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

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

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

```
Cloning into 'thesis'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (27/27), 311.32 KiB | 7.24 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)
```

CAS

Importing CAS and Splitting

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

```
X_orig_CAS = CAS.drop(columns='Total Coliform Effluent (MPN/100mL)')
y_orig_CAS = CAS['Total Coliform Effluent (MPN/100mL)']
X_train_orig_CAS, X_test_orig_CAS, y_train_orig_CAS, y_test_orig_CAS = train_test_split(X_orig_CAS,
                                                                                         y_orig_CAS,
                                                                                         test_size = 0.3,
                                                                                         random_state=808)
```

```
df_train_orig_CAS = pd.concat([X_train_orig_CAS,y_train_orig_CAS], axis=1)
df_test_orig_CAS = pd.concat([X_test_orig_CAS,y_test_orig_CAS], axis=1)
```

Data Analysis for Raw Dataset

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

```
pd.options.display.float_format = '{:,.2f}'.format
CAS_transposed = CAS.describe().T
CAS_transposed['Missingness Rate'] = missing_rate_CAS
```

```
CAS_transposed
```

	count	mean	std	min	25%	50%	75%	max	Missingness Rate
Flow Rate Influent (m3/d)	289.00	10,366.21	4,355.40	172.00	7,567.00	10,852.00	14,358.00	18,291.00	2.6
Total Coliform Influent (MPN/100mL)	230.00	187,227,782.61	768,962,279.86	490,000.00	13,250,000.00	29,000,000.00	67,750,000.00	10,000,000,000.00	22.5
Total Coliform Effluent (MPN/100mL)	296.00	79,275.50	1,336,778.85	1.00	2.00	10.00	20.00	23,000,000.00	0.3
Fecal Coliform Influent (MPN/100mL)	112.00	84,843,125.00	259,408,932.49	330,000.00	7,800,000.00	13,000,000.00	30,500,000.00	1,700,000,000.00	62.2
Fecal Coliform Effluent (MPN/100mL)	179.00	44,660.29	590,441.16	2.00	10.00	10.00	10.00	7,900,000.00	39.7
BOD Influent \n(ppm)	232.00	94.02	67.76	7.00	50.00	78.00	115.50	456.00	21.8
BOD Pre-	241.00	6.25	6.04	1.00	2.00	5.00	8.00	50.00	18.8

Data Analysis for Training Set (Pre-Imputation)

```
missing_rate_train_orig_CAS = [(df_train_orig_CAS.isnull().sum()[val]/df_train_orig_CAS.shape[0])*100 for val in range(0,df_trai
```

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

```
df_train_orig_CAS_transposed
```

	count	mean	std	min	25%	50%	75%	max	Missingness Rate
Flow Rate Influent (m3/d)	204.00	10,378.50	4,436.49	781.00	7,487.50	10,736.50	14,393.75	18,011.00	1.4
Total Coliform Influent (MPN/100mL)	163.00	147,519,631.90	425,607,781.81	1,000,000.00	13,000,000.00	26,000,000.00	69,000,000.00	3,100,000,000.00	21.2
Total Coliform Effluent (MPN/100mL)	206.00	112,871.10	1,602,407.01	1.00	2.00	10.00	19.25	23,000,000.00	0.4
Fecal Coliform Influent (MPN/100mL)	82.00	67,232,926.83	213,302,904.11	1,700,000.00	6,475,000.00	12,500,000.00	26,900,000.00	1,500,000,000.00	60.3
Fecal Coliform Effluent (MPN/100mL)	125.00	63,732.38	706,556.25	2.00	8.00	10.00	10.00	7,900,000.00	39.6
BOD Influent \n(ppm)	166.00	88.54	64.17	7.00	39.00	74.50	111.00	456.00	19.8
BOD Pre-	171.00	6.15	6.14	1.00	2.00	5.00	8.00	50.00	17.[illegible]

Data Analysis for Testing Set (Pre-imputation)

```
missing_rate_test_orig_CAS = [(df_test_orig_CAS.isnull().sum()[val]/df_test_orig_CAS.shape[0])*100 for val in range(0,df_test_or
```

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

```
df_test_orig_CAS_transposed
```

	count	mean	std	min	25%	50%	75%	max	Missing
Flow Rate Influent (m3/d)	8.500000e+01	1.033672e+04	4.179829e+03	1.720000e+02	7.765000e+03	1.085600e+04	1.363500e+04	1.829100e+04	5.555556
Total Coliform Influent (MPN/100mL)	6.700000e+01	2.838312e+08	1.262400e+09	4.900000e+05	2.100000e+07	3.300000e+07	5.550000e+07	1.000000e+10	2.555556
Total Coliform Effluent (MPN/100mL)	9.000000e+01	2.378911e+03	1.234018e+04	1.000000e+00	2.000000e+00	1.000000e+01	2.000000e+01	1.100000e+05	0.000000
Fecal Coliform Influent (MPN/100mL)	3.000000e+01	1.329777e+08	3.566722e+08	3.300000e+05	1.200000e+07	1.550000e+07	3.300000e+07	1.700000e+09	6.666667
Fecal Coliform Effluent (MPN/100mL)	5.400000e+01	5.119259e+02	2.306060e+03	2.000000e+00	1.000000e+01	1.000000e+01	1.000000e+01	1.553100e+04	4.000000
BOD Influent \n(ppm)	6.600000e+01	1.078030e+02	7.478651e+01	2.200000e+01	5.750000e+01	8.400000e+01	1.270000e+02	3.490000e+02	2.666667
BOD Pre-	7.000000e+01	6.514286e+00	5.810393e+00	1.000000e+00	2.000000e+00	5.000000e+00	8.750000e+00	3.100000e+01	2.222222

Data Imputation

Exporting Datasets to R

```
df_train_orig_CAS.to_csv('CAS_train_set.csv',index=False)
df_test_orig_CAS.to_csv('CAS_test_set.csv',index=False)

# Export to R for mixgb
```

Mixgb imputation

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

CAS_train_set <- read.csv("C:/Users/nikko/PycharmProjects/Thesis/CAS_train_set.csv")
CAS_test_set <- read.csv("C:/Users/nikko/PycharmProjects/Thesis/CAS_test_set.csv")

CAS_train_set_df = as.data.frame(CAS_train_set)
CAS_test_set_df = as.data.frame(CAS_test_set)

clean_CAS_train_set_df <- data_clean(CAS_train_set_df)
clean_CAS_test_set_df <- data_clean(CAS_test_set_df)

cv.results_2 <- mixgb_cv(data = clean_CAS_train_set_df, nrounds = 5000, verbose = FALSE)
cv.results_2$evaluation.log
cv.results_2$best.nrounds

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

CAS_test.imputed <- impute_new(object = mixgb_obj, newdata = clean_CAS_test_set_df)

write.xlsx(CAS_train_imputed[[1]], file = 'cas_m1_imputed_train.xlsx')
write.xlsx(CAS_train_imputed[[2]], file = 'cas_m2_imputed_train.xlsx')
write.xlsx(CAS_train_imputed[[3]], file = 'cas_m3_imputed_train.xlsx')
write.xlsx(CAS_train_imputed[[4]], file = 'cas_m4_imputed_train.xlsx')
write.xlsx(CAS_train_imputed[[5]], file = 'cas_m5_imputed_train.xlsx')

write.xlsx(CAS_test.imputed[[1]], file = 'cas_m1_imputed_test.xlsx')
write.xlsx(CAS_test.imputed[[2]], file = 'cas_m2_imputed_test.xlsx')
write.xlsx(CAS_test.imputed[[3]], file = 'cas_m3_imputed_test.xlsx')
write.xlsx(CAS_test.imputed[[4]], file = 'cas_m4_imputed_test.xlsx')
write.xlsx(CAS_test.imputed[[5]], file = 'cas_m5_imputed_test.xlsx')
```

Import imputed datasets from R

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

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

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

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

Data Analysis for Training Set (Post-Imputation)

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

