```
!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 | 3.54 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='Residual chlorine\n(ppm)')
y_orig_CAS = CAS['Residual chlorine\n(ppm)']
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	Missingnes Rat
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	Missingnes Rat
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	10,000,000,000
Total.Coliform.Effluent..MPN.100mL.	90.00	2,378.91	12,340.18	1.00	2.00	10.00	20.00	110,000
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	1,700,000,000
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	349
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	1,238
COD.Pre.chlorination..ppm.	90.00	20.40	11.86	3.00	13.00	17.50	26.30	68
TSS.Pre.chlorination..ppm.	90.00	6.06	6.19	1.00	2.00	3.00	8.00	38
pH.Pre.chlorination	90.00	7.08	0.29	6.25	6.90	7.09	7.27	7
Chlorine.Dosage..L.d.	90.00	180.33	97.55	3.00	110.30	156.10	216.35	573
Residual.chlorine..ppm.	90.00	2.48	1.88	0.01	0.58	2.19	4.74	5

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

```
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
434	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.431129	1.703038	0.027306	0.161537
70	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.613562	1.706146	0.055901	0.122945
184	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.503695	1.709929	0.054186	0.146570
15	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.644796	1.714408	0.058059	0.173788
143	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.516150	1.714627	0.061319	0.094092
...	...	...	...	...	...
213	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	0.552300	2.026122	0.044545	0.102608
115	(Chlorine.Dosage..L.d., BOD.Influent...ppm., C...	0.563381	2.035176	0.046364	0.110444
98	(Chlorine.Dosage..L.d., Fecal.Coliform.Influen...	0.514983	2.038046	0.017078	0.171205
210	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	0.568931	2.055464	0.054060	0.120276
20	(Chlorine.Dosage..L.d., Total.Coliform.Influen...	0.747910	2.056769	0.048512	0.092515

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
434	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.431129	1.703038	0.027306	0.161537
70	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.613562	1.706146	0.055901	0.122945
184	(Chlorine.Dosage..L.d., Flow.Rate.Influent..m3...	0.503695	1.709929	0.054186	0.146570

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

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

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

```
('Chlorine.Dosage..L.d.',
 'Flow.Rate.Influent..m3.d.',
 'COD.Pre.chlorination..ppm.',
 'TSS.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.Influent...ppm.',
 '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)	Total Coliform Effluent (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
174	6253.0	60000000.0	97.0	90900000.0	95.0	155.0	7.0	441.0	35.(
70	8019.0	50000000.0	1.0	NaN	NaN	55.0	5.0	211.0	10.(
179	17590.0	20000000.0	245.0	15000000.0	10.0	52.0	3.0	207.0	33.(
252	14848.0	23000000.0	2.0	7900000.0	2.0	58.0	2.0	100.0	5.(
284	4635.0	NaN	10.0	NaN	10.0	NaN	NaN	149.0	Nal

```
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)',
       'Total Coliform Effluent (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'],
      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
152	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.876139	1.545199	0.047778	0.160089
470	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.541888	1.562970	0.036731	0.130902
36	(Chlorine Dosage (L/d), Total Coliform Effluen...	1.050614	1.570873	0.036254	0.107744
947	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.448809	1.575514	0.039836	0.164993
484	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.635234	1.577490	0.031198	0.160092
...	...	...	...	...	...
57	(Chlorine Dosage (L/d), BOD Pre-chlorination\n...	1.079445	2.281653	0.048419	0.104983
1	(Chlorine Dosage (L/d), Total Coliform Influen...	1.275154	2.291641	0.045415	0.134590
27	(Chlorine Dosage (L/d), Total Coliform Influen...	0.933585	2.298750	0.058474	0.160184
138	(Chlorine Dosage (L/d), Total Coliform Influen...	0.919063	2.356408	0.057193	0.152600
25	(Chlorine Dosage (L/d), Total Coliform Influen...	1.062029	2.362547	0.043413	0.096269

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
152	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.876139	1.545199	0.047778	0.160089
470	(Chlorine Dosage (L/d), Total Coliform Effluen...	0.541888	1.562970	0.036731	0.130902
36	(Chlorine Dosage (L/d), Total Coliform Effluen...	1.050614	1.570873	0.036254	0.107744

