```
!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 | 1.80 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='Residual chlorine\n(ppm)')
y_orig_MBBR = MBBR['Residual chlorine\n(ppm)']
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	11,14
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	3,500,000,00
Total.Coliform.Effluent..MPN.100mL.	100.00	8,933.07	37,466.19	0.00	2.00	10.00	134.75	241,96
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	3,000,000,00
Fecal.Coliform.Effluent..MPN.100mL.	100.00	2,363.05	6,219.15	2.00	10.00	10.00	107.30	24,19
BOD.Influent..ppm.	100.00	134.13	88.52	19.00	66.45	108.40	188.25	40
BOD.Pre.chlorination..ppm.	100.00	12.06	13.39	1.00	5.00	9.00	15.00	10
COD.Influent..ppm.	100.00	268.66	192.32	41.00	135.75	210.00	357.25	1,25
COD.Pre.chlorination..ppm.	100.00	45.03	47.09	5.00	20.00	29.70	53.55	31
TSS.Pre.chlorination..ppm.	100.00	18.00	24.39	1.00	5.00	10.00	19.25	16
pH.Pre.chlorination	100.00	7.21	0.37	5.28	7.10	7.21	7.40	
Chlorine.dosage..L.d.	100.00	809.32	498.65	0.00	488.70	748.90	1,005.05	2,90
Residual.chlorine..ppm.	100.00	2.14	1.69	0.01	0.49	2.05	3.57	

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['Residual.chlorine..ppm.']
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['Residual.chlorine..ppm.']

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
474	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.386445	1.611471	0.028575	0.143105
350	(Chlorine.dosage..L.d., Total.Coliform.Influen...	0.410715	1.622629	0.044093	0.123522
409	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.362284	1.631725	0.047728	0.139558
410	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.382969	1.633061	0.036756	0.105161
278	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.374995	1.635504	0.032118	0.089523
...	...	...	...	...	...
106	(Chlorine.dosage..L.d., Fecal.Coliform.Influen...	0.788887	2.085864	0.043836	0.179699
100	(Chlorine.dosage..L.d., Fecal.Coliform.Influen...	0.742502	2.099164	0.053870	0.108392
2	(Chlorine.dosage..L.d., Fecal.Coliform.Influen...	0.966307	2.111154	0.030007	0.200713
6	(Chlorine.dosage..L.d., COD.Pre.chlorination.....	1.070077	2.117816	0.101896	0.117162
27	(Chlorine.dosage..L.d., Fecal.Coliform.Influen...	0.857085	2.160103	0.053470	0.113003

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
474	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.386445	1.611471	0.028575	0.143105
350	(Chlorine.dosage..L.d., Total.Coliform.Influen...	0.410715	1.622629	0.044093	0.123522
409	(Chlorine.dosage..L.d., Flow.Rate.Influent..m3...	0.362284	1.631725	0.047728	0.139558

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

```
('Chlorine.dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'Fecal.Coliform.Influent..MPN.100mL.',
 'BOD.Influent..ppm.',
 'COD.Pre.chlorination..ppm.',
 'TSS.Pre.chlorination..ppm.',
 'pH.Pre.chlorination')
```

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

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

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

```
('Chlorine.dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'BOD.Influent..ppm.',
 'COD.Influent..ppm.',
 'TSS.Pre.chlorination..ppm.',
 'pH.Pre.chlorination')
```

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

```
('Chlorine.dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'Total.Coliform.Influent..MPN.100mL.',
 'Fecal.Coliform.Influent..MPN.100mL.',
 'BOD.Influent..ppm.',
 'COD.Pre.chlorination..ppm.',
 'TSS.Pre.chlorination..ppm.',
 'pH.Pre.chlorination')
```

```
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)',
       'Total Coliform Effluent (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'],
      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
31	(Chlorine dosage (L/d), Total Coliform Effluen...	1.022659	1.512094	0.031956	0.180507
2	(Chlorine dosage (L/d), Total Coliform Effluen...	1.031525	1.520116	0.038504	0.184722
32	(Chlorine dosage (L/d), Total Coliform Effluen...	0.636437	1.554748	0.050116	0.192632
160	(Chlorine dosage (L/d), Total Coliform Effluen...	0.612772	1.574227	0.047039	0.187918
574	(Chlorine dosage (L/d), Flow Rate Influent (m3...	0.424320	1.577279	0.044534	0.284987
...	...	...	...	...	...
1000	(Chlorine dosage (L/d), Fecal Coliform Influen...	0.420868	2.110952	0.038620	0.188357
139	(Chlorine dosage (L/d), Total Coliform Influen...	0.576095	2.111329	0.039815	0.206058
44	(Chlorine dosage (L/d), Fecal Coliform Influen...	0.804355	2.112326	0.033380	0.167510
525	(Chlorine dosage (L/d), Fecal Coliform Influen...	0.415596	2.112704	0.030191	0.144628
58	(Chlorine dosage (L/d), BOD Pre-chlorination\n...	0.648844	2.153476	0.061506	0.111363

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
31	(Chlorine dosage (L/d), Total Coliform Effluen...	1.022659	1.512094	0.031956	0.180507
2	(Chlorine dosage (L/d), Total Coliform Effluen...	1.031525	1.520116	0.038504	0.184722
32	(Chlorine dosage (L/d), Total Coliform Effluen...	0.636437	1.554748	0.050116	0.192632

