MBBR_dropped['lambda'] = 10 ** best_params_MBBR_dropped['lambd']
best_params_MBBR_dropped['gamma'] = 10 ** best_params_MBBR_dropped['gamma']
best_params_MBBR_dropped['max_depth'] = int(round(2 ** best_params_MBBR_dropped['max_depth']))

# Define the remaining xgboost parameters
best_params_MBBR_dropped['objective'] = 'reg:squarederror'  # or 'binary:logistic' for classification
best_params_MBBR_dropped['eval_metric'] = 'rmse'  # or 'auc' for classification
best_params_MBBR_dropped['colsample_bytree'] = best_params_MBBR_dropped['col_subsample']
best_params_MBBR_dropped['subsample'] = best_params_MBBR_dropped['subsample']

del best_params_MBBR_dropped['col_subsample']
del best_params_MBBR_dropped['lambd']
```

```
best_params_MBBR_dropped
```

```
{'alpha': 1.9571699265928916e-05,
 'eta': 1.0,
 'gamma': 6.658026860110587e-06,
 'max_depth': 8,
 'subsample': 0.5,
 'lambda': 0.2239738935239124,
 'objective': 'reg:squarederror',
 'eval_metric': 'rmse',
 'colsample_bytree': 0.7933101665088923}
```

## Final Model Training and Testing

### Optimized XGBoost 1

- Optimal Features
- Optimal Hyperparameters
- Trained on Imputed Dataset

```
# Convert test data to DMatrix format
dtrain = xgb.DMatrix(X_train_MBBR[list(optimal_features_MBBR)], label=y_train_MBBR)
dtest = xgb.DMatrix(X_test_MBBR[list(optimal_features_MBBR)], label=y_test_MBBR)
```

### Determination of optimal num_boost_round

```
evals_result_MBBR = {}

# Train the final model
```

```
final_model_MBBR = xgb.train(best_params_MBBR, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(dtrain, 'train'),
                  evals_result=evals_result_MBBR)
```

```
[0]	train-rmse:15326757.31881	test-rmse:20556440.54062
[1]	train-rmse:13158088.40357	test-rmse:11652636.19547
[2]	train-rmse:13112056.73454	test-rmse:8251574.30139
[3]	train-rmse:10208890.52535	test-rmse:11777054.61839
[4]	train-rmse:10533833.63598	test-rmse:10208212.97752
[5]	train-rmse:11666431.06081	test-rmse:17356592.63384
[6]	train-rmse:6523781.28018	test-rmse:16784534.79832
[7]	train-rmse:4641129.28839	test-rmse:13006880.57134
[8]	train-rmse:4615933.02585	test-rmse:14433315.05283
[9]	train-rmse:4188194.28595	test-rmse:12356161.10686
[10]	train-rmse:3980675.44196	test-rmse:16524990.57339
[11]	train-rmse:3876631.52641	test-rmse:11828306.22478
[12]	train-rmse:3850751.62850	test-rmse:11639219.10463
[13]	train-rmse:3111913.35200	test-rmse:12480128.69677
[14]	train-rmse:3144384.01503	test-rmse:14000071.55297
[15]	train-rmse:4490402.78577	test-rmse:11273455.75827
[16]	train-rmse:3135431.76279	test-rmse:14219482.91843
[17]	train-rmse:2916114.69919	test-rmse:13148038.43770
[18]	train-rmse:2963560.25627	test-rmse:14125190.31490
[19]	train-rmse:3987981.28920	test-rmse:10953098.02412
[20]	train-rmse:4144037.82337	test-rmse:17228416.63439
[21]	train-rmse:4858238.92076	test-rmse:12441047.04393
[22]	train-rmse:4034396.11340	test-rmse:15214692.33078
[23]	train-rmse:3990673.28328	test-rmse:12317691.97788
[24]	train-rmse:4281055.38655	test-rmse:15146669.78916
[25]	train-rmse:4395766.82648	test-rmse:15384318.84493
[26]	train-rmse:3055663.05621	test-rmse:12935065.49016
[27]	train-rmse:3948434.80413	test-rmse:12842799.75838
[28]	train-rmse:3207340.55277	test-rmse:11654321.05840
[29]	train-rmse:3043039.69794	test-rmse:11009765.91870
[30]	train-rmse:3188854.30697	test-rmse:10128036.38984
[31]	train-rmse:4662322.25246	test-rmse:10585836.41850
[32]	train-rmse:2979828.25900	test-rmse:10872295.01321
```

```
# Train the final model
final_model_MBBR = xgb.train(best_params_MBBR, dtrain, num_boost_round=(np.argmin(evals_result_MBBR['train']['rmse'])+1), early_
                  evals_result=evals_result_MBBR)

