```
!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 | 4.51 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='Fecal Coliform Effluent (MPN/100mL)')
y_orig_MBBR = MBBR['Fecal 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['Fecal.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['Fecal.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
241	(Chlorine.dosage..L.d., BOD.Influent..ppm., BO...	2100.299445	7556.871893	296.516294	870.813651
114	(Chlorine.dosage..L.d., BOD.Influent..ppm., CO...	2136.827950	7581.065276	132.556804	443.237014
248	(Chlorine.dosage..L.d., BOD.Influent..ppm., CO...	1991.224671	7651.991078	124.770157	459.211068
246	(Chlorine.dosage..L.d., BOD.Influent..ppm., CO...	2117.826255	7677.704257	195.843326	934.008264
205	(Chlorine.dosage..L.d., Total.Coliform.Influen...	2164.685210	7724.491702	218.540663	792.364563
...	...	...	...	...	...
217	(Chlorine.dosage..L.d., Total.Coliform.Influen...	2622.006765	9449.279683	160.482385	1616.644110
38	(Chlorine.dosage..L.d., BOD.Pre.chlorination.....	3145.176730	9459.540961	271.785598	785.716311
6	(Chlorine.dosage..L.d., COD.Pre.chlorination.....	3860.637185	9509.776645	304.727469	1239.469209
27	(Chlorine.dosage..L.d., Fecal.Coliform.Influen...	2965.036467	9652.201636	297.621937	679.191297
193	(Chlorine.dosage..L.d., Total.Coliform.Influen...	2490.729270	9861.735305	316.960876	839.772053

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
241	(Chlorine.dosage..L.d., BOD.Influent..ppm., BO...	2100.299445	7556.871893	296.516294	870.813651
114	(Chlorine.dosage..L.d., BOD.Influent..ppm., CO...	2136.827950	7581.065276	132.556804	443.237014
248	(Chlorine.dosage..L.d., BOD.Influent..ppm., CO...	1991.224671	7651.991078	124.770157	459.211068

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

```
('Chlorine.dosage..L.d.',
 'BOD.Influent..ppm.',
 'BOD.Pre.chlorination..ppm.',
 'COD.Influent..ppm.',
 'TSS.Pre.chlorination..ppm.')
```

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

```
('Chlorine.dosage..L.d.',
 'BOD.Influent..ppm.',
 'COD.Influent..ppm.',
 'TSS.Pre.chlorination..ppm.')
```

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

```
('Chlorine.dosage..L.d.',
 '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.',
 'BOD.Influent..ppm.',
 'BOD.Pre.chlorination..ppm.',
 'COD.Influent..ppm.',
 '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)',
       'Total Coliform Effluent (MPN/100mL)',
       'Fecal Coliform Influent (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
116	(Chlorine dosage (L/d), Total Coliform Influen...	321.229337	428.798484	7.617849	98.369977
21	(Chlorine dosage (L/d), Total Coliform Influen...	321.229337	428.798484	7.617849	98.369977
366	(Chlorine dosage (L/d), Total Coliform Influen...	321.235363	428.804774	7.620255	98.377382
113	(Chlorine dosage (L/d), Total Coliform Influen...	321.236135	428.807746	7.620590	98.380688
365	(Chlorine dosage (L/d), Total Coliform Influen...	321.235966	428.812640	7.619451	98.387105
...	...	...	...	...	...
1555	(Chlorine dosage (L/d), Flow Rate Influent (m3...	763.317070	7417.047809	89.546959	1610.559146
1114	(Chlorine dosage (L/d), Flow Rate Influent (m3...	763.336947	7417.231945	89.552467	1610.327706
1219	(Chlorine dosage (L/d), Flow Rate Influent (m3...	762.391358	7418.023906	89.187099	1607.545426
91	(Chlorine dosage (L/d), Flow Rate Influent (m3...	762.420873	7477.601997	89.561682	2265.881832
101	(Chlorine dosage (L/d), Flow Rate Influent (m3...	772.338354	7494.328554	81.490661	1493.398012

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
116	(Chlorine dosage (L/d), Total Coliform Influen...	321.229337	428.798484	7.617849	98.369977
21	(Chlorine dosage (L/d), Total Coliform Influen...	321.229337	428.798484	7.617849	98.369977
366	(Chlorine dosage (L/d), Total Coliform Influen...	321.235363	428.804774	7.620255	98.377382

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

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

```
 'Total Coliform Effluent (MPN/100mL)',
 'TSS Pre-chlorination (ppm)')
```