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

```
('Chlorine dosage (L/d)',
 'Total Coliform Effluent (MPN/100mL)',
 'Fecal Coliform Effluent (MPN/100mL)')
```

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

```
('Chlorine dosage (L/d)', 'Total Coliform Effluent (MPN/100mL)')
```

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

```
('Chlorine dosage (L/d)',
 'Total Coliform Effluent (MPN/100mL)',
 'BOD Influent (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)',
 'Total Coliform Effluent (MPN/100mL)',
 'Fecal Coliform Effluent (MPN/100mL)')
```

```
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	-1.671	0.04075	0.4513	-2.68	-1.662	-1.582	2.026	0.7673
2	-1.623	-4.514	0.7529	-1.843	-2.2	-1.339	1.596	0.5436
3	-1.698	-1.108	0.5069	-1.136	-4.974	-0.5216	2.693	0.8202
4	-1.771	-3.147	0.6275	-2.063	1.604	-0.4504	1.466	0.7294
5	-1.713	-2.356	0.4052	-0.3806	-0.09483	-5.01	1.643	0.6674
6	-1.641	-2.162	0.5228	-2.659	-5.793	-3.144	2.227	0.7522
7	-1.639	-0.1605	0.6002	-1.973	0.8823	-3.597	1.193	0.6362
8	-1.753	-0.9486	0.7394	-0.4943	-0.4982	-3.564	2.166	0.5334
9	-1.63	-1.814	0.8098	-2.344	-2.07	-3.54	1.193	0.8742
10	-1.711	0.06321	0.6188	-0.4746	-0.88	-0.04974	2.478	0.7132
11	-1.603	-4.543	0.8022	-1.469	-2.3	-1.735	1.546	0.6257
12	-1.627	-3.644	0.8309	-1.947	-2.894	-2.698	1.318	0.7972
13	-1.574	-5.741	0.9132	-1.013	-3.386	-2.238	1.468	0.6447
14	-2.136	-6.0	0.3	0.0	-2.883	-2.26	3.0	1.0
15	-1.629	-5.073	1.0	-1.774	-3.319	-2.118	1.0	0.5056
16	-1.646	-6.0	1.0	-1.161	-4.403	-2.505	1.0	0.5
17	-1.642	-6.0	1.0	-1.186	-2.958	-1.212	1.0	0.5
18	-1.64	-5.877	1.0	-1.4	-2.537	-3.009	1.0	0.5
19	-1.736	-4.575	1.0	-0.3881	-3.405	-2.28	1.0	0.5
20	-1.632	-6.0	1.0	-2.145	-3.377	-2.195	1.728	1.0
21	-1.648	-4.42	1.0	-2.332	-1.536	-2.603	1.222	1.0
22	-1.657	-2.758	0.3	-2.073	-1.874	-1.948	1.33	0.5024
23	-1.678	-2.295	0.6771	-2.886	-3.659	-3.156	1.774	0.8583
24	-1.702	-4.588	0.3	-2.654	-2.756	-2.516	2.168	0.5
25	-1.647	-3.348	1.0	-2.132	-2.168	-4.241	1.0	1.0
26	-1.651	-3.92	1.0	-1.015	-1.276	-1.424	1.0	1.0
27	-1.739	-2.486	1.0	-3.0	-0.6419	-3.416	1.0	1.0
28	-1.643	-2.206	1.0	-1.118	-2.9	-3.281	1.0	1.0
29	-1.62	-1.667	1.0	-1.982	-3.038	-4.85	1.0	0.5
30	-1.663	-0.5516	0.3	-1.985	-3.064	-3.99	1.0	1.0
31	-1.601	-2.362	1.0	-1.616	-2.875	-4.533	2.355	0.5
32	-1.629	-2.764	1.0	-1.445	-3.912	-5.073	1.665	1.0
33	-1.625	-2.082	1.0	-2.575	-2.819	-5.585	2.3	1.0
34	-1.612	-1.253	1.0	-1.557	-3.846	-5.092	2.635	0.5
35	-1.728	-1.893	0.3	-1.0	-2.943	-5.815	2.107	0.5
36	-1.622	-2.381	1.0	-2.442	-4.07	-4.731	3.0	0.5
37	-1.645	-1.209	1.0	-2.381	-2.9	-4.242	3.0	0.9127
38	-1.679	-1.939	1.0	-1.058	-4.332	-3.825	2.582	0.5
39	-1.649	-1.425	1.0	-2.842	-4.374	-5.314	1.773	0.5
40	-1.625	-3.656	0.9717	-2.879	-2.655	-5.139	2.935	0.8396
41	-1.682	-3.277	1.0	-2.978	-3.507	-5.151	1.508	0.5
42	-1.716	-1.811	1.0	-1.807	-5.112	-5.898	3.0	1.0
43	-1.72	-3.027	1.0	-2.088	-2.407	-3.931	3.0	1.0
44	-1.613	-0.5171	1.0	-2.554	-2.935	-5.305	2.073	0.5
45	-1.622	0.2591	1.0	-2.118	-3.942	-4.831	2.747	0.5
46	-1.78	-0.8012	0.3	-3.0	-3.605	-5.333	3.0	0.5
47	-1.605	-0.9969	1.0	-1.601	-2.966	-4.455	2.017	0.5
48	-1.609	-0.04149	1.0	-1.554	-3.734	-5.27	1.507	0.5
49	-1.621	0.09739	0.8789	-1.223	-2.633	-5.104	2.702	0.9513
50	-1.618	0.3	1.0	-1.98	-2.207	-5.376	1.293	0.5
51	-1.652	0.3	1.0	-0.5831	-3.943	-4.469	2.491	0.5
52	-1.734	-0.9158	1.0	-2.969	-2.0	-5.482	1.0	1.0
53	-1.605	0.3	1.0	-2.124	-2.78	-4.309	2.066	0.5
54	-1.686	0.3	1.0	-2.8	-3.998	-4.605	1.153	0.5
55	-1.627	0.3	1.0	-2.317	-1.671	-5.077	2.77	0.5
56	-1.634	-4.956	0.8517	-2.793	-3.61	-5.846	2.85	0.753