# Make predictions on the test set
y_pred_final_MBBR = final_model_MBBR.predict(dtest)
```

```
[0]	train-rmse:15326757.31881	test-rmse:20556440.54062
[1]	train-rmse:13158088.40357	test-rmse:11652636.19547
[2]	train-rmse:13112056.73454	test-rmse:8251574.30139
[3]	train-rmse:10208890.52535	test-rmse:11777054.61839
[4]	train-rmse:10533833.63598	test-rmse:10208212.97752
[5]	train-rmse:11666431.06081	test-rmse:17356592.63384
[6]	train-rmse:6523781.28018	test-rmse:16784534.79832
[7]	train-rmse:4641129.28839	test-rmse:13006880.57134
[8]	train-rmse:4615933.02585	test-rmse:14433315.05283
[9]	train-rmse:4188194.28595	test-rmse:12356161.10686
[10]	train-rmse:3980675.44196	test-rmse:16524990.57339
[11]	train-rmse:3876631.52641	test-rmse:11828306.22478
[12]	train-rmse:3850751.62850	test-rmse:11639219.10463
[13]	train-rmse:3111913.35200	test-rmse:12480128.69677
[14]	train-rmse:3144384.01503	test-rmse:14000071.55297
[15]	train-rmse:4490402.78577	test-rmse:11273455.75827
[16]	train-rmse:3135431.76279	test-rmse:14219482.91843
[17]	train-rmse:2916114.69919	test-rmse:13148038.43770
```

## Optimized XGBoost 2

- Optimal Features
- Optimal Hyperparameters
- Trained on Raw Dataset

```
# Convert test data to DMatrix format
dtrain = xgb.DMatrix(X_train_MBBR_dropped[list(optimal_features_MBBR_dropped)], label=y_train_MBBR_dropped)
dtest = xgb.DMatrix(X_test_MBBR_dropped[list(optimal_features_MBBR_dropped)], label=y_test_MBBR_dropped)
```

## Determination of optimal num_boost_round

```
evals_result_MBBR_dropped = {}

# Train the final model
final_model_MBBR_dropped = xgb.train(best_params_MBBR_dropped, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(c
                 evals_result=evals_result_MBBR_dropped)
```

```
[0]	train-rmse:5374639.73841	test-rmse:10710738.33162
[1]	train-rmse:5261032.81372	test-rmse:10794379.47035
[2]	train-rmse:5261062.24118	test-rmse:10794402.13798
[3]	train-rmse:5259431.79352	test-rmse:10816562.41102
[4]	train-rmse:5248956.95379	test-rmse:10821987.03532
[5]	train-rmse:9131312.98946	test-rmse:10810542.96985
[6]	train-rmse:6360658.34510	test-rmse:13105262.19796
[7]	train-rmse:4784346.41598	test-rmse:13105906.82526
[8]	train-rmse:3215751.26893	test-rmse:13284376.87235
[9]	train-rmse:3613229.24059	test-rmse:15517334.99757
[10]	train-rmse:3783223.21007	test-rmse:15509583.11839
[11]	train-rmse:3959346.29623	test-rmse:15517073.78101
[12]	train-rmse:4671634.61922	test-rmse:15704051.78270
[13]	train-rmse:3562085.61010	test-rmse:15473097.64522
[14]	train-rmse:2722604.92831	test-rmse:16084128.45143
[15]	train-rmse:2662003.35424	test-rmse:16089807.50350
[16]	train-rmse:1396620.29196	test-rmse:16235586.99514
[17]	train-rmse:619017.73207	test-rmse:16097032.76861
[18]	train-rmse:545703.28252	test-rmse:16175620.26666
[19]	train-rmse:495044.70435	test-rmse:16187470.35250
[20]	train-rmse:463611.74499	test-rmse:16222250.56298
[21]	train-rmse:399261.63186	test-rmse:16268002.51719
[22]	train-rmse:372574.76568	test-rmse:16221093.65466
[23]	train-rmse:231409.25297	test-rmse:16188447.36982
[24]	train-rmse:213484.05147	test-rmse:16185089.24820
[25]	train-rmse:200677.69898	test-rmse:16081437.82314
[26]	train-rmse:187250.42981	test-rmse:16082133.53278
[27]	train-rmse:167653.00241	test-rmse:16093221.89212
[28]	train-rmse:136968.28216	test-rmse:16107159.97033
[29]	train-rmse:104738.40030	test-rmse:16087805.70848
[30]	train-rmse:90127.15404	test-rmse:16087468.20215
```

```
# Train the final model
final_model_MBBR_dropped = xgb.train(best_params_MBBR_dropped, dtrain, num_boost_round=(np.argmin(evals_result_MBBR_dropped['tra
                 evals_result=evals_result_MBBR_dropped)