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

```
('Chlorine Dosage (L/d)',
 'Total Coliform Effluent (MPN/100mL)',
 'Fecal Coliform Influent (MPN/100mL)',
 'TSS Pre-chlorination (ppm)')
```

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

```
('Chlorine Dosage (L/d)',
 'Total Coliform Effluent (MPN/100mL)',
 'Fecal Coliform Effluent (MPN/100mL)',
 'TSS Pre-chlorination (ppm)',
 'pH Pre-chlorination')
```

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

```
('Chlorine Dosage (L/d)',
 'Total Coliform Effluent (MPN/100mL)',
 'TSS Pre-chlorination (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)',
 'Total Coliform Effluent (MPN/100mL)',
 'Fecal Coliform Influent (MPN/100mL)',
 'TSS Pre-chlorination (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

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         | -1.66     | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -1.635    | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -1.701    | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -1.767    | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
| 5         | -1.802    | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -1.658    | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -1.665    | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -1.8      | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -1.67     | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -1.714    | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -1.644    | -3.301    | 0.7099    | -2.401    | -3.045    | -1.984    | 1.589     | 0.6833    |
| 12        | -1.648    | -5.558    | 0.8641    | -2.37     | -4.079    | -1.919    | 1.459     | 0.514     |
| 13        | -1.751    | -4.602    | 0.3       | -3.0      | -3.392    | 0.3       | 1.0       | 1.0       |
| 14        | -1.659    | -4.822    | 1.0       | -1.873    | -2.903    | -2.847    | 2.502     | 0.5       |
| 15        | -1.734    | -4.395    | 1.0       | -0.5926   | -3.558    | -2.059    | 1.0       | 0.5       |
| 16        | -1.701    | -4.73     | 0.8322    | -3.0      | -2.35     | -2.348    | 1.372     | 0.5       |
| 17        | -1.644    | -3.946    | 0.7216    | -1.985    | -2.673    | -1.483    | 2.684     | 0.5349    |
| 18        | -1.652    | -3.901    | 0.6847    | -2.67     | -4.436    | -2.383    | 2.551     | 0.6122    |
| 19        | -1.691    | -1.863    | 0.6207    | -3.0      | -3.532    | -2.527    | 2.448     | 0.7649    |
| 20        | -1.673    | -5.756    | 0.7963    | -1.767    | -2.813    | -1.243    | 2.721     | 0.5       |
| 21        | -1.67     | -5.781    | 1.0       | -2.651    | -4.937    | -2.759    | 2.7       | 0.5       |
| 22        | -1.695    | -3.118    | 1.0       | -1.866    | -1.792    | -1.216    | 1.748     | 1.0       |
| 23        | -1.732    | -3.782    | 0.5451    | -3.0      | -5.059    | -3.049    | 1.023     | 0.669     |
```