```
# 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': 1.8143466498648387e-06,
 'eta': 0.0970581980025668,
 'gamma': 0.00041160003988771333,
 'max_depth': 3,
 'subsample': 0.6447492980647876,
 'lambda': 0.005780282239906642,
 'objective': 'reg:squarederror',
 'eval_metric': 'rmse',
 'colsample_bytree': 0.9131916743594339}
```

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         | -1.592    | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -1.483    | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -1.534    | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
```

4	-1.654	-3.147	0.6275	-2.063	1.604	-0.4504	1.466	0.7294
5	-1.566	-2.356	0.4052	-0.3806	-0.09483	-5.01	1.643	0.6674
6	-1.569	-2.162	0.5228	-2.659	-5.793	-3.144	2.227	0.7522
7	-1.551	-0.1605	0.6002	-1.973	0.8823	-3.597	1.193	0.6362
8	-1.504	-0.9486	0.7394	-0.4943	-0.4982	-3.564	2.166	0.5334
9	-1.522	-1.814	0.8098	-2.344	-2.07	-3.54	1.193	0.8742
10	-1.56	0.06321	0.6188	-0.4746	-0.88	-0.04974	2.478	0.7132
11	-1.542	-5.294	0.6658	-1.215	-5.342	-0.3629	2.421	0.9975
12	-1.622	-4.018	0.5023	-2.865	-0.7772	-2.167	2.849	0.9048
13	-1.477	-5.736	0.7323	-1.385	-3.443	-1.267	1.381	0.8479
14	-1.509	-4.497	0.9763	-1.066	-3.183	-1.361	1.0	0.5
15	-1.531	-5.569	0.6173	-2.111	-2.615	-0.5689	1.211	0.5
16	-1.466	-5.204	0.7135	-1.388	-2.85	-1.831	1.981	0.9342
17	-1.521	-5.694	0.9343	-1.793	-3.029	-2.629	1.079	1.0
18	-1.594	-5.566	0.3124	-0.3799	-2.642	-1.238	2.008	1.0
19	-1.489	-4.999	0.9307	-1.94	-3.211	-1.569	1.718	0.7673
20	-1.532	-5.065	0.3	-1.622	-2.747	-1.902	1.44	0.5
21	-1.486	-5.674	1.0	-1.506	-3.387	-1.795	2.028	1.0
22	-1.496	-4.543	1.0	-1.514	-2.659	-1.56	2.235	1.0
23	-1.487	-5.493	1.0	-1.826	-2.421	-1.567	1.919	1.0
24	-1.5	-5.182	1.0	-1.595	-2.758	-2.514	2.436	1.0
25	-1.514	-5.436	1.0	-1.351	-4.249	-1.52	1.0	1.0
26	-1.542	-5.319	0.3351	-1.932	-3.1	-1.522	2.547	1.0
27	-1.475	-5.226	1.0	-1.3	-2.865	-1.619	1.382	1.0
28	-1.529	-3.836	1.0	-1.964	-2.295	-0.9241	1.0	1.0
29	-1.494	-4.798	1.0	-0.9794	-3.405	-2.251	1.843	1.0
30	-1.478	-4.697	1.0	-1.243	-1.978	-2.177	1.594	1.0
31	-1.502	-3.63	1.0	-0.7825	-2.029	-2.309	1.527	0.5
32	-1.487	-4.715	1.0	-0.4656	-2.014	-3.38	1.625	1.0
33	-1.503	-4.725	1.0	-0.5147	-0.9607	-2.811	1.0	1.0
34	-1.538	-3.747	1.0	0.0	-2.665	-3.611	1.0	1.0
35	-1.515	-5.402	1.0	-0.801	-1.543	-2.923	2.097	0.5
36	-1.518	-4.402	1.0	-1.567	-1.757	-3.441	1.123	1.0
37	-1.538	-5.597	1.0	0.0	-1.542	-4.242	1.227	1.0
38	-1.517	-4.125	1.0	-0.165	-1.594	-2.846	2.41	1.0
39	-1.48	0.03424	1.0	0.0	-1.2	-3.918	3.0	0.5
40	-1.54	0.04954	1.0	0.0	-1.751	-3.964	1.927	0.5
41	-1.514	0.1303	1.0	0.0	-0.1083	-3.742	3.0	0.5
42	-1.555	-0.7936	1.0	0.0	-1.263	-3.339	3.0	1.0
43	-1.489	-4.593	1.0	-1.203	-1.466	-1.485	1.387	0.5
44	-1.494	-4.851	0.935	-0.1538	-2.206	-2.432	1.018	0.9766
45	-1.523	-4.05	1.0	-0.2374	-1.385	-1.646	1.0	1.0
46	-1.472	0.3	0.8367	-0.3183	-1.037	-4.812	3.0	0.5
47	-1.479	0.3	1.0	-1.099	-1.109	-4.229	3.0	0.5
48	-1.527	0.3	0.3	-0.6027	-1.715	-4.459	3.0	0.5
49	-1.476	-0.3601	1.0	-0.815	-0.5424	-4.82	3.0	0.5
50	-1.487	0.3	1.0	-1.237	-0.6958	-5.547	3.0	0.5
51	-1.557	0.06674	0.3517	-0.3089	-0.1756	-5.308	2.71	0.926
52	-1.476	-0.5299	1.0	-1.743	-1.145	-4.928	3.0	0.5
53	-1.478	6.574e-05	1.0	-2.012	-0.3042	-4.579	3.0	0.5
54	-1.505	0.3	1.0	-2.669	-1.15	-5.25	3.0	0.5
55	-1.481	-0.988	1.0	-1.731	-0.5035	-4.081	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': 1.9952623149688795,
 'eta': 1.0,
 'gamma': 3.5362952331631605e-06,
 'max_depth': 8,
 'subsample': 0.5,
 'lambda': 0.0013228849154839303,
 'objective': 'reg:squarederror',
```

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

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:1.83281	test-rmse:1.62397
[1]	train-rmse:1.80099	test-rmse:1.60884
[2]	train-rmse:1.74685	test-rmse:1.55963
[3]	train-rmse:1.69571	test-rmse:1.54121