# Make predictions on the test set
y_pred_final_MBBR_dropped = final_model_MBBR_dropped.predict(dtest)
```

```
[0]	train-rmse:5374639.73841	test-rmse:10710738.33162
[1]	train-rmse:5261032.81372	test-rmse:10794379.47035
[2]	train-rmse:5261062.24118	test-rmse:10794402.13798
[3]	train-rmse:5259431.79352	test-rmse:10816562.41102
[4]	train-rmse:5248956.95379	test-rmse:10821987.03532
[5]	train-rmse:9131312.98946	test-rmse:10810542.96985
[6]	train-rmse:6360658.34510	test-rmse:13105262.19796
[7]	train-rmse:4784346.41598	test-rmse:13105906.82526
[8]	train-rmse:3215751.26893	test-rmse:13284376.87235
[9]	train-rmse:3613229.24059	test-rmse:15517334.99757
[10]	train-rmse:3783223.21007	test-rmse:15509583.11839
[11]	train-rmse:3959346.29623	test-rmse:15517073.78101
[12]	train-rmse:4671634.61922	test-rmse:15704051.78270
[13]	train-rmse:3562085.61010	test-rmse:15473097.64522
[14]	train-rmse:2722604.92831	test-rmse:16084128.45143
[15]	train-rmse:2662003.35424	test-rmse:16089807.50350
[16]	train-rmse:1396620.29196	test-rmse:16235586.99514
[17]	train-rmse:619017.73207	test-rmse:16097032.76861
[18]	train-rmse:545703.28252	test-rmse:16175620.26666
[19]	train-rmse:495044.70435	test-rmse:16187470.35250
[20]	train-rmse:463611.74499	test-rmse:16222250.56298
[21]	train-rmse:399261.63186	test-rmse:16268002.51719
[22]	train-rmse:372574.76568	test-rmse:16221093.65466
[23]	train-rmse:231409.25297	test-rmse:16188447.36982
[24]	train-rmse:213484.05147	test-rmse:16185089.24820
[25]	train-rmse:200677.69898	test-rmse:16081437.82314
[26]	train-rmse:187250.42981	test-rmse:16082133.53278
[27]	train-rmse:167653.00241	test-rmse:16093221.89212
[28]	train-rmse:136968.28216	test-rmse:16107159.97033
[29]	train-rmse:104738.40030	test-rmse:16087805.70848
[30]	train-rmse:90127.15404	test-rmse:16087468.20215
```

## Untuned XGBoost 1

- No Feature Selection

- No Hyperparameter Tuning
- Trained on **Imputed Dataset**

```
dtrain = xgb.DMatrix(X_train_MBBR, label=y_train_MBBR)
dtest = xgb.DMatrix(X_test_MBBR, label=y_test_MBBR)


params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'seed': 808
}

# Train the out of the box xgboost model
oob_model_imputed_MBBR = xgb.train(params, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(dtrain, 'train'),(dte

# Make predictions on the test set
y_pred_oob_imputed_MBBR = oob_model_imputed_MBBR.predict(dtest)
```

```
[0]	train-rmse:9479143.35908	test-rmse:1642540.75899
[1]	train-rmse:8050074.35392	test-rmse:2683902.79716
[2]	train-rmse:6838375.35646	test-rmse:3638750.97395
[3]	train-rmse:5810190.48871	test-rmse:4466012.83917
[4]	train-rmse:4937243.65107	test-rmse:5173479.44755
[5]	train-rmse:4195842.01663	test-rmse:5776747.24762
[6]	train-rmse:3566002.98969	test-rmse:6290331.26426
[7]	train-rmse:3030833.15816	test-rmse:6727204.91587
[8]	train-rmse:2576052.70790	test-rmse:7098702.41227
[9]	train-rmse:2189559.53832	test-rmse:7414568.62093
[10]	train-rmse:1861070.31422	test-rmse:7683094.92386
[11]	train-rmse:1581887.61555	test-rmse:7911356.96262
[12]	train-rmse:1344594.44834	test-rmse:8105403.53968
[13]	train-rmse:1142903.90105	test-rmse:8270270.61016
[14]	train-rmse:971465.91060	test-rmse:8410336.04238
[15]	train-rmse:825750.20177	test-rmse:8529413.30762
[16]	train-rmse:701902.73426	test-rmse:8630719.19057
[17]	train-rmse:596627.84450	test-rmse:8716746.62247
[18]	train-rmse:507147.05936	test-rmse:8789911.93024
[19]	train-rmse:431095.35655	test-rmse:8852121.86062
[20]	train-rmse:366442.18501	test-rmse:8904955.26132
[21]	train-rmse:311494.44381	test-rmse:8949873.70577
[22]	train-rmse:264797.30939	test-rmse:8988052.23713
[23]	train-rmse:225103.70349	test-rmse:9020508.24990
[24]	train-rmse:191359.23743	test-rmse:9048122.98595
[25]	train-rmse:162678.36140	test-rmse:9071580.39981
[26]	train-rmse:138300.46214	test-rmse:9091511.02220
[27]	train-rmse:117584.91336	test-rmse:9108465.64005
[28]	train-rmse:99978.87545	test-rmse:9122860.35250
[29]	train-rmse:85019.20339	test-rmse:9135116.39976
[30]	train-rmse:72313.96033	test-rmse:9145522.15679
```

## Untuned XGBoost 2

- No Feature Selection
- No Hyperparameter Tuning
- Trained on **Non-Imputed (Raw) Dataset**