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

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

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

```
('Chlorine dosage (L/d)',
 'Total Coliform Influent (MPN/100mL)',
 'Total Coliform Effluent (MPN/100mL)',
 'BOD Pre-chlorination\n(ppm)',
 'TSS Pre-chlorination (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 Influent (MPN/100mL)',
 'Total Coliform Effluent (MPN/100mL)',
 'TSS Pre-chlorination (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         | -7.742e+0 | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
```

```
| 2         | -7.577e+0 | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -7.662e+0 | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -7.659e+0 | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
| 5         | -7.926e+0 | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -7.791e+0 | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -7.715e+0 | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -8.006e+0 | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -7.774e+0 | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -7.851e+0 | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -7.668e+0 | -4.263    | 0.7014    | -2.23     | -0.8528   | -0.698    | 1.447     | 0.6343    |
| 12        | -7.629e+0 | -3.808    | 0.6678    | -1.792    | -3.422    | -0.3647   | 1.883     | 0.663     |
| 13        | -7.678e+0 | -6.0      | 1.0       | -2.108    | -3.252    | -1.176    | 1.0       | 0.5       |
| 14        | -8.143e+0 | -5.274    | 0.3       | -0.8337   | -2.395    | -1.086    | 3.0       | 0.5       |
| 15        | -7.669e+0 | -4.157    | 0.8436    | -2.202    | -2.551    | -0.7292   | 1.27      | 0.6293    |
| 16        | -7.693e+0 | -4.902    | 0.9656    | -2.257    | -1.818    | -1.625    | 1.0       | 0.5136    |
| 17        | -7.613e+0 | -3.599    | 0.7227    | -1.837    | -1.922    | -1.6      | 1.61      | 0.539     |
| 18        | -7.611e+0 | -4.058    | 0.8756    | -1.732    | -3.137    | -1.892    | 1.403     | 0.5       |
| 19        | -8.014e+0 | -4.11     | 0.3       | -2.661    | -2.647    | -1.851    | 1.945     | 1.0       |
| 20        | -7.621e+0 | -4.055    | 1.0       | -1.305    | -2.402    | -1.385    | 1.198     | 0.5       |
| 21        | -7.677e+0 | -4.591    | 1.0       | -1.503    | -3.441    | -1.06     | 1.243     | 0.5       |
| 22        | -7.817e+0 | -4.179    | 0.3       | -1.47     | -1.618    | -1.126    | 1.137     | 0.5       |
| 23        | -7.608e+0 | -3.841    | 1.0       | -1.615    | -2.672    | -1.204    | 1.906     | 0.5       |
| 24        | -7.655e+0 | -3.191    | 1.0       | -1.542    | -3.226    | -1.241    | 1.233     | 0.5       |
| 25        | -7.69e+03 | -4.221    | 1.0       | -1.421    | -2.187    | -2.215    | 1.683     | 0.5       |
| 26        | -7.605e+0 | -3.453    | 1.0       | -2.005    | -1.675    | -0.6944   | 2.099     | 0.5       |
| 27        | -7.519e+0 | -2.831    | 1.0       | -1.707    | -2.631    | -0.08403  | 2.034     | 0.5       |
| 28        | -7.472e+0 | -2.495    | 1.0       | -1.631    | -3.52     | 0.07099   | 2.395     | 0.5       |
| 29        | -7.519e+0 | -2.419    | 1.0       | -2.533    | -3.166    | 0.3       | 2.466     | 0.5       |
| 30        | -7.951e+0 | -2.035    | 0.3       | -1.793    | -3.326    | 0.3       | 1.833     | 0.5       |
| 31        | -7.562e+0 | -2.945    | 1.0       | -1.93     | -3.168    | -0.08203  | 2.811     | 0.5       |
| 32        | -7.547e+0 | -3.033    | 1.0       | -2.085    | -3.668    | 0.3       | 2.212     | 0.5       |
| 33        | -7.49e+03 | -2.335    | 1.0       | -2.08     | -4.033    | -0.0214   | 2.903     | 0.5508    |
| 34        | -7.585e+0 | -2.741    | 1.0       | -1.141    | -4.195    | -0.1525   | 2.766     | 0.5       |
| 35        | -7.797e+0 | -2.441    | 1.0       | -2.038    | -3.631    | -0.5648   | 2.444     | 1.0       |
| 36        | -7.499e+0 | -2.302    | 1.0       | -1.625    | -3.552    | 0.3       | 3.0       | 0.5       |
| 37        | -7.531e+0 | -3.035    | 1.0       | -1.044    | -3.129    | 0.3       | 2.404     | 0.5       |
| 38        | -7.626e+0 | -2.464    | 1.0       | -2.992    | -4.047    | 0.3       | 3.0       | 0.5       |
| 39        | -7.566e+0 | -2.073    | 1.0       | -1.817    | -4.8      | 0.3       | 3.0       | 0.5       |
| 40        | -7.541e+0 | -2.579    | 1.0       | -2.101    | -2.029    | 0.3       | 2.767     | 0.5       |
| 41        | -7.588e+0 | -3.129    | 1.0       | -2.906    | -2.255    | 0.3       | 2.189     | 0.5       |
| 42        | -7.496e+0 | -2.858    | 1.0       | -0.9569   | -1.768    | 0.3       | 2.366     | 0.5       |
| 43        | -7.52e+03 | -2.578    | 1.0       | -1.643    | -1.1      | 0.3       | 1.812     | 0.5       |
| 44        | -7.51e+03 | -2.875    | 1.0       | -1.303    | -0.674    | 0.3       | 2.907     | 0.5       |
| 45        | -7.66e+03 | -2.83     | 1.0       | -2.432    | -0.407    | 0.3       | 2.581     | 1.0       |
| 46        | -8.023e+0 | -2.788    | 0.972     | -0.4285   | -0.4408   | -0.1332   | 2.05      | 0.8096    |
| 47        | -7.631e+0 | -3.283    | 0.6171    | -1.673    | -1.64     | 0.289     | 2.209     | 0.975     |
| 48        | -7.65e+03 | -1.706    | 0.6838    | -1.322    | -1.886    | 0.1303    | 2.719     | 0.5454    |
| 49        | -7.675e+0 | -2.101    | 0.9847    | -2.658    | -1.26     | -0.1563   | 1.674     | 0.7453    |
| 50        | -7.53e+03 | -1.288    | 1.0       | -2.312    | -3.954    | 0.3       | 3.0       | 0.5       |
| 51        | -7.58e+03 | -1.064    | 0.8441    | -2.864    | -4.966    | -0.1234   | 2.641     | 0.5004    |
| 52        | -7.571e+0 | 0.2358    | 0.7083    | -2.007    | -5.425    | 0.048     | 2.967     | 0.6229    |
| 53        | -7.821e+0 | -0.1165   | 0.4287    | -2.909    | -5.705    | 0.1773    | 1.876     | 0.7659    |
| 54        | -7.575e+0 | -0.188    | 0.923     | -2.899    | -4.148    | 0.1165    | 2.943     | 0.7923    |
| 55        | -7.63e+03 | 0.1055    | 0.9078    | -1.885    | -3.89     | -0.7125   | 2.692     | 0.866     |
| 56        | -7.545e+0 | -0.392    | 1.0       | -1.701    | -4.555    | 0.3       | 3.0       | 0.5       |
```

```
# 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': 0.23462247187264512,
 'eta': 0.1090450524366011,
 'gamma': 0.2761597227720788,
 'max_depth': 7,
```

```
      'subsample': 0.5,
      'lambda': 1.9952623149688795,
      'objective': 'reg:squarederror',
      'eval_metric': 'rmse',
      'colsample_bytree': 1.0}


# 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	-2.993e+0	0.04075	0.4513	-2.68	-1.662	-1.582	2.026	0.7673
2	-788.6	-4.514	0.7529	-1.843	-2.2	-1.339	1.596	0.5436
3	-1.497e+0	-1.108	0.5069	-1.136	-4.974	-0.5216	2.693	0.8202
4	-1.363e+0	-3.147	0.6275	-2.063	1.604	-0.4504	1.466	0.7294
5	-2.517e+0	-2.356	0.4052	-0.3806	-0.09483	-5.01	1.643	0.6674
6	-2.053e+0	-2.162	0.5228	-2.659	-5.793	-3.144	2.227	0.7522
7	-1.074e+0	-0.1605	0.6002	-1.973	0.8823	-3.597	1.193	0.6362
8	-2.906e+0	-0.9486	0.7394	-0.4943	-0.4982	-3.564	2.166	0.5334
9	-793.5	-1.814	0.8098	-2.344	-2.07	-3.54	1.193	0.8742
10	-2.633e+0	0.06321	0.6188	-0.4746	-0.88	-0.04974	2.478	0.7132
11	-762.4	-4.02	0.7641	-1.938	-2.176	-1.745	1.519	0.6054
12	-1.525e+0	-3.339	1.0	-3.0	-2.252	-3.317	1.0	0.9576
13	-1.052e+0	-4.102	0.6572	-2.021	-0.8261	-1.182	1.0	0.6043
14	-2.281e+0	-3.94	0.3	-0.5554	-2.447	-1.386	1.0	1.0
15	-1.253e+0	-4.292	1.0	-2.802	-1.582	-1.468	2.15	0.5
16	-427.9	-5.083	1.0	-2.177	-1.731	-2.181	1.0	0.5
17	-2.988e+0	-5.385	0.3	-2.847	-2.567	-1.887	1.0	0.5
18	-446.6	-4.613	1.0	-1.802	-1.411	-1.983	1.392	0.5
19	-208.8	-4.976	1.0	-2.038	-1.001	-2.621	1.0	1.0
20	-505.6	-5.747	1.0	-1.442	-0.9472	-2.42	1.0	0.5
21	-537.0	-4.978	1.0	-1.494	-1.554	-3.412	1.0	0.5
22	-648.7	-5.494	1.0	-2.551	-0.2134	-3.435	1.0	0.5
23	-313.6	-5.613	1.0	-1.856	-1.07	-3.142	2.34	1.0
24	-314.2	-5.893	1.0	-1.861	0.5326	-2.487	2.45	1.0
25	-313.5	-6.0	1.0	-0.9088	0.2978	-3.934	2.326	1.0
26	-484.0	-6.0	1.0	-2.6	0.5715	-4.182	3.0	1.0
27	-308.1	-6.0	1.0	-0.6029	1.8	-3.088	3.0	1.0
28	-312.0	-6.0	1.0	0.0	0.03643	-2.264	3.0	1.0
29	-282.8	-6.0	1.0	0.0	1.602	-2.3	1.187	1.0
30	-285.9	-6.0	1.0	0.0	1.8	-0.7617	2.872	1.0
31	-208.0	-6.0	1.0	-1.151	1.8	-0.3146	1.0	1.0
32	-357.8	-6.0	1.0	-2.553	1.8	-1.917	1.0	1.0
33	-1.649e+0	-5.885	0.7123	-2.289	1.663	0.1913	2.599	0.6501
34	-728.8	-5.573	0.7768	-0.002446	1.033	-0.3921	1.013	0.5452
35	-1.988e+0	-5.911	0.3211	-1.486	1.791	-5.844	1.812	0.571