```python
average_CAS_train_transposed
```

	count	mean	std	min	25%	50%	75%	
Flow.Rate.Influent..m3.d.	2.070000e+02	1.036842e+04	4.408625e+03	7.810000e+02	7.560500e+03	1.062100e+04	1.437400e+04	1.80110
Total.Coliform.Influent..MPN.100mL.	2.070000e+02	1.486484e+08	4.185196e+08	1.000000e+06	1.600000e+07	2.900000e+07	9.100000e+07	3.10000
Total.Coliform.Effluent..MPN.100mL.	2.070000e+02	1.123286e+05	1.598532e+06	1.000000e+00	2.000000e+00	1.000000e+01	2.000000e+01	2.30000
Fecal.Coliform.Influent..MPN.100mL.	2.070000e+02	5.388222e+07	1.506356e+08	1.700000e+06	8.610000e+06	1.760000e+07	3.500000e+07	1.50000
Fecal.Coliform.Effluent..MPN.100mL.	2.070000e+02	3.863027e+04	5.490612e+05	2.000000e+00	8.400000e+00	1.000000e+01	1.390000e+01	7.90000
BOD.Influent...ppm.	2.070000e+02	8.926184e+01	6.016089e+01	7.000000e+00	4.910000e+01	7.800000e+01	1.130000e+02	4.56000
BOD.Pre.chlorination..ppm.	2.070000e+02	6.439614e+00	5.783688e+00	1.000000e+00	2.000000e+00	5.800000e+00	8.600000e+00	5.90000
COD.Influent..ppm.	2.070000e+02	1.577382e+02	9.181686e+01	1.300000e+01	9.450000e+01	1.420000e+02	2.110000e+02	5.73000
COD.Pre.chlorination..ppm.	2.070000e+02	2.029855e+01	2.117572e+01	2.000000e+00	1.150000e+01	1.760000e+01	2.400000e+01	2.65000
TSS.Pre.chlorination..ppm.	2.070000e+02	6.443478e+00	1.447618e+01	1.000000e+00	1.000000e+00	4.000000e+00	7.000000e+00	1.90000
pH.Pre.chlorination	2.070000e+02	7.149420e+00	3.365641e-01	6.500000e+00	6.940000e+00	7.120000e+00	7.300000e+00	8.65000
Chlorine.Dosage..L.d.	2.070000e+02	1.647594e+02	1.032712e+02	3.000000e+00	1.026000e+02	1.360000e+02	1.867000e+02	8.40000
Residual.chlorine..ppm.	2.070000e+02	2.346387e+00	1.836642e+00	0.000000e+00	6.015000e-01	1.939000e+00	4.107000e+00	5.33900

Data Analysis for Testing Set (Post-Imputation)

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

```
average_CAS_test_transposed
```

	count	mean	std	min	25%	50%	75%	
Flow.Rate.Influent..m3.d.	90.00	10,369.95	4,157.57	172.00	7,821.25	10,958.50	14,130.00	18,291
Total.Coliform.Influent..MPN.100mL.	90.00	235,817,666.67	1,090,795,688.88	490,000.00	23,000,000.00	44,340,000.00	97,700,000.00	10,000,000,00
Total.Coliform.Effluent..MPN.100mL.	90.00	2,378.91	12,340.18	1.00	2.00	10.00	20.00	110,00
Fecal.Coliform.Influent..MPN.100mL.	90.00	65,077,666.67	210,186,867.66	330,000.00	13,000,000.00	22,980,000.00	36,335,000.00	1,700,000,00
Fecal.Coliform.Effluent..MPN.100mL.	90.00	640.66	2,432.92	2.00	8.40	10.00	15.20	15,531
BOD.Influent...ppm.	90.00	100.93	67.98	21.40	56.25	79.00	123.45	34
BOD.Pre.chlorination..ppm.	90.00	6.88	5.40	1.00	3.00	6.00	9.00	31
COD.Influent..ppm.	90.00	186.36	151.64	35.00	102.00	147.50	238.25	1,23
COD.Pre.chlorination..ppm.	90.00	20.40	11.86	3.00	13.00	17.50	26.30	6
TSS.Pre.chlorination..ppm.	90.00	6.06	6.19	1.00	2.00	3.00	8.00	3
pH.Pre.chlorination	90.00	7.08	0.29	6.25	6.90	7.09	7.27	
Chlorine.Dosage..L.d.	90.00	180.33	97.55	3.00	110.30	156.10	216.35	57
Residual.chlorine..ppm.	90.00	2.48	1.88	0.01	0.58	2.19	4.74	

Exhaustive Feature Selection

For Imputed Dataset

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

```
X_train_CAS = average_CAS_train.drop(columns=['Residual.chlorine..ppm.','Total.Coliform.Effluent..MPN.100mL.','Fecal.Coliform.Ef
y_train_CAS = average_CAS_train['Total.Coliform.Effluent..MPN.100mL.']
X_test_CAS = average_CAS_test.drop(columns=['Residual.chlorine..ppm.','Total.Coliform.Effluent..MPN.100mL.','Fecal.Coliform.Effl
y_test_CAS = average_CAS_test['Total.Coliform.Effluent..MPN.100mL.']
```

```
features_wo_chlorine_dosage = X_train_CAS.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_CAS.columns[-1],) + combo for combo in combinations]


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

results = []
for combo in combinations:
    dtrain = xgb.DMatrix(X_train_CAS[list(combo)], label=y_train_CAS)
    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_CAS = pd.DataFrame(results, columns=['Combination', 'Train RMSE', 'Validation RMSE', 'Train RMSE Std. Dev.', ' Valida
```

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

	Combination	Train RMSE	Validation RMSE	Train RMSE Std. Dev.	Validation RMSE Std. Dev
323	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280011.047738	7.228241e+05	139710.616816	1.433136e+06
414	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280011.142987	7.228547e+05	139710.969639	1.433121e+06
284	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280016.513652	7.229206e+05	139701.212847	1.433553e+06
434	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280010.378140	7.229333e+05	139711.871439	1.433082e+06
167	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280073.563225	7.229333e+05	139588.264363	1.433290e+06
...	...	...	...	...	...
354	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	280039.984622	2.355080e+06	139666.166552	1.925991e+06
61	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280099.422890	2.355161e+06	139544.501879	1.926474e+06
116	(Chlorine.Dosage..L.d., BOD.Influent...ppm., C...	280069.209328	2.355169e+06	139608.399453	1.926328e+06
207	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	280068.782687	2.355171e+06	139608.520929	1.926060e+06
219	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	280055.577319	2.355208e+06	139635.695689	1.926157e+06

511 rows × 5 columns

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

	Combination	Train RMSE	Validation RMSE	Train RMSE Std. Dev.	Validation RMSE Std. Dev
323	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280011.047738	722824.115553	139710.616816	1.433136e+06
414	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280011.142987	722854.706969	139710.969639	1.433121e+06
284	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	280016.513652	722920.574878	139701.212847	1.433553e+06

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

```
('Chlorine.Dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'BOD.Pre.chlorination..ppm.',
 'COD.Pre.chlorination..ppm.',
 'TSS.Pre.chlorination..ppm.',
 'pH.Pre.chlorination')
```

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

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

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

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

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

```
('Chlorine.Dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'BOD.Pre.chlorination..ppm.',
 'COD.Pre.chlorination..ppm.',
 'TSS.Pre.chlorination..ppm.',
 'pH.Pre.chlorination')
```

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

For Raw Dataset

```
non_imputed_mask_CAS_train = ~np.isnan(y_train_orig_CAS)
non_imputed_mask_CAS_test = ~np.isnan(y_test_orig_CAS)
```

```
X_train_orig_CAS.head()
```

	Flow Rate Influent (m3/d)	Total Coliform Influent (MPN/100mL)	Fecal Coliform Influent (MPN/100mL)	Fecal Coliform Effluent (MPN/100mL)	BOD Influent \n(ppm)	BOD Pre-chlorination\n(ppm)	COD Influent (ppm)	COD Pre-chlorination\n(ppm)	TSS Pr chlorinati (ppi
174	6253.0	60000000.0	90900000.0	95.0	155.0	7.0	441.0	35.0	1
70	8019.0	50000000.0	NaN	NaN	55.0	5.0	211.0	10.0	8
179	17590.0	20000000.0	15000000.0	10.0	52.0	3.0	207.0	33.0	6
252	14848.0	23000000.0	7900000.0	2.0	58.0	2.0	100.0	5.0	1

```
X_train_CAS_dropped = X_train_orig_CAS[non_imputed_mask_CAS_train]
y_train_CAS_dropped = y_train_orig_CAS[non_imputed_mask_CAS_train]
X_test_CAS_dropped = X_test_orig_CAS[non_imputed_mask_CAS_test]
y_test_CAS_dropped = y_test_orig_CAS[non_imputed_mask_CAS_test]