```
dtrain = xgb.DMatrix(X_train_MBBR_dropped, label=y_train_MBBR_dropped)
dtest = xgb.DMatrix(X_test_MBBR_dropped, label=y_test_MBBR_dropped)

params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'seed': 808
}

# Train the out of the box xgboost model
oob_model_MBBR = xgb.train(params, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(dtrain, 'train'),(dtest, 'tes

# Make predictions on the test set
y_pred_oob_MBBR = oob_model_MBBR.predict(dtest)
```

```
[0]	train-rmse:9519815.53422	test-rmse:1652771.59713
[1]	train-rmse:8084553.07502	test-rmse:2698503.36215
[2]	train-rmse:6867627.99533	test-rmse:3657831.42325
[3]	train-rmse:5835023.21290	test-rmse:4489087.68919
```

```
[4]	train-rmse:4958332.22973	test-rmse:5200008.49322
[5]	train-rmse:4213756.41561	test-rmse:5806236.68403
[6]	train-rmse:3581201.64223	test-rmse:6321627.05176
[7]	train-rmse:3043752.33745	test-rmse:6760666.50286
[8]	train-rmse:2587024.99466	test-rmse:7133632.34705
[9]	train-rmse:2198879.98559	test-rmse:7450760.44869
[10]	train-rmse:1869006.02574	test-rmse:7720335.04068
[11]	train-rmse:1588630.46331	test-rmse:7949746.57804
[12]	train-rmse:1350326.42363	test-rmse:8144779.17268
[13]	train-rmse:1147778.96603	test-rmse:8310555.68754
[14]	train-rmse:975619.46369	test-rmse:8451463.72314
[15]	train-rmse:829277.73479	test-rmse:8571184.24704
[16]	train-rmse:704894.71358	test-rmse:8673000.13791
[17]	train-rmse:599171.65153	test-rmse:8759475.10567
[18]	train-rmse:509305.29322	test-rmse:8833007.37989
[19]	train-rmse:432928.31771	test-rmse:8895487.84073
[20]	train-rmse:367999.13234	test-rmse:8948650.41734
[21]	train-rmse:312817.13270	test-rmse:8993838.68937
[22]	train-rmse:265906.07836	test-rmse:9032212.64033
[23]	train-rmse:226025.09356	test-rmse:9064827.42938
[24]	train-rmse:192130.67907	test-rmse:9092542.38062
[25]	train-rmse:163331.56076	test-rmse:9116108.44238
[26]	train-rmse:138848.48541	test-rmse:9136163.11216
[27]	train-rmse:118041.93585	test-rmse:9153187.27123
[28]	train-rmse:100358.68146	test-rmse:9167679.43571
[29]	train-rmse:85321.99228	test-rmse:9179979.12628
[30]	train-rmse:72538.19016	test-rmse:9190435.86780
```

## Naive Model 1

- **Always predicts** the mean effluent chlorine residual of the **imputed training dataset**

```
y_pred_naive_MBBR = np.full(y_test_MBBR.shape, y_train_MBBR.mean())
```

## Naive Model 2

- **Always predicts** the mean effluent chlorine residual of the **Non-imputed (raw) training dataset**

```
y_pred_naive_orig_MBBR = np.full(y_test_MBBR.shape, y_train_orig_MBBR.mean())
```

## Model Evaluation

```
def compute_metrics(y_pred,y_test):
  std_obs = np.std(y_test)
  std_sim = np.std(y_pred)

  mean_obs = np.mean(y_test)
  mean_sim = np.mean(y_pred)

  # Computing correlation
  r = np.corrcoef(y_test, y_pred)[0, 1]

  # Computing KGE
  alpha = std_sim / std_obs
  beta = mean_sim / mean_obs

  kge = 1 - np.sqrt(np.square(r - 1) + np.square(alpha - 1) + np.square(beta - 1))

  # PBIAS Calculation
  pbias = np.sum((y_test - y_pred)) / np.sum(y_test) * 100

  # Computing NSE
  nse = 1 - (np.sum((y_test-y_pred)**2))/(np.sum((y_test-np.mean(y_test))**2))

  if nse > 0.35:
    nse = (nse,'good')
  else:
    nse = (nse,'bad')
  if abs(pbias) < 15:
    pbias = (abs(pbias),'good')
  else:
    pbias = (abs(pbias),'bad')
```

```
  if kge > -0.41:
    kge = (kge,'good')
  else:
    kge = (kge,'bad')

  return(nse,pbias,kge)


def compute_nrmse(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    nrmse = rmse / (np.max(y_true) - np.min(y_true))
    return nrmse


non_imputed_mask_MBBR = ~np.isnan(y_test_orig_MBBR)
```

## Model Metrics evaluated on Imputed Test Set

### Optimized XGBoost 1