```
| 36        | -313.5    | -6.0      | 1.0       | 0.0       | -1.806    | -4.359    | 3.0       | 1.0       |
| 37        | -308.9    | -6.0      | 1.0       | -1.819    | -2.025    | -5.751    | 3.0       | 1.0       |
| 38        | -310.0    | -6.0      | 1.0       | -0.07773  | -3.792    | -6.0      | 3.0       | 1.0       |
| 39        | -308.9    | -6.0      | 1.0       | -2.314    | -4.572    | -6.0      | 3.0       | 1.0       |
| 40        | -208.3    | -6.0      | 1.0       | -1.115    | -3.518    | -6.0      | 1.0       | 1.0       |
| 41        | -367.4    | -6.0      | 1.0       | -0.5232   | -5.876    | -6.0      | 1.436     | 0.5       |
| 42        | -394.7    | -4.221    | 1.0       | -1.004    | -4.176    | -6.0      | 2.184     | 0.5       |
| 43        | -4.455e+0 | -5.714    | 0.6728    | -0.3452   | -4.943    | -4.18     | 2.919     | 0.5961    |
| 44        | -314.2    | -6.0      | 1.0       | 0.0       | -1.909    | -6.0      | 1.881     | 1.0       |
| 45        | -1.195e+0 | -6.0      | 1.0       | -2.899    | -5.792    | -6.0      | 1.0       | 1.0       |
| 46        | -308.9    | -4.495    | 1.0       | -0.5012   | -2.392    | -6.0      | 3.0       | 1.0       |
| 47        | -1.529e+0 | -0.8644   | 1.0       | -3.0      | -3.109    | -6.0      | 1.0       | 1.0       |
| 48        | -1.529e+0 | 0.3       | 1.0       | -3.0      | 1.8       | -6.0      | 1.0       | 1.0       |
| 49        | -1.535e+0 | -4.479    | 1.0       | -3.0      | -3.158    | -6.0      | 3.0       | 1.0       |
| 50        | -309.9    | -3.719    | 1.0       | 0.0       | -5.795    | -6.0      | 1.0       | 1.0       |
| 51        | -217.6    | -1.257    | 1.0       | 0.0       | -6.0      | -6.0      | 1.0       | 0.5       |
| 52        | -309.9    | -4.238    | 1.0       | 0.0       | -3.427    | -6.0      | 1.0       | 1.0       |
| 53        | -312.4    | 0.3       | 1.0       | 0.0       | -6.0      | -6.0      | 2.847     | 1.0       |
| 54        | -4.576e+0 | -1.653    | 0.6737    | -0.4743   | -4.99     | -5.867    | 2.651     | 0.9358    |
| 55        | -5.571e+0 | -4.589    | 0.538     | -0.02375  | 1.374     | -2.436    | 2.524     | 0.7955    |
| 56        | -2.158e+0 | -5.373    | 0.3519    | -0.8827   | -2.406    | -5.47     | 2.124     | 0.5874    |
```