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

```
Index(['Flow Rate Influent (m3/d)', 'Total Coliform Influent (MPN/100mL)',
       'Fecal Coliform Influent (MPN/100mL)',
       'Fecal Coliform Effluent (MPN/100mL)', 'BOD Influent \n(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_CAS_dropped.columns[-1],) + combo for combo in combinations]


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

    results = []
    for combo in combinations:
        dtrain = xgb.DMatrix(X_train_CAS_dropped[list(combo)], label=y_train_CAS_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_CAS_dropped = pd.DataFrame(results, columns=['Combination', 'Train RMSE', 'Validation RMSE', 'Train RMSE Std. Dev.',


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

	Combination	Train RMSE	Validation RMSE	Train RMSE Std. Dev.	Validation RMSE Std. Dev
701	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280643.557667	7.215556e+05	140012.576333	1.433474e+06
1079	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280642.184227	7.215606e+05	140015.323202	1.433471e+06
296	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280643.474975	7.215721e+05	140012.914102	1.433506e+06
595	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280640.513043	7.215748e+05	140018.711156	1.433464e+06
590	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280641.192490	7.215772e+05	140017.479053	1.433504e+06
...	...	...	...	...	...
876	(Chlorine Dosage (L/d), Total Coliform Influen...	280656.883197	2.289559e+06	139998.557615	1.580999e+06
893	(Chlorine Dosage (L/d), Total Coliform Influen...	280630.728531	2.289593e+06	140051.476551	1.580910e+06
1777	(Chlorine Dosage (L/d), Total Coliform Influen...	280621.643446	2.289595e+06	140069.288047	1.580911e+06
1400	(Chlorine Dosage (L/d), Total Coliform Influen...	280620.211262	2.289604e+06	140072.511002	1.580919e+06
1391	(Chlorine Dosage (L/d), Total Coliform Influen...	280631.400729	2.289610e+06	140049.773563	1.580924e+06

2047 rows × 5 columns

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

	Combination	Train RMSE	Validation RMSE	Train RMSE Std. Dev.	Validation RMSE Std. Dev
701	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280643.557667	721555.592443	140012.576333	1.433474e+06
1079	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280642.184227	721560.578280	140015.323202	1.433471e+06
296	(Chlorine Dosage (L/d), Flow Rate Influent (m3...	280643.474975	721572.051024	140012.914102	1.433506e+06

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

```
('Chlorine Dosage (L/d)',
 'Flow Rate Influent (m3/d)',
 'Fecal Coliform Effluent (MPN/100mL)',
 'BOD Influent \n(ppm)',
 'BOD Pre-chlorination\n(ppm)',
 'COD Influent (ppm)')
```

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

```
('Chlorine Dosage (L/d)',
 'Flow Rate Influent (m3/d)',
 'Total Coliform Influent (MPN/100mL)',
 'Fecal Coliform Effluent (MPN/100mL)',
 'BOD Influent \n(ppm)',
 'BOD Pre-chlorination\n(ppm)',
 'COD Influent (ppm)')
```

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

```
('Chlorine Dosage (L/d)',
 'Flow Rate Influent (m3/d)',
 'Fecal Coliform Effluent (MPN/100mL)',
```

```
 'BOD Influent \n(ppm)',
 'COD Influent (ppm)')
```

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

```
('Chlorine Dosage (L/d)',
 'Flow Rate Influent (m3/d)',
 'Fecal Coliform Effluent (MPN/100mL)',
 'BOD Influent \n(ppm)',
 'BOD Pre-chlorination\n(ppm)',
 'COD Influent (ppm)')
```

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

Hyperparameter Optimization

```
Start coding or generate with AI.
```

For Imputed Dataset

```
# Convert the data into DMatrix format
dtrain = xgb.DMatrix(X_train_CAS[list(optimal_features_CAS)], label=y_train_CAS)

# 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_CAS = 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_CAS.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         | -8.298e+0 | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -8.279e+0 | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -8.726e+0 | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -8.307e+0 | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
| 5         | -1.194e+0 | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -8.299e+0 | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -8.327e+0 | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -7.941e+0 | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -8.304e+0 | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -1.02e+06 | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -8.571e+0 | -5.294    | 0.6658    | -1.215    | -5.342    | -0.3629   | 2.421     | 0.9975    |
| 12        | -8.297e+0 | -4.018    | 0.5023    | -2.865    | -0.7772   | -2.167    | 2.849     | 0.9048    |
```

```
| 13        | -8.551e+0 | -5.736    | 0.7323    | -1.385    | -3.443    | -1.267    | 1.381     | 0.8479    |
| 14        | -8.271e+0 | -4.549    | 0.6199    | -1.97     | -2.466    | -1.327    | 1.587     | 0.5003    |
| 15        | -8.513e+0 | -0.4235   | 0.8076    | -1.559    | -0.8907   | -3.059    | 1.962     | 0.5891    |
| 16        | -1.385e+0 | -0.4211   | 0.718     | -0.3578   | 0.6482    | -3.043    | 2.168     | 0.9886    |
| 17        | -8.125e+0 | -1.045    | 0.8055    | -0.8125   | -1.274    | -3.58     | 2.138     | 0.5       |
| 18        | -7.234e+0 | -0.8188   | 0.7891    | 0.0       | -1.113    | -3.898    | 2.692     | 0.5       |
| 19        | -1.306e+0 | -0.5335   | 0.8491    | -0.544    | -1.374    | -4.207    | 2.918     | 0.8932    |
| 20        | -7.528e+0 | -0.9643   | 0.7668    | -0.1575   | -0.9436   | -3.681    | 2.4       | 0.5       |
| 21        | -8.291e+0 | -3.597    | 0.3809    | -2.441    | -3.707    | -4.822    | 2.478     | 0.5501    |
| 22        | -7.236e+0 | -1.161    | 0.6781    | 0.0       | -0.7441   | -3.728    | 2.913     | 0.5       |
| 23        | -8.725e+0 | -1.135    | 0.4966    | -1.187    | -1.414    | -5.401    | 2.736     | 0.6107    |
| 24        | -8.308e+0 | -4.182    | 0.9605    | -2.47     | -2.19     | -5.291    | 1.743     | 0.6425    |
| 25        | -7.236e+0 | -1.256    | 0.7226    | 0.0       | -1.309    | -3.439    | 2.808     | 0.5       |
| 26        | -7.219e+0 | -1.558    | 1.0       | 0.0       | -1.094    | -3.95     | 2.607     | 0.5       |
| 27        | -1.057e+0 | -1.426    | 0.3       | 0.0       | -1.185    | -3.945    | 2.532     | 0.5       |
| 28        | -7.254e+0 | -1.374    | 1.0       | -0.1118   | -0.8555   | -3.39     | 2.741     | 0.5       |
| 29        | -9.485e+0 | -4.808    | 0.9896    | -0.4094   | -5.86     | -3.441    | 1.36      | 0.5318    |
| 30        | -8.319e+0 | -2.419    | 0.5783    | -1.985    | -4.593    | -0.2901   | 2.045     | 0.5732    |
| 31        | -8.304e+0 | -5.969    | 0.4113    | -2.329    | 0.1311    | -5.538    | 2.738     | 0.6068    |
| 32        | -8.359e+0 | -0.1682   | 0.5353    | -1.784    | 0.9031    | -3.447    | 1.366     | 0.7435    |
| 33        | -1.706e+0 | -3.113    | 0.9634    | -0.422    | -4.206    | -1.789    | 1.987     | 0.6836    |
| 34        | -7.22e+05 | -0.567    | 1.0       | 0.0       | -1.165    | -2.934    | 2.882     | 0.5       |
| 35        | -7.216e+0 | -0.9678   | 1.0       | 0.0       | -1.643    | -2.754    | 2.041     | 0.5       |
| 36        | -8.026e+0 | -1.695    | 0.6671    | -0.6154   | -1.457    | -2.223    | 2.631     | 0.5033    |
| 37        | -7.965e+0 | -1.436    | 1.0       | -0.7928   | -1.309    | -2.676    | 1.343     | 0.5       |
| 38        | -7.971e+0 | -0.5119   | 1.0       | -0.8024   | -1.952    | -2.215    | 2.294     | 0.5       |
| 39        | -8.223e+0 | -1.511    | 1.0       | -2.018    | -1.694    | -2.337    | 2.119     | 0.5       |
| 40        | -7.212e+0 | -0.1056   | 1.0       | 0.0       | -1.456    | -3.087    | 1.449     | 0.5       |
| 41        | -7.511e+0 | -0.5379   | 1.0       | -0.3388   | -2.359    | -2.891    | 1.039     | 0.5       |
| 42        | -1.056e+0 | -0.3201   | 0.3       | 0.0       | -1.739    | -2.226    | 1.303     | 0.5       |
| 43        | -7.209e+0 | -0.8668   | 1.0       | 0.0       | -1.685    | -3.679    | 1.151     | 0.5       |
| 44        | -8.045e+0 | -0.2363   | 1.0       | -0.9211   | -1.715    | -3.596    | 1.0       | 0.5       |
| 45        | -7.218e+0 | -0.44     | 1.0       | 0.0       | -2.374    | -3.428    | 1.915     | 0.5       |
| 46        | -1.429e+0 | -1.439    | 0.7311    | -0.4135   | -2.984    | -3.786    | 1.011     | 0.5808    |
| 47        | -8.298e+0 | -2.97     | 0.8314    | -2.67     | -1.21     | -2.575    | 1.528     | 0.7009    |
| 48        | -8.241e+0 | -1.425    | 1.0       | -2.616    | -0.6538   | -2.913    | 1.0       | 0.5       |
| 49        | -8.223e+0 | -2.745    | 1.0       | -1.672    | -0.594    | -1.903    | 2.53      | 0.5       |
| 50        | -8.296e+0 | -1.808    | 0.3       | -2.975    | -0.6002   | -1.835    | 2.182     | 0.7743    |
| 51        | -8.299e+0 | -3.35     | 0.4919    | -2.53     | -0.1594   | -1.157    | 1.627     | 0.7204    |
| 52        | -8.293e+0 | -3.033    | 0.4498    | -2.67     | -1.67     | -1.171    | 2.397     | 0.5       |
| 53        | -7.213e+0 | 0.3       | 1.0       | 0.0       | -2.228    | -2.868    | 2.43      | 0.5       |
| 54        | -8.257e+0 | -2.397    | 1.0       | -2.93     | -1.04     | -3.053    | 3.0       | 0.5       |
| 55        | -1.357e+0 | -0.216    | 0.6097    | -0.464    | -3.302    | -2.456    | 2.744     | 0.5828    |
| 56        | -8.261e+0 | -2.868    | 1.0       | -3.0      | 0.4033    | -2.506    | 2.206     | 0.5       |
```

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

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

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

del best_params_CAS['col_subsample']
del best_params_CAS['lambd']
```

```python
best_params_CAS
```

```
{'alpha': 1.9952623149688795,
 'eta': 1.0,
 'gamma': 1e-06,
 'max_depth': 8,
 'subsample': 1.0,
 'lambda': 0.03168506034401033,
 'objective': 'reg:squarederror',
 'eval_metric': 'rmse',
 'colsample_bytree': 1.0}
```