```
nse_final, pbias_final, kge_final = compute_metrics(y_pred_final_MBBR, y_test_MBBR)
print(f"Final model metrics:\n\nNSE: {nse_final}, \nPBIAS: {pbias_final}, \nKGE: {kge_final}")

rmse = mean_squared_error(y_test_MBBR, y_pred_final_MBBR, squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR, y_pred_final_MBBR)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-124395.34883840327, 'bad'),
PBIAS: (18521.398907237868, 'bad'),
KGE: (-394.02292984788403, 'bad')

Root Mean Squared Error: 13148038.353033414
Normalized Root Mean Squared Error: 54.33971876770298
```

### Untuned XGBoost 1

```
nse_naive, pbias_naive, kge_naive = compute_metrics(y_pred_oob_imputed_MBBR, y_test_MBBR)
print(f"Final model metrics:\n\nNSE: {nse_naive}, \nPBIAS: {pbias_naive}, \nKGE: {kge_naive}")

rmse = mean_squared_error(y_test_MBBR, y_pred_oob_imputed_MBBR, squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR, y_pred_oob_imputed_MBBR)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-60185.99892512592, 'bad'),
PBIAS: (10942.715794715634, 'bad'),
KGE: (-265.4704090498381, 'bad')

Root Mean Squared Error: 9145521.812809095
Normalized Root Mean Squared Error: 37.797659996731255
```

### Naive Model 1

```
rmse = mean_squared_error(y_test_MBBR, y_pred_naive_MBBR, squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR, y_pred_naive_MBBR)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 1028079.5096218928
Normalized Root Mean Squared Error: 4.248964744676363
```

### Naive Model 2

```
rmse = mean_squared_error(y_test_MBBR, y_pred_naive_orig_MBBR, squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR, y_pred_naive_orig_MBBR)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 1036980.085520665
Normalized Root Mean Squared Error: 4.285750064145582
```

## Model Metrics evaluated on Non-Imputed (Raw) Test Set

### Optimized XGBoost 1

```
nse_final, pbias_final, kge_final = compute_metrics(y_pred_final_MBBR[non_imputed_mask_MBBR], y_test_MBBR_dropped)
print(f"Final model metrics:\n\nNSE: {nse_final}, \nPBIAS: {pbias_final}, \nKGE: {kge_final}")

rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_final_MBBR[non_imputed_mask_MBBR],squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR_dropped, y_pred_final_MBBR[non_imputed_mask_MBBR])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-124482.86052319643, 'bad'),
PBIAS: (18904.129253841766, 'bad'),
KGE: (-395.9192472098591, 'bad')

Root Mean Squared Error: 13214269.26435048
Normalized Root Mean Squared Error: 54.61344546350835
```

### Optimized XGBoost 2

```
nse_final, pbias_final, kge_final = compute_metrics(y_pred_final_MBBR_dropped, y_test_MBBR_dropped)
print(f"Final model metrics:\n\nNSE: {nse_final}, \nPBIAS: {pbias_final}, \nKGE: {kge_final}")

rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_final_MBBR_dropped,squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR_dropped, y_pred_final_MBBR_dropped)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-184501.454805468, 'bad'),
PBIAS: (55579.13913757668, 'bad'),
KGE: (-688.5362345486618, 'bad')

Root Mean Squared Error: 16087468.23350308
Normalized Root Mean Squared Error: 66.4881312345143
```

### Untuned XGBoost 2

```
nse_naive, pbias_naive, kge_naive = compute_metrics(y_pred_oob_MBBR, y_test_MBBR_dropped)
print(f"Final model metrics:\n\nNSE: {nse_naive}, \nPBIAS: {pbias_naive}, \nKGE: {kge_naive}")

rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_oob_MBBR, squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR_dropped, y_pred_oob_MBBR)
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-60213.20914385645, 'bad'),
PBIAS: (11142.354281017806, 'bad'),
KGE: (-266.3424514166646, 'bad')

Root Mean Squared Error: 9190436.131325297
Normalized Root Mean Squared Error: 37.98328703639154
```

## Naive Model 1