```
# 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': 1.854435961006547e-05,
 'eta': 1.0,
 'gamma': 6.220326073224884e-06,
 'max_depth': 2,
 'subsample': 1.0,
 'lambda': 0.17135367370598076,
 '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:7938.34661	test-rmse:6376.64259
[1]	train-rmse:7589.48878	test-rmse:6275.62083
[2]	train-rmse:7222.34557	test-rmse:6248.52757
[3]	train-rmse:6906.47431	test-rmse:6284.50982
[4]	train-rmse:6670.73687	test-rmse:6279.04361
[5]	train-rmse:6442.15273	test-rmse:6250.27330
[6]	train-rmse:6213.12298	test-rmse:6272.63744
[7]	train-rmse:6013.79024	test-rmse:6238.05461
[8]	train-rmse:5848.91577	test-rmse:6218.43112
[9]	train-rmse:5659.89979	test-rmse:6215.80775
[10]	train-rmse:5529.18883	test-rmse:6245.50303
[11]	train-rmse:5385.88174	test-rmse:6255.62951
[12]	train-rmse:5233.01081	test-rmse:6307.45699
[13]	train-rmse:5088.77098	test-rmse:6348.19896
[14]	train-rmse:4925.88947	test-rmse:6364.94993
[15]	train-rmse:4785.09633	test-rmse:6430.27950
[16]	train-rmse:4664.35442	test-rmse:6439.32849
[17]	train-rmse:4585.26810	test-rmse:6488.78365
[18]	train-rmse:4489.10982	test-rmse:6542.51794
[19]	train-rmse:4385.58931	test-rmse:6663.39762
[20]	train-rmse:4272.68376	test-rmse:6674.47286
[21]	train-rmse:4180.72574	test-rmse:6702.99268
[22]	train-rmse:4090.10598	test-rmse:6785.68805
[23]	train-rmse:3954.38341	test-rmse:6841.78496
[24]	train-rmse:3832.77041	test-rmse:6862.51487
[25]	train-rmse:3759.10478	test-rmse:6869.74787
[26]	train-rmse:3681.33131	test-rmse:6916.94729
[27]	train-rmse:3575.49641	test-rmse:6905.93555
[28]	train-rmse:3500.32614	test-rmse:6922.27536
[29]	train-rmse:3452.88837	test-rmse:6969.16661
[30]	train-rmse:3367.96491	test-rmse:6994.86683
[31]	train-rmse:3327.38264	test-rmse:7029.83162
[32]	train-rmse:3244.12376	test-rmse:7040.08481
[33]	train-rmse:3176.35687	test-rmse:7105.66843
[34]	train-rmse:3091.76204	test-rmse:7123.14758
[35]	train-rmse:3033.65153	test-rmse:7148.74169
[36]	train-rmse:2978.83142	test-rmse:7133.92043
[37]	train-rmse:2949.62726	test-rmse:7124.16713
[38]	train-rmse:2896.83021	test-rmse:7145.00846
[39]	train-rmse:2861.69180	test-rmse:7181.31705
```

```
# 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:7938.34661	test-rmse:6376.64259
[1]	train-rmse:7589.48878	test-rmse:6275.62083
[2]	train-rmse:7222.34557	test-rmse:6248.52757
[3]	train-rmse:6906.47431	test-rmse:6284.50982
[4]	train-rmse:6670.73687	test-rmse:6279.04361
[5]	train-rmse:6442.15273	test-rmse:6250.27330
[6]	train-rmse:6213.12298	test-rmse:6272.63744
[7]	train-rmse:6013.79024	test-rmse:6238.05461
[8]	train-rmse:5848.91577	test-rmse:6218.43112
[9]	train-rmse:5659.89979	test-rmse:6215.80775
[10]	train-rmse:5529.18883	test-rmse:6245.50303
[11]	train-rmse:5385.88174	test-rmse:6255.62951
[12]	train-rmse:5233.01081	test-rmse:6307.45699
[13]	train-rmse:5088.77098	test-rmse:6348.19896
[14]	train-rmse:4925.88947	test-rmse:6364.94993
[15]	train-rmse:4785.09633	test-rmse:6430.27950
[16]	train-rmse:4664.35442	test-rmse:6439.32849
[17]	train-rmse:4585.26810	test-rmse:6488.78365
[18]	train-rmse:4489.10982	test-rmse:6542.51794
[19]	train-rmse:4385.58931	test-rmse:6663.39762
[20]	train-rmse:4272.68376	test-rmse:6674.47286
[21]	train-rmse:4180.72574	test-rmse:6702.99268
[22]	train-rmse:4090.10598	test-rmse:6785.68805
[23]	train-rmse:3954.38341	test-rmse:6841.78496
[24]	train-rmse:3832.77041	test-rmse:6862.51487
[25]	train-rmse:3759.10478	test-rmse:6869.74787
[26]	train-rmse:3681.33131	test-rmse:6916.94729
[27]	train-rmse:3575.49641	test-rmse:6905.93555
```