[4]	train-rmse:1.63720	test-rmse:1.49432
[5]	train-rmse:1.60046	test-rmse:1.46383
[6]	train-rmse:1.56640	test-rmse:1.44605
[7]	train-rmse:1.53273	test-rmse:1.42124
[8]	train-rmse:1.49641	test-rmse:1.39748
[9]	train-rmse:1.46776	test-rmse:1.37692
[10]	train-rmse:1.44477	test-rmse:1.37356
[11]	train-rmse:1.41625	test-rmse:1.36399
[12]	train-rmse:1.39234	test-rmse:1.35255
[13]	train-rmse:1.37290	test-rmse:1.35734
[14]	train-rmse:1.34499	test-rmse:1.34116
[15]	train-rmse:1.32109	test-rmse:1.32393
[16]	train-rmse:1.29797	test-rmse:1.31431
[17]	train-rmse:1.28107	test-rmse:1.31604
[18]	train-rmse:1.25722	test-rmse:1.32725
[19]	train-rmse:1.23970	test-rmse:1.33870
[20]	train-rmse:1.23085	test-rmse:1.33786
[21]	train-rmse:1.21392	test-rmse:1.34890
[22]	train-rmse:1.19262	test-rmse:1.34346
[23]	train-rmse:1.17355	test-rmse:1.33278
[24]	train-rmse:1.15916	test-rmse:1.33643
[25]	train-rmse:1.14377	test-rmse:1.33581
[26]	train-rmse:1.13682	test-rmse:1.32983
[27]	train-rmse:1.12799	test-rmse:1.32334
[28]	train-rmse:1.12069	test-rmse:1.32276
[29]	train-rmse:1.11284	test-rmse:1.31973
[30]	train-rmse:1.10153	test-rmse:1.32310
[31]	train-rmse:1.09153	test-rmse:1.32107
[32]	train-rmse:1.08087	test-rmse:1.32216
[33]	train-rmse:1.07602	test-rmse:1.31794
[34]	train-rmse:1.06127	test-rmse:1.31838
[35]	train-rmse:1.04700	test-rmse:1.32250
[36]	train-rmse:1.02933	test-rmse:1.33290
[37]	train-rmse:1.01959	test-rmse:1.33235
[38]	train-rmse:1.00896	test-rmse:1.33122
[39]	train-rmse:1.00218	test-rmse:1.33058
[40]	train-rmse:0.99702	test-rmse:1.33031
[41]	train-rmse:0.98924	test-rmse:1.32241
[42]	train-rmse:0.98024	test-rmse:1.32625
[43]	train-rmse:0.96780	test-rmse:1.33809
[44]	train-rmse:0.95884	test-rmse:1.34195
[45]	train-rmse:0.95332	test-rmse:1.34101
[46]	train-rmse:0.93901	test-rmse:1.34511
```

```
# 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:1.83281	test-rmse:1.62397
[1]	train-rmse:1.80099	test-rmse:1.60884
[2]	train-rmse:1.74685	test-rmse:1.55963
[3]	train-rmse:1.69571	test-rmse:1.54121
[4]	train-rmse:1.63720	test-rmse:1.49432
[5]	train-rmse:1.60046	test-rmse:1.46383
[6]	train-rmse:1.56640	test-rmse:1.44605
[7]	train-rmse:1.53273	test-rmse:1.42124
[8]	train-rmse:1.49641	test-rmse:1.39748
[9]	train-rmse:1.46776	test-rmse:1.37692
[10]	train-rmse:1.44477	test-rmse:1.37356
[11]	train-rmse:1.41625	test-rmse:1.36399
[12]	train-rmse:1.39234	test-rmse:1.35255
[13]	train-rmse:1.37290	test-rmse:1.35734
[14]	train-rmse:1.34499	test-rmse:1.34116
[15]	train-rmse:1.32109	test-rmse:1.32393
[16]	train-rmse:1.29797	test-rmse:1.31431
[17]	train-rmse:1.28107	test-rmse:1.31604
[18]	train-rmse:1.25722	test-rmse:1.32725
[19]	train-rmse:1.23970	test-rmse:1.33870
[20]	train-rmse:1.23085	test-rmse:1.33786
[21]	train-rmse:1.21392	test-rmse:1.34890
[22]	train-rmse:1.19262	test-rmse:1.34346
[23]	train-rmse:1.17355	test-rmse:1.33278
[24]	train-rmse:1.15916	test-rmse:1.33643
[25]	train-rmse:1.14377	test-rmse:1.33581
[26]	train-rmse:1.13682	test-rmse:1.32983
[27]	train-rmse:1.12799	test-rmse:1.32334
[28]	train-rmse:1.12069	test-rmse:1.32276
[29]	train-rmse:1.11284	test-rmse:1.31973
[30]	train-rmse:1.10153	test-rmse:1.32310
[31]	train-rmse:1.09153	test-rmse:1.32107
[32]	train-rmse:1.08087	test-rmse:1.32216
[33]	train-rmse:1.07602	test-rmse:1.31794
[34]	train-rmse:1.06127	test-rmse:1.31838
[35]	train-rmse:1.04700	test-rmse:1.32250
[36]	train-rmse:1.02933	test-rmse:1.33290
[37]	train-rmse:1.01959	test-rmse:1.33235
[38]	train-rmse:1.00896	test-rmse:1.33122
[39]	train-rmse:1.00218	test-rmse:1.33058
[40]	train-rmse:0.99702	test-rmse:1.33031
[41]	train-rmse:0.98924	test-rmse:1.32241
[42]	train-rmse:0.98024	test-rmse:1.32625
[43]	train-rmse:0.96780	test-rmse:1.33809
[44]	train-rmse:0.95884	test-rmse:1.34195
[45]	train-rmse:0.95332	test-rmse:1.34101
[46]	train-rmse:0.93901	test-rmse:1.34511
```