For Raw Dataset

```
# Convert the data into DMatrix format
dtrain = xgb.DMatrix(X_train_CAS_dropped[list(optimal_features_CAS_dropped)], label=y_train_CAS_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_CAS_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_CAS_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	-8.285e+0	0.04075	0.4513	-2.68	-1.662	-1.582	2.026	0.7673
2	-7.796e+0	-4.514	0.7529	-1.843	-2.2	-1.339	1.596	0.5436
3	-8.474e+0	-1.108	0.5069	-1.136	-4.974	-0.5216	2.693	0.8202
4	-8.17e+05	-3.147	0.6275	-2.063	1.604	-0.4504	1.466	0.7294
5	-1.48e+06	-2.356	0.4052	-0.3806	-0.09483	-5.01	1.643	0.6674
6	-8.229e+0	-2.162	0.5228	-2.659	-5.793	-3.144	2.227	0.7522
7	-8.329e+0	-0.1605	0.6002	-1.973	0.8823	-3.597	1.193	0.6362
8	-1.086e+0	-0.9486	0.7394	-0.4943	-0.4982	-3.564	2.166	0.5334
9	-7.963e+0	-1.814	0.8098	-2.344	-2.07	-3.54	1.193	0.8742
10	-9.606e+0	0.06321	0.6188	-0.4746	-0.88	-0.04974	2.478	0.7132
11	-8.011e+0	-2.469	0.7178	-2.679	-3.238	-1.805	1.63	0.757
12	-7.589e+0	-2.503	1.0	-3.0	-0.7462	-1.253	1.0	1.0
13	-7.589e+0	0.3	1.0	-3.0	1.8	-1.115	1.0	1.0
14	-7.596e+0	-6.0	1.0	-3.0	-0.379	0.3	1.0	1.0
15	-7.596e+0	-6.0	1.0	-3.0	-4.432	0.3	1.0	1.0
16	-7.201e+0	-6.0	1.0	0.0	-5.2	0.3	3.0	0.5
17	-7.196e+0	-6.0	1.0	-1.794	-6.0	-2.323	3.0	1.0
18	-7.198e+0	-6.0	1.0	0.0	-6.0	-1.874	1.0	0.5
19	-7.588e+0	-6.0	1.0	-3.0	-6.0	-5.251	1.0	0.5
20	-7.588e+0	0.3	1.0	-3.0	-6.0	-6.0	1.0	1.0
21	-1.239e+0	-6.0	0.3	0.0	-6.0	-5.413	3.0	1.0
22	-7.199e+0	-4.166	1.0	-0.6378	-6.0	0.3	1.0	1.0
23	-7.588e+0	-6.0	1.0	-3.0	-6.0	-2.086	1.0	0.5
24	-7.602e+0	-6.0	1.0	-3.0	-6.0	0.3	3.0	0.5
25	-7.588e+0	-6.0	1.0	-3.0	-3.237	-1.969	3.0	1.0
26	-7.196e+0	-6.0	1.0	0.0	1.8	0.3	3.0	1.0
27	-1.009e+0	-6.0	0.3	0.0	-2.601	0.3	1.0	1.0
28	-7.595e+0	-6.0	1.0	-3.0	1.8	0.3	3.0	1.0
29	-7.602e+0	-4.236	1.0	-3.0	-1.334	0.3	3.0	0.5
30	-7.199e+0	-4.061	1.0	0.0	-6.0	-0.9599	3.0	0.5
31	-8.279e+0	0.07783	0.4317	-2.983	-3.539	-5.949	2.914	0.7015
32	-7.596e+0	0.3	1.0	-3.0	-6.0	0.3	1.0	1.0
33	-7.199e+0	0.3	1.0	0.0	-6.0	-3.505	1.0	0.5
34	-7.199e+0	0.3	1.0	0.0	-6.0	-6.0	3.0	0.5
35	-7.588e+0	0.3	1.0	-3.0	-6.0	-2.892	1.0	1.0
36	-1.432e+0	0.3	0.3	0.0	-6.0	-6.0	1.0	0.5
37	-7.197e+0	-5.384	1.0	-1.149	-5.77	-0.8193	2.084	0.687
38	-7.196e+0	-4.737	1.0	-2.183	-4.679	-1.188	3.0	0.5
39	-7.196e+0	-3.979	1.0	-0.8915	-5.624	-1.831	1.067	0.5
40	-8.247e+0	-4.024	0.3	-2.625	-6.0	-0.5526	1.571	1.0
41	-7.197e+0	0.3	1.0	-0.3115	-6.0	-3.279	3.0	0.5361
42	-7.197e+0	0.3	1.0	-0.4342	-6.0	-1.696	1.0	0.5

```
| 43        | -1.356e+0 | -1.767    | 0.5948    | -0.05913  | -5.684    | -2.688    | 1.039     | 0.9782    |
| 44        | -7.197e+0 | -5.387    | 1.0       | -1.477    | -4.752    | -2.217    | 1.654     | 0.5       |
| 45        | -8.786e+0 | -4.308    | 0.3858    | -0.8986   | -4.76     | 0.1554    | 2.855     | 0.7401    |
| 46        | -7.587e+0 | -4.902    | 1.0       | -3.0      | -5.237    | -2.713    | 2.581     | 0.5       |
| 47        | -8.248e+0 | -6.0      | 0.3       | -2.827    | -4.867    | -1.251    | 2.347     | 0.5       |
| 48        | -7.201e+0 | -5.967    | 1.0       | 0.0       | -6.0      | 0.1885    | 1.0       | 0.5       |
| 49        | -7.197e+0 | -6.0      | 1.0       | 0.0       | -6.0      | -1.517    | 3.0       | 0.5       |
| 50        | -7.197e+0 | 0.3       | 1.0       | -2.305    | -6.0      | -4.337    | 3.0       | 0.5       |
| 51        | -7.198e+0 | -5.594    | 0.9977    | -1.521    | 0.08586   | -0.34     | 2.824     | 0.8313    |
| 52        | -7.198e+0 | -6.0      | 1.0       | -1.202    | 1.762     | 0.3       | 1.092     | 0.5       |
| 53        | -7.197e+0 | -6.0      | 1.0       | -1.21     | 1.8       | -1.672    | 2.47      | 0.5       |
| 54        | -7.588e+0 | -6.0      | 1.0       | -3.0      | 0.01779   | -2.091    | 2.423     | 1.0       |
| 55        | -8.259e+0 | -6.0      | 0.3       | -3.0      | 1.8       | -1.272    | 1.0       | 0.5       |
| 56        | -7.201e+0 | -3.902    | 1.0       | 0.0       | 1.8       | 0.3       | 3.0       | 0.5       |
```

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

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

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

del best_params_CAS_dropped['col_subsample']
del best_params_CAS_dropped['lambd']
```

```
best_params_CAS_dropped
```

```
{'alpha': 1e-06,
 'eta': 1.0,
 'gamma': 0.011294693573416021,
 'max_depth': 8,
 'subsample': 0.5,
 'lambda': 0.18249597207006912,
 '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_CAS[list(optimal_features_CAS)], label=y_train_CAS)
dtest = xgb.DMatrix(X_test_CAS[list(optimal_features_CAS)], label=y_test_CAS)
```