```
[28]	train-rmse:3500.32614	test-rmse:6922.27536
[29]	train-rmse:3452.88837	test-rmse:6969.16661
[30]	train-rmse:3367.96491	test-rmse:6994.86683
[31]	train-rmse:3327.38264	test-rmse:7029.83162
[32]	train-rmse:3244.12376	test-rmse:7040.08481
[33]	train-rmse:3176.35687	test-rmse:7105.66843
[34]	train-rmse:3091.76204	test-rmse:7123.14758
[35]	train-rmse:3033.65153	test-rmse:7148.74169
[36]	train-rmse:2978.83142	test-rmse:7133.92043
[37]	train-rmse:2949.62726	test-rmse:7124.16713
[38]	train-rmse:2896.83021	test-rmse:7145.00846
[39]	train-rmse:2861.69180	test-rmse:7181.31705
```

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:241.40400	test-rmse:468.14798
[1]	train-rmse:98.18515	test-rmse:598.60394
[2]	train-rmse:70.43796	test-rmse:613.93280
[3]	train-rmse:27.93794	test-rmse:594.33781
[4]	train-rmse:18.52827	test-rmse:597.11881
[5]	train-rmse:15.76184	test-rmse:597.56308
[6]	train-rmse:13.52992	test-rmse:597.40967
[7]	train-rmse:10.31418	test-rmse:596.67337
[8]	train-rmse:9.66591	test-rmse:596.74222
[9]	train-rmse:8.30679	test-rmse:596.69133
[10]	train-rmse:7.11336	test-rmse:597.98079
[11]	train-rmse:4.92182	test-rmse:597.47689
[12]	train-rmse:4.31951	test-rmse:596.61404
[13]	train-rmse:4.01964	test-rmse:596.87693
[14]	train-rmse:3.80004	test-rmse:596.86808
[15]	train-rmse:3.48637	test-rmse:597.21718
[16]	train-rmse:3.29071	test-rmse:596.98130
[17]	train-rmse:3.04450	test-rmse:596.96556
[18]	train-rmse:2.89239	test-rmse:597.04921
[19]	train-rmse:2.70107	test-rmse:597.04948
[20]	train-rmse:2.55136	test-rmse:597.04933
[21]	train-rmse:2.37817	test-rmse:596.86476
[22]	train-rmse:2.27732	test-rmse:596.82348
[23]	train-rmse:2.23002	test-rmse:596.75120
[24]	train-rmse:2.13802	test-rmse:596.83475
[25]	train-rmse:2.06868	test-rmse:596.73689
[26]	train-rmse:1.87541	test-rmse:596.94704
[27]	train-rmse:1.78322	test-rmse:596.74861
[28]	train-rmse:1.72632	test-rmse:596.86226
[29]	train-rmse:1.63567	test-rmse:596.83061
```

```
# 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:241.40400	test-rmse:468.14798
[1]	train-rmse:98.18515	test-rmse:598.60394
[2]	train-rmse:70.43796	test-rmse:613.93280
[3]	train-rmse:27.93794	test-rmse:594.33781
[4]	train-rmse:18.52827	test-rmse:597.11881
[5]	train-rmse:15.76184	test-rmse:597.56308
[6]	train-rmse:13.52992	test-rmse:597.40967
```

```
[7]	train-rmse:10.31418	test-rmse:596.67337
[8]	train-rmse:9.66591	test-rmse:596.74222
[9]	train-rmse:8.30679	test-rmse:596.69133
[10]	train-rmse:7.11336	test-rmse:597.98079
[11]	train-rmse:4.92182	test-rmse:597.47689
[12]	train-rmse:4.31951	test-rmse:596.61404
[13]	train-rmse:4.01964	test-rmse:596.87693
[14]	train-rmse:3.80004	test-rmse:596.86808
[15]	train-rmse:3.48637	test-rmse:597.21718
[16]	train-rmse:3.29071	test-rmse:596.98130
[17]	train-rmse:3.04450	test-rmse:596.96556
[18]	train-rmse:2.89239	test-rmse:597.04921
[19]	train-rmse:2.70107	test-rmse:597.04948
[20]	train-rmse:2.55136	test-rmse:597.04933
[21]	train-rmse:2.37817	test-rmse:596.86476
[22]	train-rmse:2.27732	test-rmse:596.82348
[23]	train-rmse:2.23002	test-rmse:596.75120
[24]	train-rmse:2.13802	test-rmse:596.83475
[25]	train-rmse:2.06868	test-rmse:596.73689
[26]	train-rmse:1.87541	test-rmse:596.94704
[27]	train-rmse:1.78322	test-rmse:596.74861
[28]	train-rmse:1.72632	test-rmse:596.86226
[29]	train-rmse:1.63567	test-rmse:596.83061
```