```
rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_naive_MBBR[non_imputed_mask_MBBR],squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR_dropped, y_pred_naive_MBBR[non_imputed_mask_MBBR])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 1028185.1972427529
Normalized Root Mean Squared Error: 4.249401542580397
```

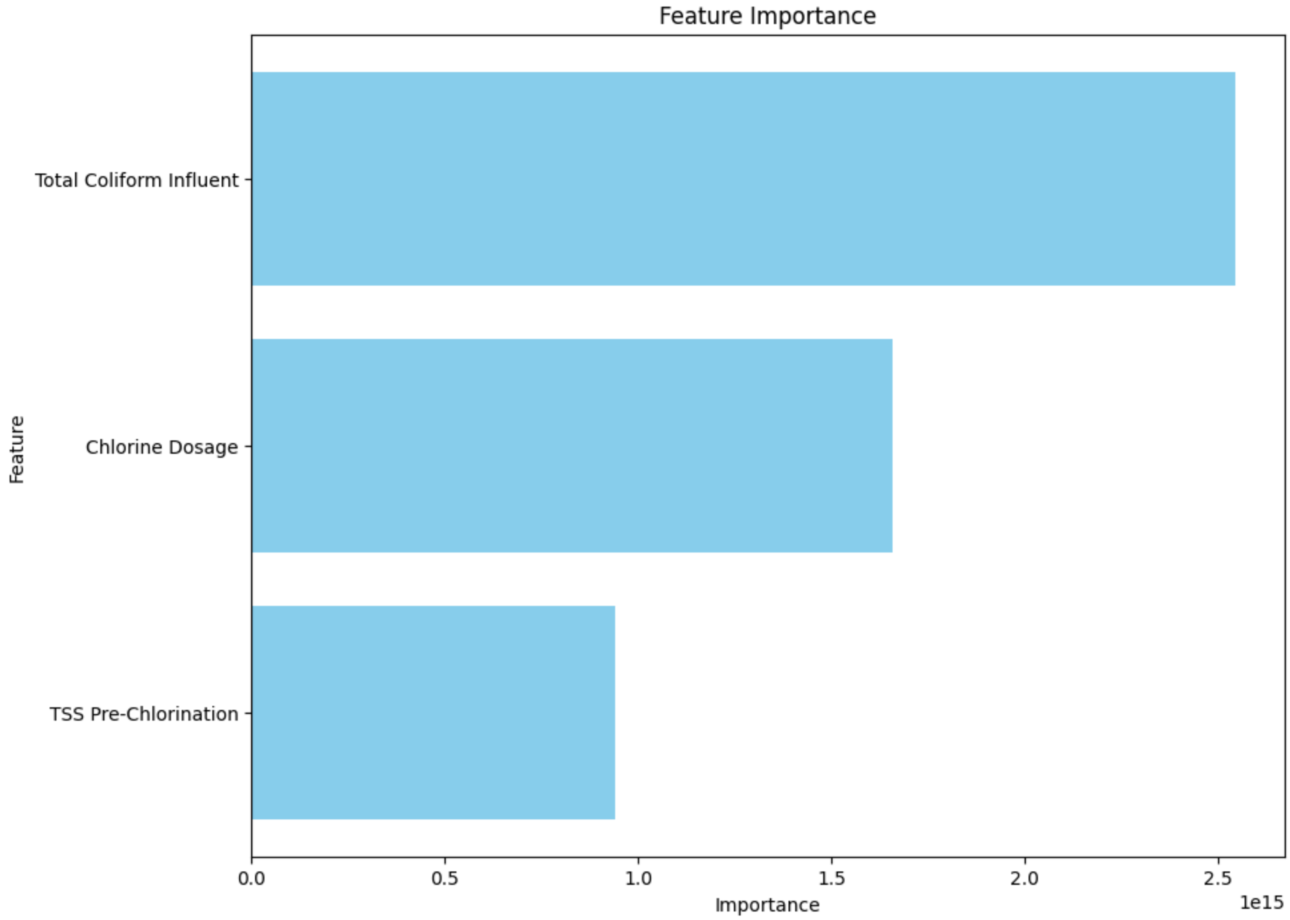
## Naive Model 2

```
rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_naive_orig_MBBR[non_imputed_mask_MBBR],squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_MBBR_dropped, y_pred_naive_orig_MBBR[non_imputed_mask_MBBR])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 1037085.7197988323
Normalized Root Mean Squared Error: 4.286186641588826
```

## Feature Importance

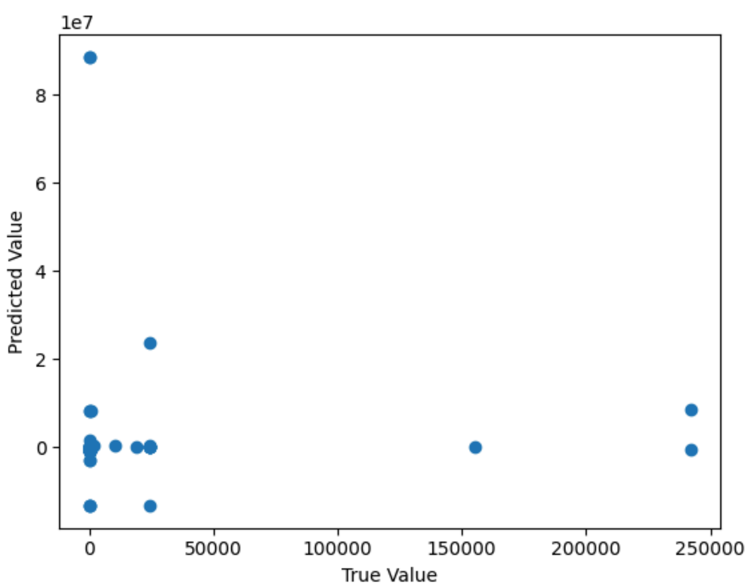
```
# Get feature importance
importance_MBBR = final_model_MBBR.get_score(importance_type='gain')