```
| 24 | -1.708 | -4.552 | 0.3    | -2.178 | -3.524  | -1.808  | 2.286 | 1.0    |
| 25 | -1.671 | -4.584 | 1.0    | -1.577 | -1.848  | -1.804  | 2.454 | 0.5    |
| 26 | -1.66  | -3.406 | 1.0    | -2.263 | -3.141  | -2.501  | 2.646 | 0.5    |
| 27 | -1.665 | -3.209 | 1.0    | -2.649 | -3.525  | -1.245  | 2.382 | 0.5    |
| 28 | -1.658 | -3.029 | 1.0    | -2.154 | -5.239  | -2.499  | 3.0   | 0.5    |
| 29 | -1.679 | -4.492 | 1.0    | -1.509 | -2.269  | -0.3492 | 2.21  | 0.5    |
| 30 | -1.659 | -4.141 | 1.0    | -2.104 | -4.421  | -3.607  | 3.0   | 0.5    |
| 31 | -1.671 | -2.571 | 0.3    | -1.782 | -5.334  | -4.07   | 3.0   | 0.5    |
| 32 | -1.653 | -6.0   | 1.0    | -2.339 | -5.335  | -1.386  | 1.503 | 0.5    |
| 33 | -1.667 | -4.457 | 1.0    | -2.791 | -5.701  | -1.537  | 3.0   | 0.5    |
| 34 | -1.753 | 0.3    | 0.3466 | -3.0   | -0.2478 | -2.493  | 1.0   | 0.913  |
| 35 | -1.667 | -5.327 | 0.3    | -1.516 | -1.544  | -1.198  | 1.03  | 0.5    |
| 36 | -1.731 | 0.3    | 0.3    | -3.0   | -2.684  | -0.7793 | 2.695 | 0.5    |
| 37 | -1.782 | -3.637 | 1.0    | -3.0   | -5.541  | -3.254  | 3.0   | 1.0    |
| 38 | -1.642 | -3.716 | 1.0    | -1.707 | -4.295  | -2.343  | 3.0   | 0.5    |
| 39 | -1.654 | -2.906 | 0.4166 | -1.819 | -4.369  | -2.374  | 2.405 | 0.5    |
| 40 | -1.65  | -3.43  | 0.7276 | -1.99  | -4.87   | -1.366  | 3.0   | 0.5    |
| 41 | -1.742 | -3.63  | 0.3    | -1.104 | -5.274  | -2.339  | 3.0   | 0.5    |
| 42 | -1.672 | -3.031 | 1.0    | -2.411 | -4.281  | -2.003  | 3.0   | 0.5    |
| 43 | -1.663 | -3.394 | 1.0    | -1.554 | -3.385  | -1.718  | 2.216 | 0.5    |
| 44 | -1.665 | -4.672 | 1.0    | -2.165 | -4.016  | -2.589  | 3.0   | 0.5    |
| 45 | -1.645 | -3.785 | 0.3    | -1.87  | -2.313  | -1.976  | 1.825 | 0.5    |
| 46 | -1.685 | -3.3   | 1.0    | -1.507 | -3.811  | -3.468  | 2.566 | 0.5    |
| 47 | -1.66  | -4.083 | 1.0    | -2.273 | -4.817  | -1.461  | 2.096 | 0.5    |
| 48 | -1.686 | -6.0   | 1.0    | -3.0   | -4.475  | -1.401  | 2.194 | 0.5    |
| 49 | -1.665 | -6.0   | 1.0    | -2.007 | -4.869  | -2.503  | 1.182 | 0.5    |
| 50 | -1.668 | -4.189 | 0.3    | -2.491 | -1.932  | -1.186  | 2.124 | 0.5    |
| 51 | -1.664 | -1.7   | 1.0    | -1.817 | -5.351  | -2.72   | 2.423 | 0.5    |
| 52 | -1.651 | -2.558 | 0.8347 | -2.418 | -5.545  | -1.52   | 2.192 | 0.5    |
| 53 | -1.674 | -3.228 | 0.6335 | -2.762 | -5.576  | -0.6509 | 2.883 | 0.9166 |
| 54 | -1.706 | -1.484 | 0.3    | -2.867 | -6.0    | -2.056  | 2.438 | 0.5    |
| 55 | -1.671 | -6.0   | 1.0    | -1.747 | -4.344  | -1.155  | 1.0   | 0.5    |
```

```
# 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']
```

```
best_params_CAS
```

```
{'alpha': 1.2893278167862703e-05,
 'eta': 0.012194477345009136,
 'gamma': 0.0002313333925757085,
 'max_depth': 5,
 'subsample': 0.5,
 'lambda': 0.035986251485942866,
 'objective': 'reg:squarederror',
 'eval_metric': 'rmse',
 'colsample_bytree': 0.5670693188676399}
```