Determination of optimal num_boost_round

```
evals_result_CAS = {}

# Train the final model
final_model_CAS = xgb.train(best_params_CAS, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(dtrain, 'train'),(c
                  evals_result=evals_result_CAS)
```

```
[0]	train-rmse:49004.73752	test-rmse:12154.66071
[1]	train-rmse:1530.12270	test-rmse:12747.69412
[2]	train-rmse:97.27207	test-rmse:12762.25777
[3]	train-rmse:55.00458	test-rmse:12761.97711
[4]	train-rmse:21.11131	test-rmse:12773.49846
[5]	train-rmse:4.42933	test-rmse:12781.71096
```

```
[6]	train-rmse:2.58400	test-rmse:12781.93784
[7]	train-rmse:1.61799	test-rmse:12781.73041
[8]	train-rmse:1.05608	test-rmse:12781.60162
[9]	train-rmse:0.82475	test-rmse:12781.65792
[10]	train-rmse:0.60341	test-rmse:12781.65181
[11]	train-rmse:0.46301	test-rmse:12781.63689
[12]	train-rmse:0.40773	test-rmse:12781.63875
[13]	train-rmse:0.36362	test-rmse:12781.65444
[14]	train-rmse:0.32297	test-rmse:12781.63054
[15]	train-rmse:0.30403	test-rmse:12781.63319
[16]	train-rmse:0.28799	test-rmse:12781.63808
[17]	train-rmse:0.27338	test-rmse:12781.64446
[18]	train-rmse:0.26338	test-rmse:12781.64486
[19]	train-rmse:0.25201	test-rmse:12781.64452
[20]	train-rmse:0.25201	test-rmse:12781.64452
[21]	train-rmse:0.25201	test-rmse:12781.64452
[22]	train-rmse:0.25201	test-rmse:12781.64452
[23]	train-rmse:0.25201	test-rmse:12781.64452
[24]	train-rmse:0.25201	test-rmse:12781.64452
[25]	train-rmse:0.25201	test-rmse:12781.64452
[26]	train-rmse:0.25201	test-rmse:12781.64452
[27]	train-rmse:0.25201	test-rmse:12781.64452
[28]	train-rmse:0.25201	test-rmse:12781.64452
[29]	train-rmse:0.25201	test-rmse:12781.64452
[30]	train-rmse:0.25201	test-rmse:12781.64452
```

```
# Train the final model
final_model_CAS = xgb.train(best_params_CAS, dtrain, num_boost_round=(np.argmin(evals_result_CAS['train']['rmse'])+1), early_stc
                  evals_result=evals_result_CAS)

# Make predictions on the test set
y_pred_final_CAS = final_model_CAS.predict(dtest)
```

```
[0]	train-rmse:49004.73752	test-rmse:12154.66071
[1]	train-rmse:1530.12270	test-rmse:12747.69412
[2]	train-rmse:97.27207	test-rmse:12762.25777
[3]	train-rmse:55.00458	test-rmse:12761.97711
[4]	train-rmse:21.11131	test-rmse:12773.49846
[5]	train-rmse:4.42933	test-rmse:12781.71096
[6]	train-rmse:2.58400	test-rmse:12781.93784
[7]	train-rmse:1.61799	test-rmse:12781.73041
[8]	train-rmse:1.05608	test-rmse:12781.60162
[9]	train-rmse:0.82475	test-rmse:12781.65792
[10]	train-rmse:0.60341	test-rmse:12781.65181
[11]	train-rmse:0.46301	test-rmse:12781.63689
[12]	train-rmse:0.40773	test-rmse:12781.63875
[13]	train-rmse:0.36362	test-rmse:12781.65444
[14]	train-rmse:0.32297	test-rmse:12781.63054
[15]	train-rmse:0.30403	test-rmse:12781.63319
[16]	train-rmse:0.28799	test-rmse:12781.63808
[17]	train-rmse:0.27338	test-rmse:12781.64446
[18]	train-rmse:0.26338	test-rmse:12781.64486
[19]	train-rmse:0.25201	test-rmse:12781.64452
```

Optimized XGBoost 2

- Optimal Features
- Optimal Hyperparameters
- Trained on Raw Dataset

```
# Convert test data to DMatrix format
dtrain = xgb.DMatrix(X_train_CAS_dropped[list(optimal_features_CAS_dropped)], label=y_train_CAS_dropped)
dtest = xgb.DMatrix(X_test_CAS_dropped[list(optimal_features_CAS_dropped)], label=y_test_CAS_dropped)
```