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:6774.97495	test-rmse:6412.99024
[1]	train-rmse:5770.09905	test-rmse:6457.11577
[2]	train-rmse:5015.44845	test-rmse:6499.64560
[3]	train-rmse:4239.47156	test-rmse:6661.99689
[4]	train-rmse:3505.53341	test-rmse:6949.86112
[5]	train-rmse:3005.50509	test-rmse:7060.45223
[6]	train-rmse:2704.06248	test-rmse:7150.26573
[7]	train-rmse:2487.43675	test-rmse:7210.79102
[8]	train-rmse:2251.59427	test-rmse:7240.25033
[9]	train-rmse:2082.37840	test-rmse:7241.37554
[10]	train-rmse:1967.15108	test-rmse:7230.81001
[11]	train-rmse:1734.87011	test-rmse:7363.53215
[12]	train-rmse:1666.30993	test-rmse:7407.48252
[13]	train-rmse:1449.09812	test-rmse:7500.17073
[14]	train-rmse:1318.81378	test-rmse:7462.01234
[15]	train-rmse:1219.04957	test-rmse:7533.92873
[16]	train-rmse:1085.66512	test-rmse:7536.15376
[17]	train-rmse:926.50254	test-rmse:7593.12216
[18]	train-rmse:819.67148	test-rmse:7655.21627
[19]	train-rmse:719.95313	test-rmse:7647.35010
[20]	train-rmse:649.38902	test-rmse:7629.97257
[21]	train-rmse:559.19661	test-rmse:7623.78415
[22]	train-rmse:537.70667	test-rmse:7621.62974
[23]	train-rmse:462.07056	test-rmse:7610.48868
[24]	train-rmse:443.26464	test-rmse:7614.62903
[25]	train-rmse:384.78671	test-rmse:7627.31332
[26]	train-rmse:337.20520	test-rmse:7622.59160
[27]	train-rmse:308.66206	test-rmse:7629.13639
[28]	train-rmse:273.01383	test-rmse:7636.48918
[29]	train-rmse:237.97673	test-rmse:7635.30295
```

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:3285.90197   test-rmse:1972.13232
[1]     train-rmse:2491.86203   test-rmse:1350.86454
[2]     train-rmse:1890.40368   test-rmse:882.46196
[3]     train-rmse:1434.69235   test-rmse:530.50741
[4]     train-rmse:1089.23724   test-rmse:271.13085
[5]     train-rmse:827.19850    test-rmse:110.95618
[6]     train-rmse:628.42406    test-rmse:145.68109
[7]     train-rmse:477.61786    test-rmse:244.78122
[8]     train-rmse:363.20811    test-rmse:327.83422
[9]     train-rmse:276.41784    test-rmse:392.57120
[10]    train-rmse:210.56042    test-rmse:442.21873
[11]    train-rmse:160.62375    test-rmse:480.15992
[12]    train-rmse:122.74358    test-rmse:509.01315
[13]    train-rmse:94.08714     test-rmse:530.99817
[14]    train-rmse:72.49401     test-rmse:547.73121
[15]    train-rmse:56.29042     test-rmse:560.44855
[16]    train-rmse:44.09137     test-rmse:571.05711
[17]    train-rmse:34.56327     test-rmse:578.63850
[18]    train-rmse:27.08168     test-rmse:585.16928
[19]    train-rmse:21.29454     test-rmse:590.21965
[20]    train-rmse:16.81378     test-rmse:594.13136
[21]    train-rmse:13.31715     test-rmse:597.16654
[22]    train-rmse:10.59158     test-rmse:599.34566
[23]    train-rmse:8.36481      test-rmse:601.10263
[24]    train-rmse:6.71144      test-rmse:602.56967
[25]    train-rmse:5.35205      test-rmse:603.70925
[26]    train-rmse:4.32974      test-rmse:604.52702
[27]    train-rmse:3.52537      test-rmse:605.18638
[28]    train-rmse:2.95408      test-rmse:605.73921
[29]    train-rmse:2.37729      test-rmse:606.17067
[30]    train-rmse:1.96153      test-rmse:606.51057
[31]    train-rmse:1.65902      test-rmse:606.76755
[32]    train-rmse:1.40335      test-rmse:606.94973
[33]    train-rmse:1.19908      test-rmse:607.11337
[34]    train-rmse:1.00331      test-rmse:607.22956
[35]    train-rmse:0.87806      test-rmse:607.32988
```

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: (-0.34682405277278616, 'bad'),
PBIAS: (95.33041309754255, 'bad'),
KGE: (-0.16342687661455702, 'good')

Root Mean Squared Error: 7181.3170116709125
Normalized Root Mean Squared Error: 0.29682222913412054
```

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: (-0.5221262784296814, 'bad'),
PBIAS: (87.55809769151664, 'bad'),
KGE: (-0.16388927514051255, 'good')

Root Mean Squared Error: 7634.384140509057
Normalized Root Mean Squared Error: 0.3155486542328287
```

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: 6449.829297729834
Normalized Root Mean Squared Error: 0.2665879679974305
```

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: 6360.4231309725865
Normalized Root Mean Squared Error: 0.26289258208533467
```

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: (-2.5417735835004196, 'bad'),
PBIAS: (640.3145618721967, 'bad'),
KGE: (-5.488498017135118, 'bad')

Root Mean Squared Error: 5179.166317292788
Normalized Root Mean Squared Error: 0.26077067203528465
```

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: (0.9529564714433267, 'good'),
PBIAS: (16.684295860631067, 'bad'),
KGE: (0.7280603314656273, 'good')

Root Mean Squared Error: 596.8970121341724
Normalized Root Mean Squared Error: 0.030053723988428198
```

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: (0.9512975997533456, 'good'),
PBIAS: (19.473279272703255, 'bad'),
KGE: (0.7073986846553662, 'good')

Root Mean Squared Error: 607.3298725710008
Normalized Root Mean Squared Error: 0.030579017802275857
```

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: 4673.109335864129
Normalized Root Mean Squared Error: 0.2352907374182634
```