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         | -1.679    | 0.04075   | 0.4513    | -2.68     | -1.662    | -1.582    | 2.026     | 0.7673    |
| 2         | -1.557    | -4.514    | 0.7529    | -1.843    | -2.2      | -1.339    | 1.596     | 0.5436    |
| 3         | -1.631    | -1.108    | 0.5069    | -1.136    | -4.974    | -0.5216   | 2.693     | 0.8202    |
| 4         | -1.771    | -3.147    | 0.6275    | -2.063    | 1.604     | -0.4504   | 1.466     | 0.7294    |
| 5         | -1.619    | -2.356    | 0.4052    | -0.3806   | -0.09483  | -5.01     | 1.643     | 0.6674    |
| 6         | -1.588    | -2.162    | 0.5228    | -2.659    | -5.793    | -3.144    | 2.227     | 0.7522    |
| 7         | -1.583    | -0.1605   | 0.6002    | -1.973    | 0.8823    | -3.597    | 1.193     | 0.6362    |
| 8         | -1.681    | -0.9486   | 0.7394    | -0.4943   | -0.4982   | -3.564    | 2.166     | 0.5334    |
| 9         | -1.575    | -1.814    | 0.8098    | -2.344    | -2.07     | -3.54     | 1.193     | 0.8742    |
| 10        | -1.582    | 0.06321   | 0.6188    | -0.4746   | -0.88     | -0.04974  | 2.478     | 0.7132    |
| 11        | -1.567    | -5.294    | 0.6658    | -1.215    | -5.342    | -0.3629   | 2.421     | 0.9975    |
| 12        | -1.644    | -4.018    | 0.5023    | -2.865    | -0.7772   | -2.167    | 2.849     | 0.9048    |
| 13        | -1.567    | -5.736    | 0.7323    | -1.385    | -3.443    | -1.267    | 1.381     | 0.8479    |
| 14        | -1.576    | -4.562    | 0.7243    | -1.636    | -3.526    | -0.8784   | 2.032     | 0.678     |
| 15        | -1.572    | -4.034    | 0.8592    | -1.875    | -3.189    | -2.687    | 1.0       | 0.7322    |
| 16        | -1.767    | -4.65     | 1.0       | 0.0       | -2.496    | -1.644    | 1.0       | 0.5       |
| 17        | -1.568    | -4.843    | 0.7227    | -2.072    | -3.068    | -1.674    | 1.454     | 0.7534    |
| 18        | -1.575    | -5.479    | 0.6153    | -2.257    | -2.569    | -0.6385   | 1.815     | 0.655     |
| 19        | -1.56     | -5.869    | 0.5613    | -2.241    | -4.495    | -0.9907   | 1.949     | 1.0       |
| 20        | -1.674    | -3.514    | 0.8813    | -3.0      | -2.671    | -1.48     | 1.053     | 0.5       |
| 21        | -1.589    | -5.913    | 0.5663    | -1.35     | -3.94     | -0.5967   | 2.642     | 1.0       |
| 22        | -1.559    | -5.059    | 0.6696    | -1.594    | -4.803    | -1.827    | 1.768     | 1.0       |
| 23        | -1.662    | -5.802    | 0.3       | -2.09     | -3.859    | -2.375    | 2.216     | 0.5       |
| 24        | -1.568    | -5.24     | 0.973     | -1.637    | -4.648    | -0.7029   | 1.236     | 1.0       |
| 25        | -1.52     | -4.725    | 1.0       | -2.382    | -5.273    | -1.0      | 2.258     | 1.0       |
| 26        | -1.664    | -4.927    | 0.3       | -2.416    | -5.999    | -1.093    | 1.735     | 1.0       |
| 27        | -1.545    | -4.869    | 1.0       | -1.969    | -4.64     | -0.9729   | 2.524     | 1.0       |
| 28        | -1.594    | -4.79     | 1.0       | -2.859    | -4.673    | -0.4675   | 2.141     | 1.0       |
| 29        | -1.526    | -4.093    | 1.0       | -1.865    | -5.152    | -1.323    | 2.301     | 1.0       |
| 30        | -1.609    | -4.528    | 1.0       | -2.352    | -5.37     | -1.589    | 3.0       | 1.0       |
| 31        | -1.564    | -4.506    | 1.0       | -1.759    | -5.103    | -0.8788   | 1.991     | 0.5       |
| 32        | -1.558    | -4.374    | 1.0       | -2.33     | -4.716    | -1.495    | 1.801     | 1.0       |
| 33        | -1.542    | -4.303    | 1.0       | -0.8715   | -4.919    | -1.41     | 2.315     | 1.0       |
| 34        | -1.544    | -3.885    | 1.0       | -1.486    | -5.231    | -0.4462   | 2.737     | 1.0       |
| 35        | -1.55     | -3.204    | 1.0       | -1.158    | -5.295    | -1.29     | 1.963     | 1.0       |
| 36        | -1.574    | -3.852    | 1.0       | -0.5178   | -6.0      | -1.052    | 2.705     | 1.0       |
| 37        | -1.762    | -3.518    | 0.3       | -1.296    | -4.552    | -1.242    | 2.646     | 1.0       |
| 38        | -1.552    | -4.18     | 1.0       | -1.288    | -5.646    | -1.066    | 2.064     | 1.0       |
| 39        | -1.519    | -3.948    | 1.0       | -2.263    | -5.473    | -0.6171   | 2.302     | 1.0       |
| 40        | -1.557    | -3.179    | 1.0       | -1.845    | -5.97     | -0.6938   | 2.166     | 1.0       |
| 41        | -1.552    | -4.027    | 1.0       | -0.986    | -5.296    | -2.082    | 1.484     | 1.0       |
| 42        | -1.572    | -5.187    | 1.0       | -0.3754   | -5.188    | -1.594    | 1.952     | 1.0       |
| 43        | -1.569    | -4.497    | 1.0       | -2.097    | -5.71     | -0.02446  | 2.894     | 1.0       |
| 44        | -1.558    | -3.275    | 1.0       | -2.021    | -5.588    | -1.645    | 1.327     | 1.0       |
| 45        | -1.537    | -2.994    | 1.0       | -0.6111   | -5.956    | -1.43     | 1.096     | 1.0       |
| 46        | -1.549    | -2.762    | 1.0       | -0.8552   | -6.0      | -2.368    | 1.713     | 1.0       |
| 47        | -1.554    | -1.947    | 1.0       | -1.271    | -6.0      | -1.581    | 1.0       | 1.0       |
| 48        | -1.603    | -2.565    | 1.0       | -0.4877   | -5.159    | -2.078    | 1.0       | 0.5       |
| 49        | -1.568    | -2.743    | 0.9098    | -1.156    | -5.748    | 0.1843    | 1.125     | 0.89      |
| 50        | -1.598    | -3.725    | 1.0       | -0.07236  | -5.524    | -0.5888   | 1.262     | 1.0       |
| 51        | -1.552    | -3.774    | 1.0       | 0.0       | -6.0      | -2.409    | 2.008     | 1.0       |
| 52        | -1.545    | -3.859    | 1.0       | -0.9083   | -6.0      | -3.516    | 1.628     | 1.0       |
| 53        | -1.58     | -3.163    | 1.0       | -0.1924   | -6.0      | -3.744    | 2.516     | 1.0       |
| 54        | -1.692    | -4.503    | 1.0       | 0.0       | -6.0      | -3.238    | 1.0       | 1.0       |
```