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:1.40870	test-rmse:1.71140
[1]	train-rmse:1.24142	test-rmse:1.80005
[2]	train-rmse:1.23788	test-rmse:1.89925
[3]	train-rmse:1.17688	test-rmse:2.07116
[4]	train-rmse:1.16131	test-rmse:2.06647
[5]	train-rmse:1.14904	test-rmse:2.07194
```

```
[6]	train-rmse:1.13173	test-rmse:2.09764
[7]	train-rmse:1.09817	test-rmse:2.06489
[8]	train-rmse:1.09566	test-rmse:2.07525
[9]	train-rmse:1.09732	test-rmse:2.00322
[10]	train-rmse:1.11771	test-rmse:2.03887
[11]	train-rmse:1.11762	test-rmse:1.97840
[12]	train-rmse:1.09616	test-rmse:2.02052
[13]	train-rmse:1.10368	test-rmse:2.05023
[14]	train-rmse:1.08516	test-rmse:2.11001
[15]	train-rmse:1.08605	test-rmse:2.01925
[16]	train-rmse:1.08371	test-rmse:2.07020
[17]	train-rmse:1.10454	test-rmse:2.13831
[18]	train-rmse:1.07585	test-rmse:2.05609
[19]	train-rmse:1.08941	test-rmse:2.02826
[20]	train-rmse:1.07219	test-rmse:2.02971
[21]	train-rmse:1.08773	test-rmse:2.15171
[22]	train-rmse:1.07759	test-rmse:2.15857
[23]	train-rmse:1.07577	test-rmse:2.05925
[24]	train-rmse:1.05174	test-rmse:1.99394
[25]	train-rmse:1.04971	test-rmse:2.05482
[26]	train-rmse:1.07189	test-rmse:1.94906
[27]	train-rmse:1.05623	test-rmse:2.04827
[28]	train-rmse:1.06337	test-rmse:2.03752
[29]	train-rmse:1.06458	test-rmse:2.04442
[30]	train-rmse:1.05708	test-rmse:1.97872
```