Naive Model 2

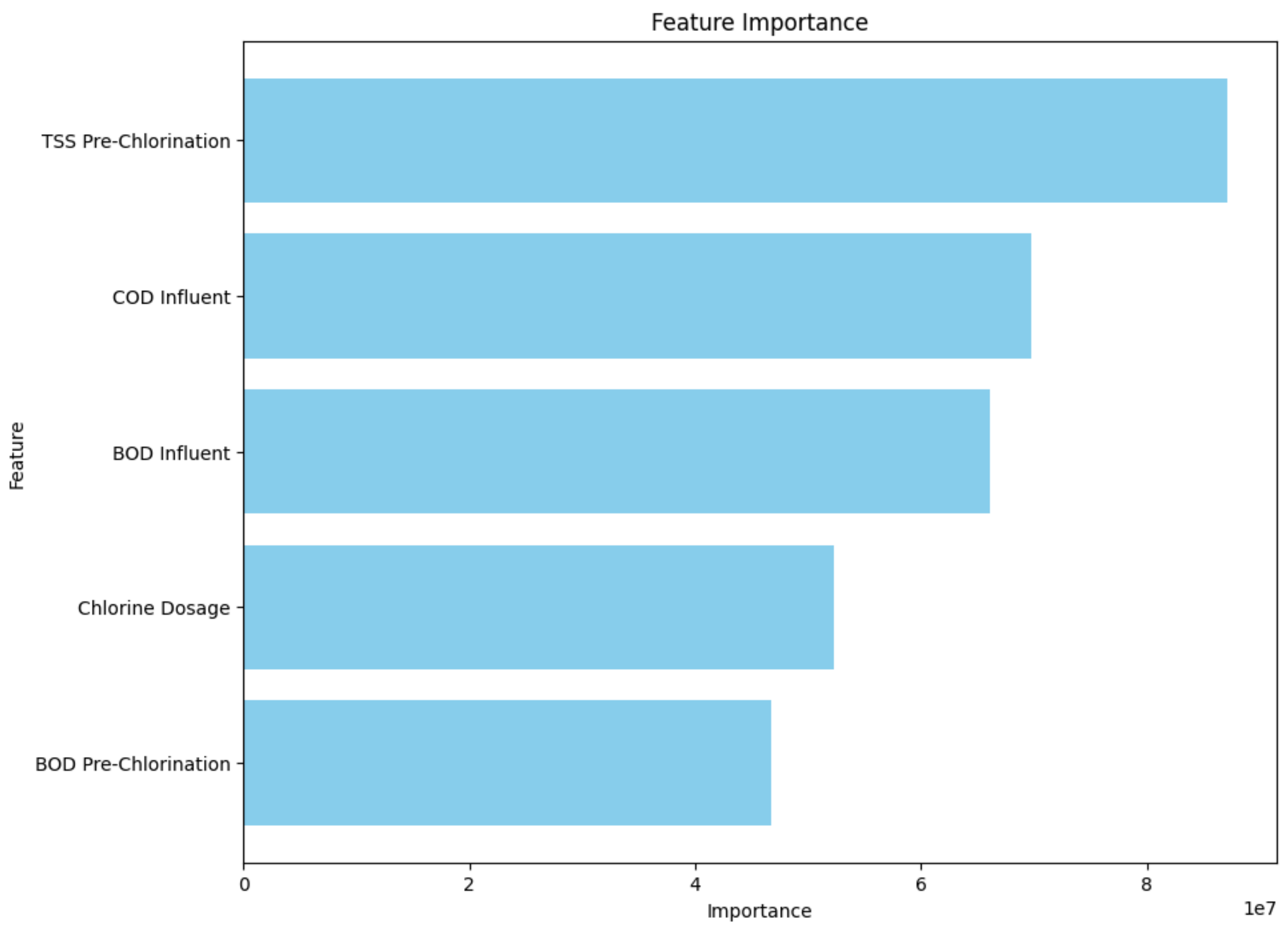
```
rmse = mean_squared_error(y_test_MBBR_dropped, y_pred_naive_orig_MBBR[non_imputed_mask_MBBR],squared=False)
print(f"Root Mean Squared Error: {rmse}")

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: 2794.705983743531
Normalized Root Mean Squared Error: 0.14071325631859075
```

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()
```

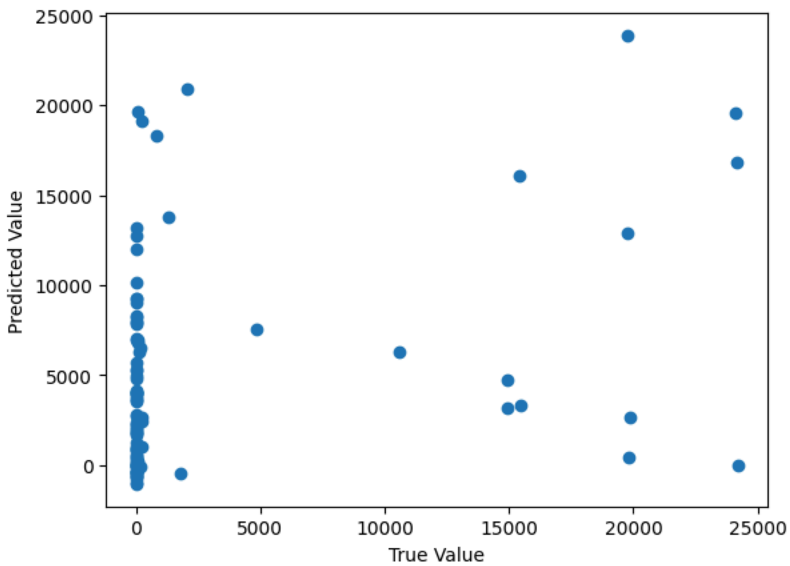


Data Visualization for Model Evaluation

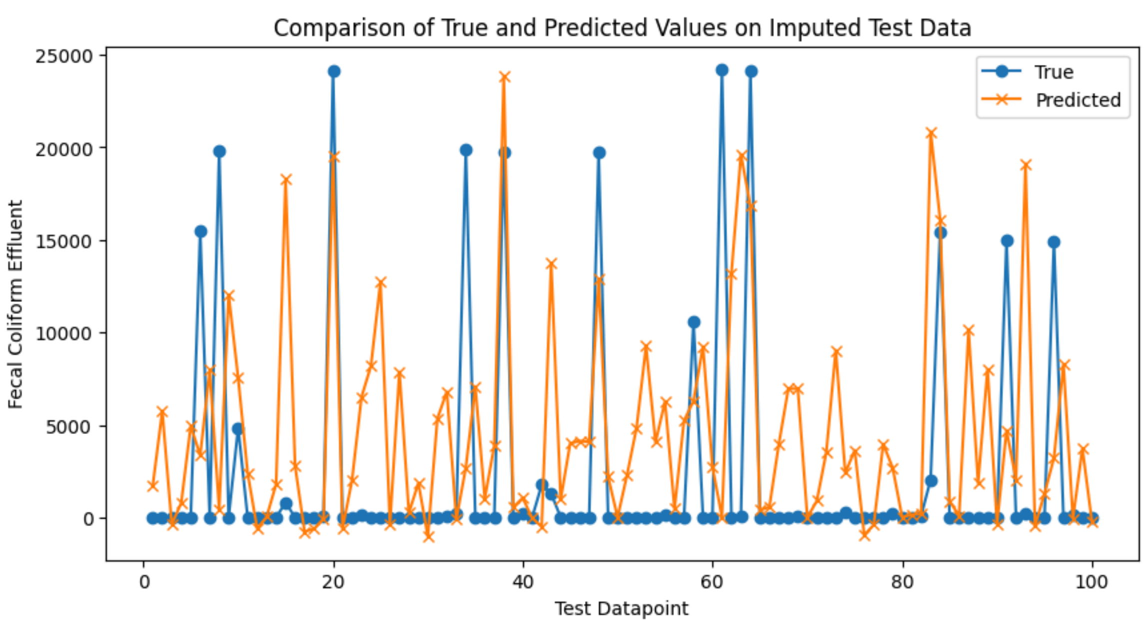
Optimized XGBoost on Imputed Test Dataset