```
| 55        | -1.531    | -3.75     | 1.0       | -1.246    | -6.0      | -2.634    | 2.281     | 1.0       |
| 56        | -1.558    | -3.466    | 1.0       | -1.92     | -6.0      | -3.331    | 1.627     | 1.0       |
```

```
# 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': 0.0012331699874173379,
 'eta': 0.06249300799212352,
 'gamma': 0.505178437417506,
 'max_depth': 6,
 'subsample': 1.0,
 'lambda': 2.37965081867787e-05,
 '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:1.82462      test-rmse:1.87365
[1]     train-rmse:1.81522      test-rmse:1.87538
[2]     train-rmse:1.80703      test-rmse:1.87581
[3]     train-rmse:1.79879      test-rmse:1.87232
[4]     train-rmse:1.79217      test-rmse:1.87202
[5]     train-rmse:1.78153      test-rmse:1.86820
[6]     train-rmse:1.77113      test-rmse:1.86644
[7]     train-rmse:1.76425      test-rmse:1.86529
[8]     train-rmse:1.75774      test-rmse:1.85905
[9]     train-rmse:1.75137      test-rmse:1.86069
[10]    train-rmse:1.74560      test-rmse:1.85948
[11]    train-rmse:1.73807      test-rmse:1.85881
[12]    train-rmse:1.73115      test-rmse:1.85862
[13]    train-rmse:1.72438      test-rmse:1.85803
[14]    train-rmse:1.71799      test-rmse:1.85974
[15]    train-rmse:1.71006      test-rmse:1.86089
[16]    train-rmse:1.70200      test-rmse:1.85867
[17]    train-rmse:1.69745      test-rmse:1.85924
```