```
# 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:1.40870	test-rmse:1.71140
[1]	train-rmse:1.24142	test-rmse:1.80005
[2]	train-rmse:1.23788	test-rmse:1.89925
[3]	train-rmse:1.17688	test-rmse:2.07116
[4]	train-rmse:1.16131	test-rmse:2.06647
[5]	train-rmse:1.14904	test-rmse:2.07194
[6]	train-rmse:1.13173	test-rmse:2.09764
[7]	train-rmse:1.09817	test-rmse:2.06489
[8]	train-rmse:1.09566	test-rmse:2.07525
[9]	train-rmse:1.09732	test-rmse:2.00322
[10]	train-rmse:1.11771	test-rmse:2.03887
[11]	train-rmse:1.11762	test-rmse:1.97840
[12]	train-rmse:1.09616	test-rmse:2.02052
[13]	train-rmse:1.10368	test-rmse:2.05023
[14]	train-rmse:1.08516	test-rmse:2.11001
[15]	train-rmse:1.08605	test-rmse:2.01925
[16]	train-rmse:1.08371	test-rmse:2.07020
[17]	train-rmse:1.10454	test-rmse:2.13831
[18]	train-rmse:1.07585	test-rmse:2.05609
[19]	train-rmse:1.08941	test-rmse:2.02826
[20]	train-rmse:1.07219	test-rmse:2.02971
[21]	train-rmse:1.08773	test-rmse:2.15171
[22]	train-rmse:1.07759	test-rmse:2.15857
[23]	train-rmse:1.07577	test-rmse:2.05925
[24]	train-rmse:1.05174	test-rmse:1.99394
[25]	train-rmse:1.04971	test-rmse:2.05482
```

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:1.50822	test-rmse:1.51609
[1]	train-rmse:1.20441	test-rmse:1.50001
[2]	train-rmse:0.98990	test-rmse:1.49717
[3]	train-rmse:0.82211	test-rmse:1.50169
[4]	train-rmse:0.71358	test-rmse:1.52169
[5]	train-rmse:0.62480	test-rmse:1.50629
[6]	train-rmse:0.54690	test-rmse:1.49833
[7]	train-rmse:0.48596	test-rmse:1.49060
[8]	train-rmse:0.42910	test-rmse:1.50414
[9]	train-rmse:0.38121	test-rmse:1.50953
[10]	train-rmse:0.33375	test-rmse:1.50861
[11]	train-rmse:0.29921	test-rmse:1.50841
[12]	train-rmse:0.26334	test-rmse:1.51815
[13]	train-rmse:0.23309	test-rmse:1.51110
[14]	train-rmse:0.22078	test-rmse:1.50781
[15]	train-rmse:0.20472	test-rmse:1.51623
[16]	train-rmse:0.19130	test-rmse:1.51380
[17]	train-rmse:0.17399	test-rmse:1.51761
[18]	train-rmse:0.16732	test-rmse:1.52052
[19]	train-rmse:0.15921	test-rmse:1.52261
[20]	train-rmse:0.14906	test-rmse:1.52428
[21]	train-rmse:0.13491	test-rmse:1.51987
[22]	train-rmse:0.13105	test-rmse:1.52192
[23]	train-rmse:0.12136	test-rmse:1.52082
[24]	train-rmse:0.11906	test-rmse:1.52181
[25]	train-rmse:0.10825	test-rmse:1.52442
[26]	train-rmse:0.09492	test-rmse:1.52286
[27]	train-rmse:0.08642	test-rmse:1.52315
[28]	train-rmse:0.08302	test-rmse:1.52431
[29]	train-rmse:0.07634	test-rmse:1.52492
[30]	train-rmse:0.07197	test-rmse:1.52554
[31]	train-rmse:0.06772	test-rmse:1.52645
[32]	train-rmse:0.06204	test-rmse:1.52908
[33]	train-rmse:0.05586	test-rmse:1.52984
[34]	train-rmse:0.05126	test-rmse:1.53033
[35]	train-rmse:0.04565	test-rmse:1.52998
[36]	train-rmse:0.04242	test-rmse:1.52988
```

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:1.54958	test-rmse:1.48793
[1]	train-rmse:1.24651	test-rmse:1.46705
[2]	train-rmse:0.97843	test-rmse:1.47737
[3]	train-rmse:0.79490	test-rmse:1.49702
[4]	train-rmse:0.66246	test-rmse:1.51259
[5]	train-rmse:0.56802	test-rmse:1.53194
[6]	train-rmse:0.50260	test-rmse:1.53623
[7]	train-rmse:0.42722	test-rmse:1.55193
[8]	train-rmse:0.38135	test-rmse:1.56244
[9]	train-rmse:0.33632	test-rmse:1.57958
[10]	train-rmse:0.28438	test-rmse:1.58056
[11]	train-rmse:0.26198	test-rmse:1.58829
[12]	train-rmse:0.23728	test-rmse:1.60317
[13]	train-rmse:0.21035	test-rmse:1.59406
[14]	train-rmse:0.18668	test-rmse:1.60318
[15]	train-rmse:0.16417	test-rmse:1.59872
```

```
[16]	train-rmse:0.15322	test-rmse:1.60157
[17]	train-rmse:0.13933	test-rmse:1.60271
[18]	train-rmse:0.12366	test-rmse:1.60606
[19]	train-rmse:0.11291	test-rmse:1.60355
[20]	train-rmse:0.10587	test-rmse:1.60410
[21]	train-rmse:0.10023	test-rmse:1.60809
[22]	train-rmse:0.09571	test-rmse:1.60888
[23]	train-rmse:0.08612	test-rmse:1.61020
[24]	train-rmse:0.08365	test-rmse:1.61056
[25]	train-rmse:0.08099	test-rmse:1.61055
[26]	train-rmse:0.06964	test-rmse:1.60636
[27]	train-rmse:0.06130	test-rmse:1.60454
[28]	train-rmse:0.05457	test-rmse:1.60606
[29]	train-rmse:0.05111	test-rmse:1.60840
[30]	train-rmse:0.04773	test-rmse:1.60865
[31]	train-rmse:0.04429	test-rmse:1.60893
```

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: (0.3604048020224956, 'good'),
PBIAS: (5.258534577934008, 'good'),
KGE: (0.49654117196534, 'good')

Root Mean Squared Error: 1.3451057796734676
Normalized Root Mean Squared Error: 0.263333159685487
```

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: (0.17250000700780665, 'bad'),
PBIAS: (6.849152214791969, 'good'),
KGE: (0.481718522004173, 'good')