name_dict_MBBR = {
    'Flow.Rate.Influent..m3.d.': 'Flow Rate Influent',
    'BOD.Influent..ppm.': 'BOD Influent',
    'Total.Coliform.Effluent..MPN.100mL.':'Total Coliform Effluent',
    'pH.Pre.chlorination': 'pH Pre-Chlorination',
    'Chlorine.dosage..L.d.':'Chlorine Dosage',
    'TSS.Pre.chlorination..ppm.':'TSS Pre-Chlorination',
    'Total.Coliform.Influent..MPN.100mL.': 'Total Coliform Influent',
    'Fecal.Coliform.Influent..MPN.100mL.':'Fecal Coliform Influent',
    'BOD.Pre.chlorination..ppm.':'BOD Pre-Chlorination',
    'Fecal.Coliform.Effluent..MPN.100mL.':'Fecal Coliform Effluent',
    'COD.Influent..ppm.':'COD Influent',
    'COD.Pre.chlorination..ppm.':'COD Pre-Chlorination',
    }

# For visualization, it is better to convert it to a DataFrame
importance_df_MBBR = pd.DataFrame({
    'Feature': list(importance_MBBR.keys()),
    'Importance': list(importance_MBBR.values())
})

importance_df_MBBR['Feature'] = importance_df_MBBR['Feature'].replace(name_dict_MBBR)

# Sort the DataFrame by importance
importance_df_MBBR = importance_df_MBBR.sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 8))
plt.barh(importance_df_MBBR['Feature'], importance_df_MBBR['Importance'], color='skyblue')
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importance")
plt.gca().invert_yaxis()  # To show the highest importance at the top
plt.show()
```



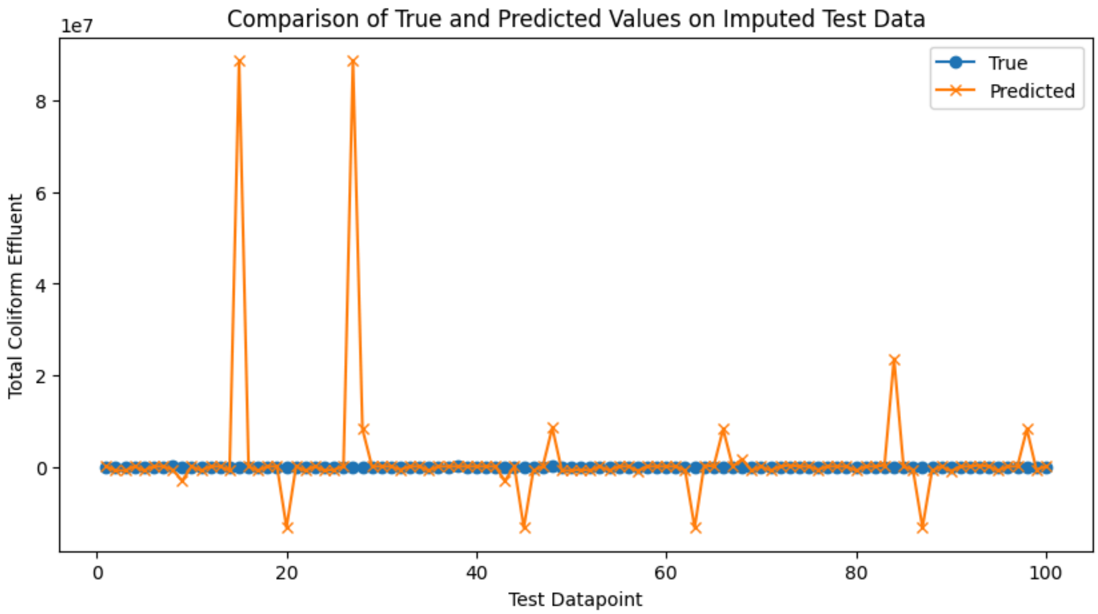

## Data Visualization for Model Evaluation

## Optimized XGBoost on Imputed Test Dataset

```
# with imputation
plt.scatter(y_test_MBBR,y_pred_final_MBBR);
plt.xlabel('True Value');
plt.ylabel('Predicted Value');
```

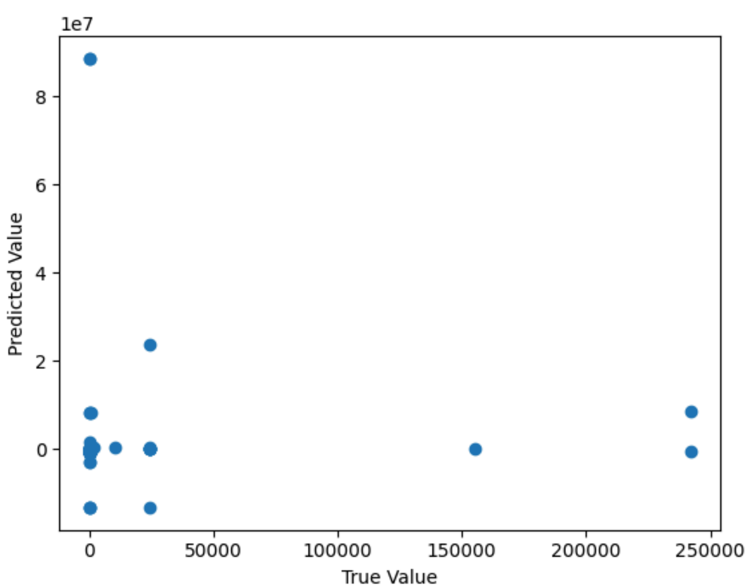