```
# with imputation
plt.scatter(y_test_MBBR,y_pred_final_MBBR);
```

```
plt.xlabel('True Value');
plt.ylabel('Predicted Value');
```



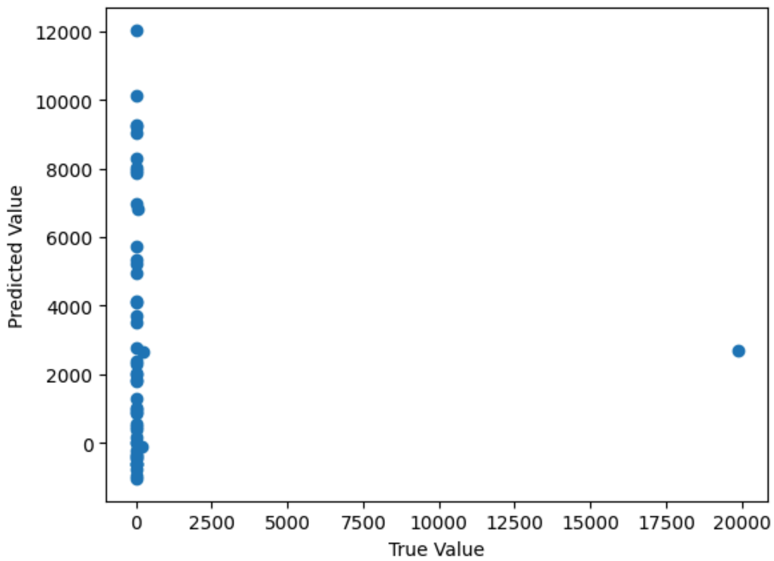

```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_MBBR) + 1)

# 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('Fecal Coliform Effluent')
plt.title('Comparison of True and Predicted Values on Imputed Test Data')
plt.legend()

# Show plot
plt.show()
```



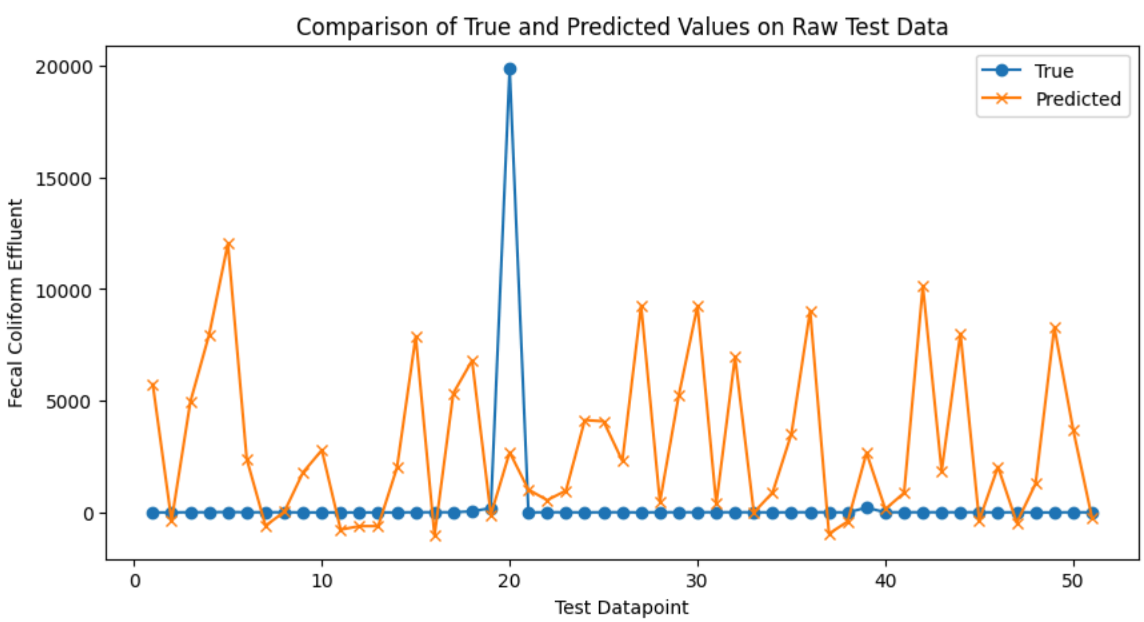
Optimized XGBoost on Non-Imputed (Raw) Test Dataset

```
# 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('Fecal 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)
```