Root Mean Squared Error: 1.5299873530816124
Normalized Root Mean Squared Error: 0.2995276728820697
```

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: 1.6828674290904115
Normalized Root Mean Squared Error: 0.3294572100803468
```

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: 1.6828999269810896
Normalized Root Mean Squared Error: 0.329463572235922
```

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: (0.3003406737574903, 'bad'),
PBIAS: (9.138080431028751, 'good'),
KGE: (0.46925790696022485, 'good')

Root Mean Squared Error: 1.3844292516749703
Normalized Root Mean Squared Error: 0.27744073179859124
```

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: (-0.5413266020400924, 'bad'),
PBIAS: (20.85801659458364, 'bad'),
KGE: (0.29327000235137435, 'good')

Root Mean Squared Error: 2.054824760065243
Normalized Root Mean Squared Error: 0.411788529071191
```

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: (0.05503191654014583, 'bad'),
PBIAS: (8.674598570215617, 'good'),
KGE: (0.44942410077765016, 'good')

Root Mean Squared Error: 1.608925827702348
Normalized Root Mean Squared Error: 0.3224300255916529
```

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: 1.6631881676095377
Normalized Root Mean Squared Error: 0.33330424200591935
```

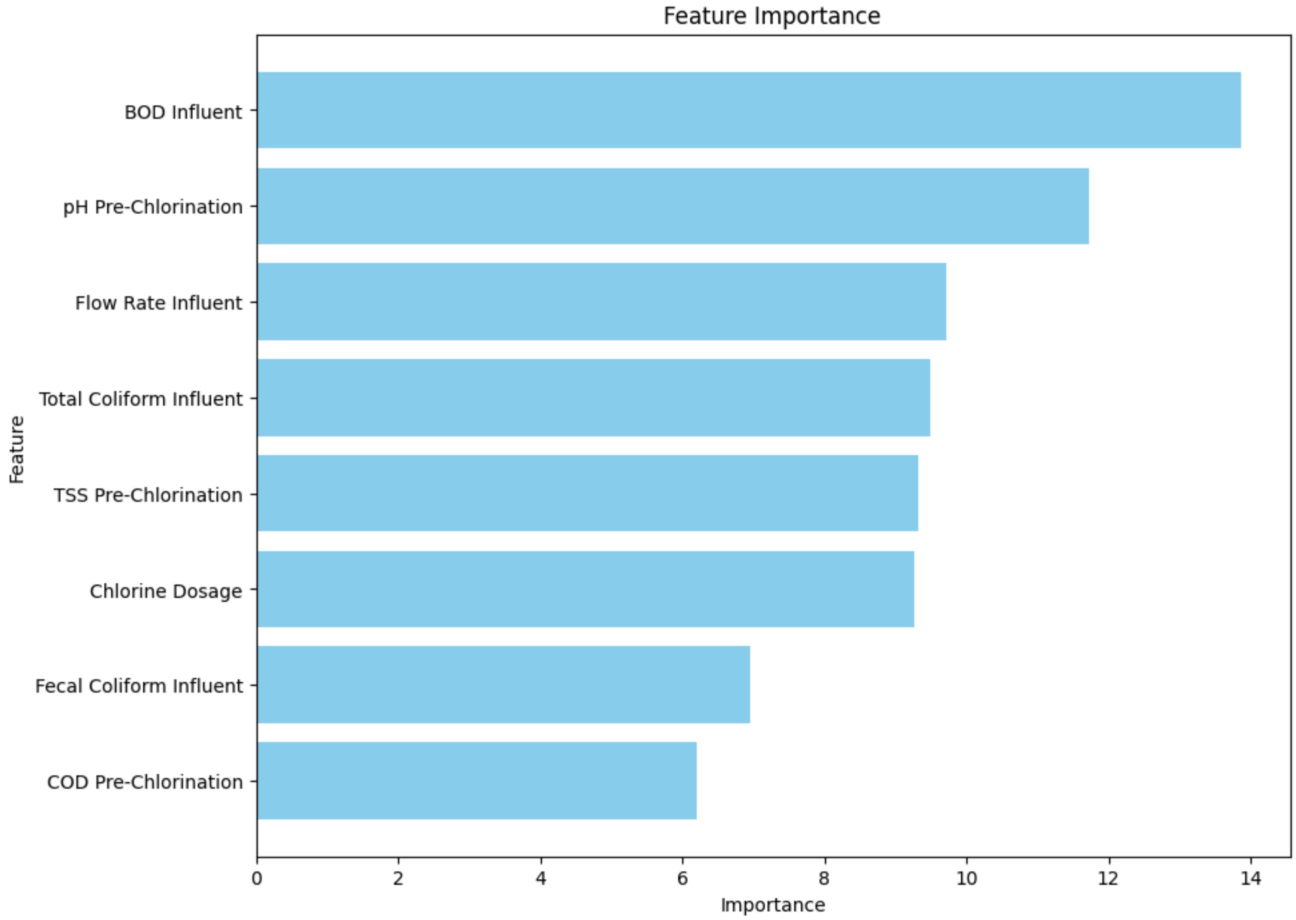
Naive Model 2