```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_MBBR) + 1)

# Plotting
plt.figure(figsize=(10, 5))
plt.plot(x, y_test_MBBR, label='True', marker='o')
plt.plot(x, y_pred_final_MBBR, label='Predicted', marker='x')

# Adding labels and title
plt.xlabel('Test Datapoint')
plt.ylabel('Total Coliform Effluent')
plt.title('Comparison of True and Predicted Values on Imputed Test Data')
plt.legend()

# Show plot
plt.show()
```

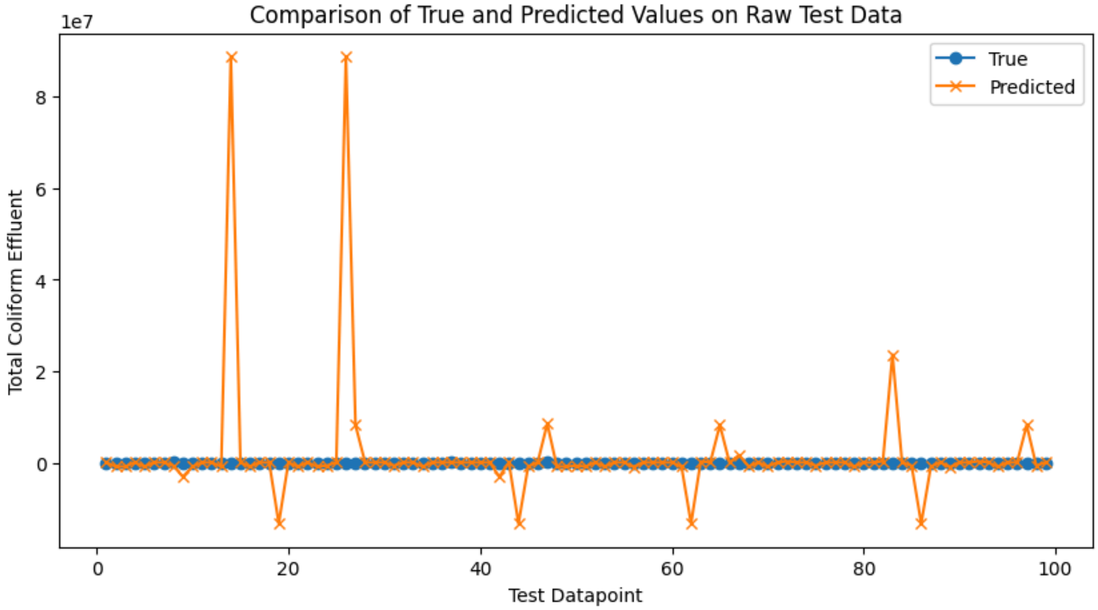



## Optimized XGBoost on Non-Imputed (Raw) Test Dataset

```
# without imputation
plt.scatter(y_test_orig_MBBR[non_imputed_mask_MBBR],y_pred_final_MBBR[non_imputed_mask_MBBR])
plt.xlabel('True Value');
plt.ylabel('Predicted Value');
```



```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_orig_MBBR[non_imputed_mask_MBBR]) + 1)

# Plotting
plt.figure(figsize=(10, 5))
plt.plot(x, y_test_orig_MBBR[non_imputed_mask_MBBR], label='True', marker='o')
plt.plot(x, y_pred_final_MBBR[non_imputed_mask_MBBR], label='Predicted', marker='x')

# Adding labels and title
plt.xlabel('Test Datapoint')
plt.ylabel('Total Coliform Effluent')
plt.title('Comparison of True and Predicted Values on Raw Test Data')
plt.legend()

# Show plot
plt.show()
```




## Exporting Results

```
# Determine the maximum length of the columns
max_length = max(len(y_test_MBBR), len(y_test_MBBR_dropped), len(y_pred_final_MBBR), len(y_pred_final_MBBR_dropped), len(y_pred_

# Function to extend a series or array to the maximum length with NaN values
def extend_with_nan(data, length):
    if isinstance(data, np.ndarray):
        data = pd.Series(data)
    return data.reindex(range(length), fill_value=np.nan)


# Extend all columns to the maximum length
y_test_MBBR = extend_with_nan(y_test_MBBR, max_length)
y_test_MBBR_dropped = extend_with_nan(y_test_MBBR_dropped.reset_index(drop='True'), max_length)
y_pred_final_MBBR = extend_with_nan(y_pred_final_MBBR, max_length)
y_pred_final_MBBR_dropped = extend_with_nan(y_pred_final_MBBR_dropped, max_length)
```