Determination of optimal num_boost_round

```
evals_result_CAS_dropped = {}

# Train the final model
final_model_CAS_dropped = xgb.train(best_params_CAS_dropped, dtrain, num_boost_round=1000, early_stopping_rounds=30, evals=[(dtr
                  evals_result=evals_result_CAS_dropped)
```

```
[0]	train-rmse:1602394.78629	test-rmse:12310.31219
[1]	train-rmse:1601049.20396	test-rmse:12447.93114
[2]	train-rmse:247107.93377	test-rmse:12508.41538
[3]	train-rmse:247094.27641	test-rmse:12421.06864
```

```
[4]	train-rmse:38153.39067	test-rmse:12446.10278
[5]	train-rmse:5913.38238	test-rmse:12204.74036
[6]	train-rmse:5915.88864	test-rmse:12403.51241
[7]	train-rmse:5745.49813	test-rmse:12444.76614
[8]	train-rmse:5883.02233	test-rmse:12466.42200
[9]	train-rmse:932.67265	test-rmse:12189.66132
[10]	train-rmse:268.09638	test-rmse:12152.21424
[11]	train-rmse:253.29569	test-rmse:12151.26545
[12]	train-rmse:241.22833	test-rmse:12157.31317
[13]	train-rmse:202.94550	test-rmse:12147.19578
[14]	train-rmse:223.40859	test-rmse:12145.76781
[15]	train-rmse:165.73906	test-rmse:12142.15171
[16]	train-rmse:158.62397	test-rmse:12144.12858
[17]	train-rmse:51.41457	test-rmse:12145.88952
[18]	train-rmse:51.80484	test-rmse:12140.34193
[19]	train-rmse:41.54818	test-rmse:12139.98623
[20]	train-rmse:49.52654	test-rmse:12140.76733
[21]	train-rmse:47.84993	test-rmse:12142.14766
[22]	train-rmse:35.16584	test-rmse:12145.02347
[23]	train-rmse:10.85244	test-rmse:12145.34387
[24]	train-rmse:8.92859	test-rmse:12145.33880
[25]	train-rmse:6.38920	test-rmse:12145.57790
[26]	train-rmse:5.70117	test-rmse:12145.48659
[27]	train-rmse:5.16950	test-rmse:12145.31321
[28]	train-rmse:3.97258	test-rmse:12146.25122
[29]	train-rmse:3.60399	test-rmse:12146.17854
[30]	train-rmse:2.80307	test-rmse:12146.10404
[31]	train-rmse:2.53569	test-rmse:12146.06513
[32]	train-rmse:2.29142	test-rmse:12146.12167
[33]	train-rmse:2.46055	test-rmse:12146.15860
[34]	train-rmse:2.36656	test-rmse:12146.17064
[35]	train-rmse:2.16004	test-rmse:12146.07583
[36]	train-rmse:2.07432	test-rmse:12146.13630
[37]	train-rmse:2.14112	test-rmse:12146.11894
[38]	train-rmse:2.46385	test-rmse:12146.11669
[39]	train-rmse:2.20447	test-rmse:12146.06519
[40]	train-rmse:2.07281	test-rmse:12146.09099
[41]	train-rmse:1.50348	test-rmse:12146.07556
[42]	train-rmse:1.64946	test-rmse:12146.12237
[43]	train-rmse:1.19462	test-rmse:12146.10532
[44]	train-rmse:1.68894	test-rmse:12146.14418
[45]	train-rmse:0.88928	test-rmse:12146.13609
[46]	train-rmse:0.76567	test-rmse:12146.00371
[47]	train-rmse:0.75361	test-rmse:12145.98221
[48]	train-rmse:0.64235	test-rmse:12145.94637
```

```python
# Train the final model
final_model_CAS_dropped = xgb.train(best_params_CAS_dropped, dtrain, num_boost_round=(np.argmin(evals_result_CAS_dropped['train'
                  evals_result=evals_result_CAS_dropped)

# Make predictions on the test set
y_pred_final_CAS_dropped = final_model_CAS_dropped.predict(dtest)
```

```
[0]	train-rmse:1602394.78629	test-rmse:12310.31219
[1]	train-rmse:1601049.20396	test-rmse:12447.93114
[2]	train-rmse:247107.93377	test-rmse:12508.41538
[3]	train-rmse:247094.27641	test-rmse:12421.06864
[4]	train-rmse:38153.39067	test-rmse:12446.10278
[5]	train-rmse:5913.38238	test-rmse:12204.74036
[6]	train-rmse:5915.88864	test-rmse:12403.51241
[7]	train-rmse:5745.49813	test-rmse:12444.76614
[8]	train-rmse:5883.02233	test-rmse:12466.42200
[9]	train-rmse:932.67265	test-rmse:12189.66132
[10]	train-rmse:268.09638	test-rmse:12152.21424
[11]	train-rmse:253.29569	test-rmse:12151.26545
[12]	train-rmse:241.22833	test-rmse:12157.31317
[13]	train-rmse:202.94550	test-rmse:12147.19578
[14]	train-rmse:223.40859	test-rmse:12145.76781
[15]	train-rmse:165.73906	test-rmse:12142.15171
[16]	train-rmse:158.62397	test-rmse:12144.12858
[17]	train-rmse:51.41457	test-rmse:12145.88952
[18]	train-rmse:51.80484	test-rmse:12140.34193
[19]	train-rmse:41.54818	test-rmse:12139.98623
[20]	train-rmse:49.52654	test-rmse:12140.76733
[21]	train-rmse:47.84993	test-rmse:12142.14766
[22]	train-rmse:35.16584	test-rmse:12145.02347
[23]	train-rmse:10.85244	test-rmse:12145.34387
[24]	train-rmse:8.92859	test-rmse:12145.33880
[25]	train-rmse:6.38920	test-rmse:12145.57790
[26]	train-rmse:5.70117	test-rmse:12145.48659
[27]	train-rmse:5.16950	test-rmse:12145.31321
[28]	train-rmse:3.97258	test-rmse:12146.25122
[29]	train-rmse:3.60399	test-rmse:12146.17854
```

```
[30]	train-rmse:2.80307	test-rmse:12146.10404
[31]	train-rmse:2.53569	test-rmse:12146.06513
[32]	train-rmse:2.29142	test-rmse:12146.12167
[33]	train-rmse:2.46055	test-rmse:12146.15860
[34]	train-rmse:2.36656	test-rmse:12146.17064
[35]	train-rmse:2.16004	test-rmse:12146.07583
[36]	train-rmse:2.07432	test-rmse:12146.13630
[37]	train-rmse:2.14112	test-rmse:12146.11894
[38]	train-rmse:2.46385	test-rmse:12146.11669
[39]	train-rmse:2.20447	test-rmse:12146.06519
[40]	train-rmse:2.07281	test-rmse:12146.09099
[41]	train-rmse:1.50348	test-rmse:12146.07556
[42]	train-rmse:1.64946	test-rmse:12146.12237
[43]	train-rmse:1.19462	test-rmse:12146.10532
[44]	train-rmse:1.68894	test-rmse:12146.14418
[45]	train-rmse:0.88928	test-rmse:12146.13609
[46]	train-rmse:0.76567	test-rmse:12146.00371
[47]	train-rmse:0.75361	test-rmse:12145.98221
[48]	train-rmse:0.64235	test-rmse:12145.94637
[49]	train-rmse:0.57622	test-rmse:12145.94569
```

Untuned XGBoost 1

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

```
dtrain = xgb.DMatrix(X_train_CAS, label=y_train_CAS)
dtest = xgb.DMatrix(X_test_CAS, label=y_test_CAS)


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

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

# Make predictions on the test set
y_pred_oob_imputed_CAS = oob_model_imputed_CAS.predict(dtest)
```

```
[0]	train-rmse:1354423.52169	test-rmse:381637.57776
[1]	train-rmse:1150657.29774	test-rmse:683512.05839
[2]	train-rmse:977710.56184	test-rmse:943594.21875
[3]	train-rmse:830853.27541	test-rmse:1165378.97870
[4]	train-rmse:706110.34248	test-rmse:1354110.01269
[5]	train-rmse:600128.39363	test-rmse:1514608.85262
[6]	train-rmse:510070.77980	test-rmse:1651061.83907
[7]	train-rmse:433538.00887	test-rmse:1767060.00561
[8]	train-rmse:368494.09609	test-rmse:1865664.06509
[9]	train-rmse:313211.59070	test-rmse:1949479.16954
[10]	train-rmse:266225.19171	test-rmse:2020722.82590
[11]	train-rmse:226288.49289	test-rmse:2081280.62915
[12]	train-rmse:192343.99227	test-rmse:2132754.94537
[13]	train-rmse:163491.36031	test-rmse:2176508.26098
[14]	train-rmse:138967.82406	test-rmse:2213698.54966
[15]	train-rmse:118122.64926	test-rmse:2245310.39081
[16]	train-rmse:100404.31155	test-rmse:2272180.64747
[17]	train-rmse:85343.87396	test-rmse:2295020.27003
[18]	train-rmse:72542.76190	test-rmse:2314433.93126
[19]	train-rmse:61661.40561	test-rmse:2330935.70805
[20]	train-rmse:52412.42676	test-rmse:2344962.16165
[21]	train-rmse:44550.98106	test-rmse:2356884.45646
[22]	train-rmse:37868.82074	test-rmse:2367018.30419
[23]	train-rmse:32188.90705	test-rmse:2375632.19335
[24]	train-rmse:27360.66554	test-rmse:2382954.10120
[25]	train-rmse:23257.02598	test-rmse:2389177.53586
[26]	train-rmse:19769.03860	test-rmse:2394467.58132
[27]	train-rmse:16803.84703	test-rmse:2398964.04661
[28]	train-rmse:14283.67274	test-rmse:2402786.15174
[29]	train-rmse:12141.50366	test-rmse:2406034.84007
```

Untuned XGBoost 2

- No Feature Selection

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

```
dtrain = xgb.DMatrix(X_train_CAS_dropped, label=y_train_CAS_dropped)
dtest = xgb.DMatrix(X_test_CAS_dropped, label=y_test_CAS_dropped)

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

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

# Make predictions on the test set
y_pred_oob_CAS = oob_model_CAS.predict(dtest)
```

```
[0]	train-rmse:1357685.79379	test-rmse:381761.33333
[1]	train-rmse:1153425.89863	test-rmse:683574.12030
[2]	train-rmse:980060.06786	test-rmse:943636.08946
[3]	train-rmse:832847.35942	test-rmse:1165410.22076
[4]	train-rmse:707802.22117	test-rmse:1354133.33251
[5]	train-rmse:601563.00580	test-rmse:1514625.86182
[6]	train-rmse:511289.02251	test-rmse:1651075.77468
[7]	train-rmse:434572.44065	test-rmse:1767070.84801
[8]	train-rmse:369373.57206	test-rmse:1865671.95156
[9]	train-rmse:313959.74865	test-rmse:1949485.47023
[10]	train-rmse:266861.11410	test-rmse:2020728.00461
[11]	train-rmse:226829.32309	test-rmse:2081284.40171
[12]	train-rmse:192803.28422	test-rmse:2132758.00004
[13]	train-rmse:163881.95854	test-rmse:2176510.45088
[14]	train-rmse:139299.07584	test-rmse:2213700.10194
[15]	train-rmse:118403.99067	test-rmse:2245311.07192
[16]	train-rmse:100643.20366	test-rmse:2272180.71708
[17]	train-rmse:85546.72110	test-rmse:2295019.66371
[18]	train-rmse:72714.76220	test-rmse:2314432.86138
[19]	train-rmse:61807.60853	test-rmse:2330934.05094
[20]	train-rmse:52536.50435	test-rmse:2344960.06151
[21]	train-rmse:44656.16165	test-rmse:2356882.13179
[22]	train-rmse:37957.90034	test-rmse:2367015.86583
[23]	train-rmse:32264.28569	test-rmse:2375629.66723
[24]	train-rmse:27424.80215	test-rmse:2382951.25055
[25]	train-rmse:23311.15951	test-rmse:2389174.76706
[26]	train-rmse:19814.45318	test-rmse:2394464.81694
[27]	train-rmse:16842.33788	test-rmse:2398961.28653
[28]	train-rmse:14316.14015	test-rmse:2402783.16113
[29]	train-rmse:12168.82551	test-rmse:2406031.82127
[30]	train-rmse:10343.68251	test-rmse:2408793.08686
```