```
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: 1.6630941882756551
Normalized Root Mean Squared Error: 0.33328540847207516
```

Feature Importance

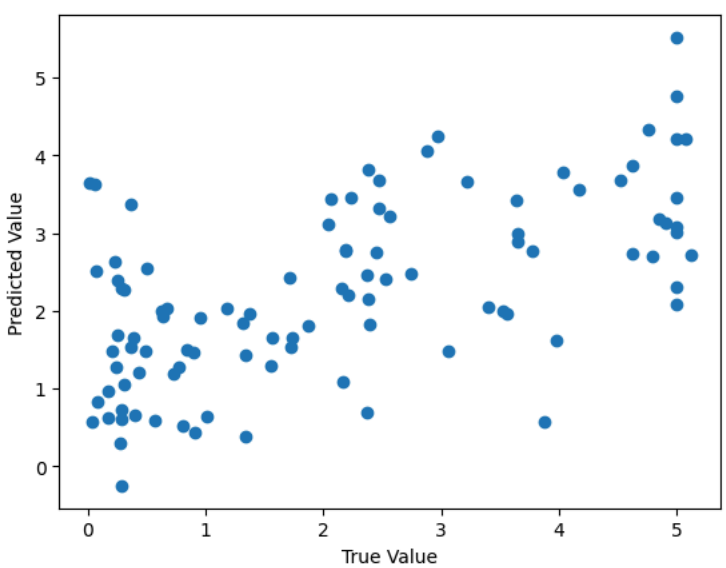
```
# Get feature importance
importance_MBBR = final_model_MBBR.get_score(importance_type='gain')

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

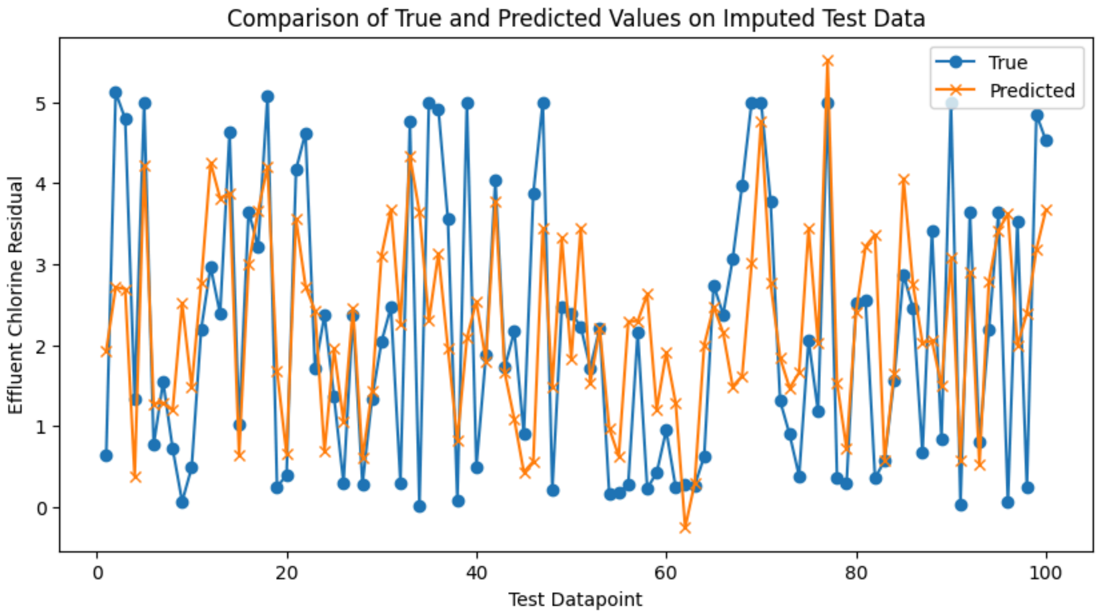
Optimized XGBoost on Imputed Test Dataset

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



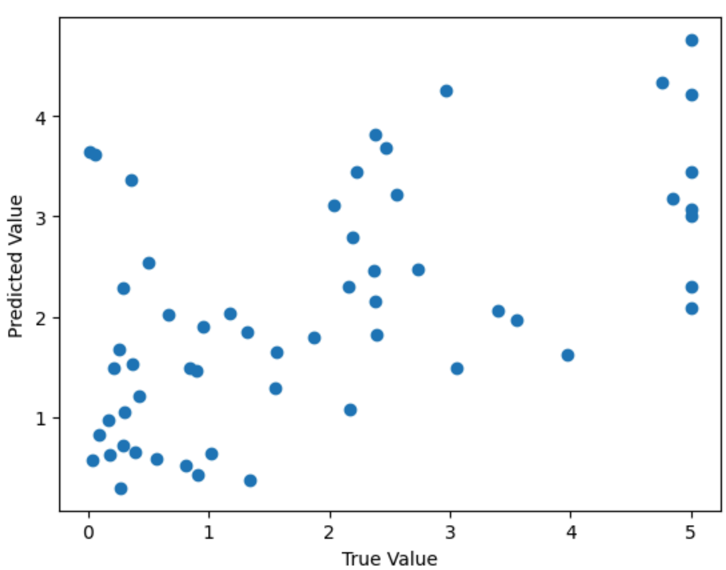

```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_MBBR) + 1)
```

```
# 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('Effluent Chlorine Residual')
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('Effluent Chlorine Residual')
plt.title('Comparison of True and Predicted Values on Raw Test Data')
plt.legend()

# Show plot
plt.show()
```