Naive Model 1

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

```
y_pred_naive_CAS = np.full(y_test_CAS.shape, y_train_CAS.mean())
```

Naive Model 2

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

```
y_pred_naive_orig_CAS = np.full(y_test_CAS.shape, y_train_orig_CAS.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_CAS = ~np.isnan(y_test_orig_CAS)
```

Model Metrics evaluated on Imputed Test Set

Optimized XGBoost 1

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

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

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

```
Final model metrics:

NSE: (-0.08488257027113977, 'bad'),
PBIAS: (43.28929556662643, 'bad'),
KGE: (-0.19047708030715937, 'good')

Root Mean Squared Error: 12781.644535660605
Normalized Root Mean Squared Error: 0.11619782484986778
```

Untuned XGBoost 1

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

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

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

```
Final model metrics:

NSE: (-38529.92342967909, 'bad'),
PBIAS: (10659.22881102169, 'bad'),
KGE: (-220.51428300288126, 'bad')

Root Mean Squared Error: 2408796.315386435
Normalized Root Mean Squared Error: 21.898347397580295
```

Naive Model 1

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

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

```
Root Mean Squared Error: 110632.38378044139
Normalized Root Mean Squared Error: 1.0057580867138918
```

Naive Model 2

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

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

```
Root Mean Squared Error: 111171.54475899595
Normalized Root Mean Squared Error: 1.0106595947144605
```

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

Optimized XGBoost 1

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

rmse = mean_squared_error(y_test_CAS_dropped, y_pred_final_CAS[non_imputed_mask_CAS],squared=False)
print(f"\nRoot Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_CAS_dropped, y_pred_final_CAS[non_imputed_mask_CAS])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Final model metrics:

NSE: (-0.08488257027113977, 'bad'),
PBIAS: (43.28929556662643, 'bad'),
KGE: (-0.19047708030715937, 'good')

Root Mean Squared Error: 12781.644535660605
Normalized Root Mean Squared Error: 0.11619782484986778
```

Optimized XGBoost 2

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

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

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

```
Final model metrics:

NSE: (0.02034785499332281, 'bad'),
PBIAS: (58.39414514665331, 'bad'),
KGE: (-0.209182803582418 8, 'good')
```

```
Root Mean Squared Error: 12145.94517211003
Normalized Root Mean Squared Error: 0.11041868718906563
```

Untuned XGBoost 2

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

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

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

```
Final model metrics:

NSE: (-38529.82129668686, 'bad'),
PBIAS: (10623.828110432103, 'bad'),
KGE: (-220.35052160477284, 'bad')

Root Mean Squared Error: 2408793.1229150835
Normalized Root Mean Squared Error: 21.898318374849623
```

Naive Model 1

```
rmse = mean_squared_error(y_test_CAS_dropped, y_pred_naive_CAS[non_imputed_mask_CAS],squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_CAS_dropped, y_pred_naive_CAS[non_imputed_mask_CAS])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 110632.38378044139
Normalized Root Mean Squared Error: 1.0057580867138918
```

Naive Model 2

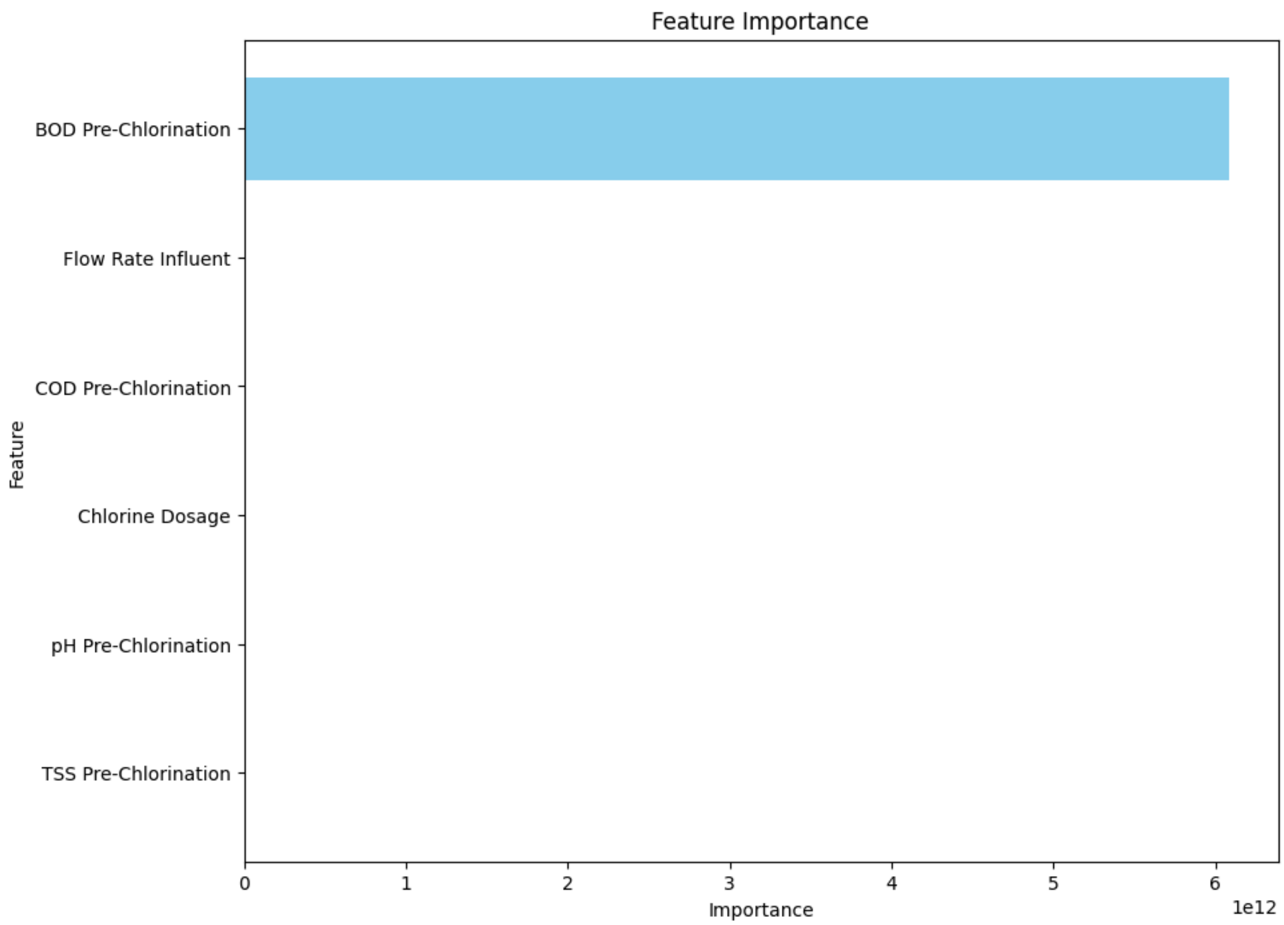
```
rmse = mean_squared_error(y_test_CAS_dropped, y_pred_naive_orig_CAS[non_imputed_mask_CAS],squared=False)
print(f"Root Mean Squared Error: {rmse}")

nrmse = compute_nrmse(y_test_CAS_dropped, y_pred_naive_orig_CAS[non_imputed_mask_CAS])
print(f"Normalized Root Mean Squared Error: {nrmse}")
```

```
Root Mean Squared Error: 111171.54475899595
Normalized Root Mean Squared Error: 1.0106595947144605
```

Feature Importance

```
# Get feature importance
importance_CAS = final_model_CAS.get_score(importance_type='gain')

name_dict_CAS = {
    '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.Pre.chlorination..ppm.':'COD Pre-Chlorination'
    }

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

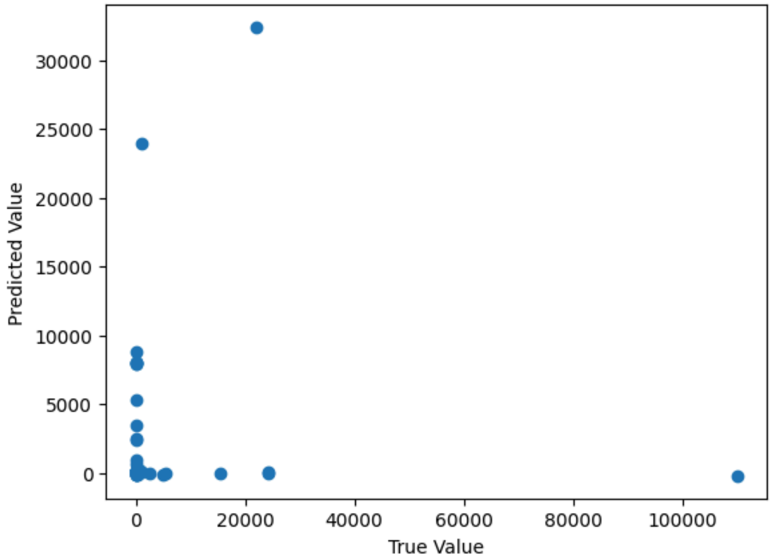
```
importance_df_CAS['Feature'] = importance_df_CAS['Feature'].replace(name_dict_CAS)

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

# Plot feature importance
plt.figure(figsize=(10, 8))
plt.barh(importance_df_CAS['Feature'], importance_df_CAS['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()
```



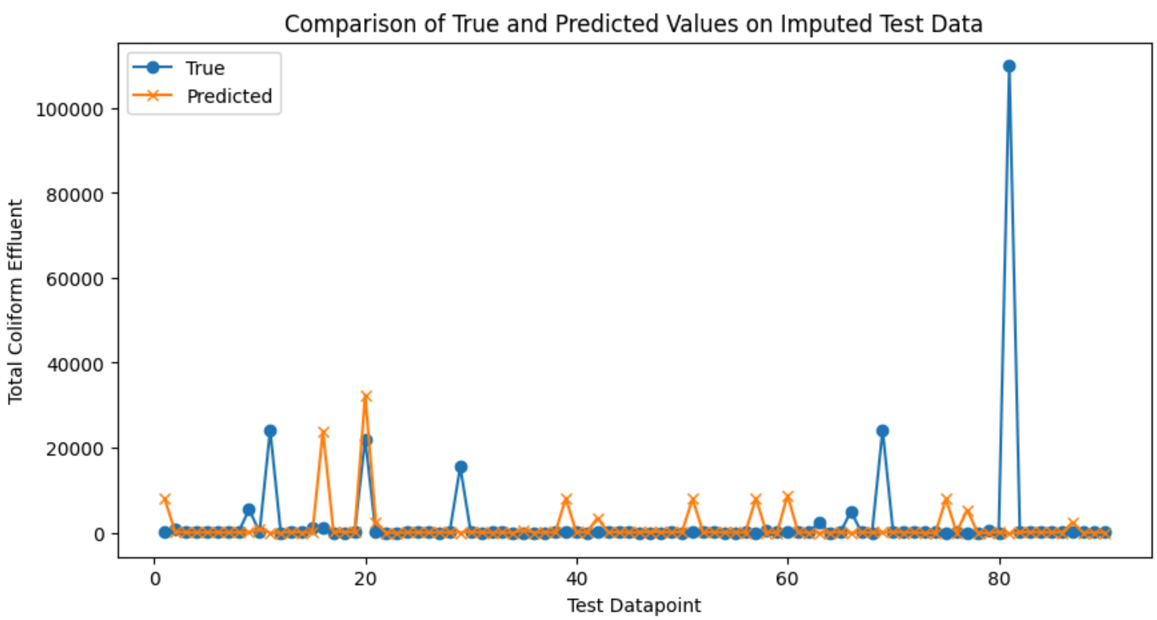
Data Visualization for Model Evaluation

Optimized XGBoost on Imputed Test Dataset

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

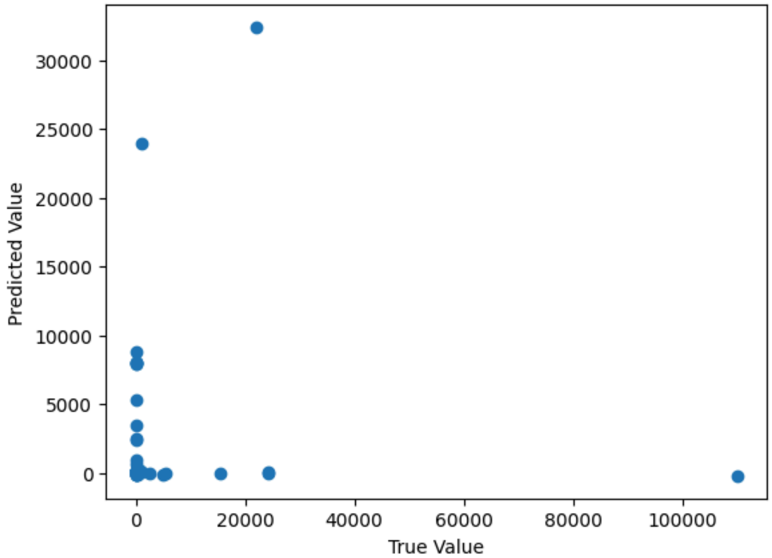



```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_CAS) + 1)

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

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

# Show plot
plt.show()
```

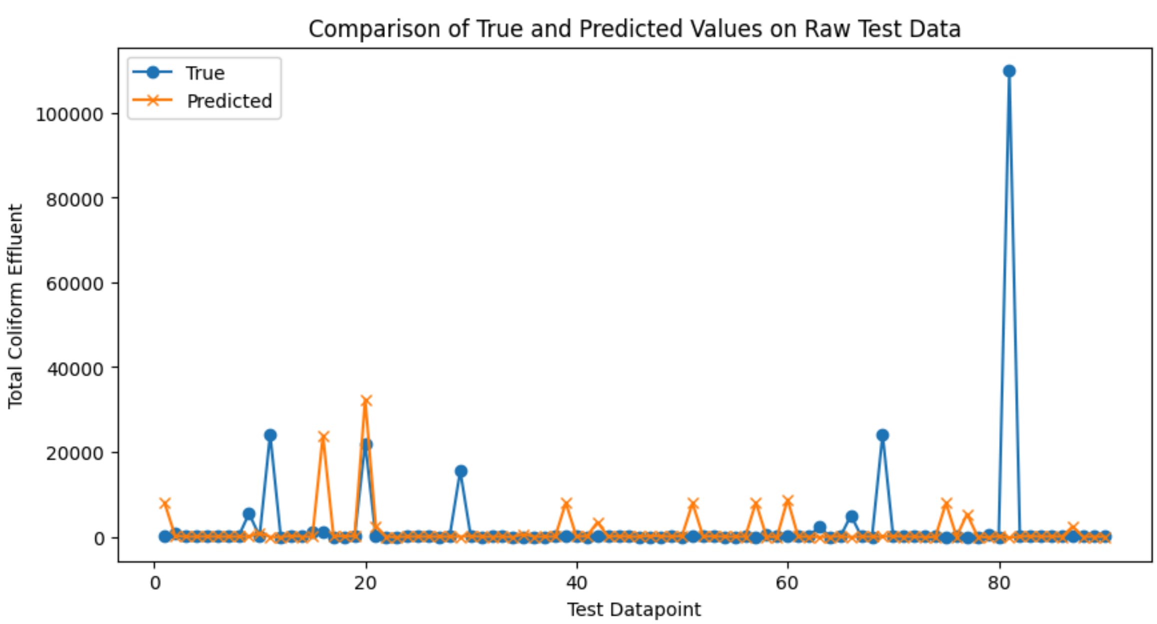


Optimized XGBoost on Non-Imputed (Raw) Test Dataset

```
# without imputation
plt.scatter(y_test_orig_CAS[non_imputed_mask_CAS],y_pred_final_CAS[non_imputed_mask_CAS])
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_CAS[non_imputed_mask_CAS]) + 1)

# Plotting
plt.figure(figsize=(10, 5))
plt.plot(x, y_test_orig_CAS[non_imputed_mask_CAS], label='True', marker='o')
plt.plot(x, y_pred_final_CAS[non_imputed_mask_CAS], label='Predicted', marker='x')

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

# Show plot
plt.show()
```



Exporting Results

```
# Determine the maximum length of the columns
max_length = max(len(y_test_CAS), len(y_test_CAS_dropped), len(y_pred_final_CAS), len(y_pred_final_CAS_dropped), len(y_pred_oob_

# 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_CAS = extend_with_nan(y_test_CAS, max_length)
    y_test_CAS_dropped = extend_with_nan(y_test_CAS_dropped.reset_index(drop='True'), max_length)
    y_pred_final_CAS = extend_with_nan(y_pred_final_CAS, max_length)
    y_pred_final_CAS_dropped = extend_with_nan(y_pred_final_CAS_dropped, max_length)
    y_pred_oob_imputed_CAS = extend_with_nan(y_pred_oob_imputed_CAS, max_length)
```