```
[18]	train-rmse:1.68674	test-rmse:1.85843
[19]	train-rmse:1.67954	test-rmse:1.86116
[20]	train-rmse:1.67170	test-rmse:1.86035
[21]	train-rmse:1.66541	test-rmse:1.86067
[22]	train-rmse:1.66071	test-rmse:1.85775
[23]	train-rmse:1.65343	test-rmse:1.85618
[24]	train-rmse:1.64571	test-rmse:1.85461
[25]	train-rmse:1.63997	test-rmse:1.85534
[26]	train-rmse:1.63064	test-rmse:1.85576
[27]	train-rmse:1.62429	test-rmse:1.85651
[28]	train-rmse:1.61911	test-rmse:1.85895
[29]	train-rmse:1.61058	test-rmse:1.85850
[30]	train-rmse:1.60529	test-rmse:1.85711
[31]	train-rmse:1.59811	test-rmse:1.85618
[32]	train-rmse:1.59046	test-rmse:1.85421
[33]	train-rmse:1.58659	test-rmse:1.85388
[34]	train-rmse:1.57859	test-rmse:1.85205
[35]	train-rmse:1.57345	test-rmse:1.85254
[36]	train-rmse:1.56641	test-rmse:1.85009
[37]	train-rmse:1.56037	test-rmse:1.85182
[38]	train-rmse:1.55424	test-rmse:1.84907
[39]	train-rmse:1.54858	test-rmse:1.84628
[40]	train-rmse:1.54294	test-rmse:1.84543
[41]	train-rmse:1.53811	test-rmse:1.84415
[42]	train-rmse:1.53265	test-rmse:1.84480
[43]	train-rmse:1.52692	test-rmse:1.84173
[44]	train-rmse:1.52067	test-rmse:1.84379
[45]	train-rmse:1.51287	test-rmse:1.84192
[46]	train-rmse:1.50851	test-rmse:1.84221
[47]	train-rmse:1.50263	test-rmse:1.84083
[48]	train-rmse:1.49762	test-rmse:1.84013
[49]	train-rmse:1.49137	test-rmse:1.83967
[50]	train-rmse:1.48665	test-rmse:1.84386
[51]	train-rmse:1.48244	test-rmse:1.84160
[52]	train-rmse:1.47806	test-rmse:1.84307
[53]	train-rmse:1.47377	test-rmse:1.83998
[54]	train-rmse:1.46739	test-rmse:1.83995
[55]	train-rmse:1.46109	test-rmse:1.84036
[56]	train-rmse:1.45640	test-rmse:1.83925
[57]	train-rmse:1.45145	test-rmse:1.84008
```

```
# 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:1.82462	test-rmse:1.87365
[1]	train-rmse:1.81522	test-rmse:1.87538
[2]	train-rmse:1.80703	test-rmse:1.87581
[3]	train-rmse:1.79879	test-rmse:1.87232
[4]	train-rmse:1.79217	test-rmse:1.87202
[5]	train-rmse:1.78153	test-rmse:1.86820
[6]	train-rmse:1.77113	test-rmse:1.86644
[7]	train-rmse:1.76425	test-rmse:1.86529
[8]	train-rmse:1.75774	test-rmse:1.85905
[9]	train-rmse:1.75137	test-rmse:1.86069
[10]	train-rmse:1.74560	test-rmse:1.85948
[11]	train-rmse:1.73807	test-rmse:1.85881
[12]	train-rmse:1.73115	test-rmse:1.85862
[13]	train-rmse:1.72438	test-rmse:1.85803
[14]	train-rmse:1.71799	test-rmse:1.85974
[15]	train-rmse:1.71006	test-rmse:1.86089
[16]	train-rmse:1.70200	test-rmse:1.85867
[17]	train-rmse:1.69745	test-rmse:1.85924
[18]	train-rmse:1.68674	test-rmse:1.85843
[19]	train-rmse:1.67954	test-rmse:1.86116
[20]	train-rmse:1.67170	test-rmse:1.86035
[21]	train-rmse:1.66541	test-rmse:1.86067
[22]	train-rmse:1.66071	test-rmse:1.85775
[23]	train-rmse:1.65343	test-rmse:1.85618
[24]	train-rmse:1.64571	test-rmse:1.85461
[25]	train-rmse:1.63997	test-rmse:1.85534
[26]	train-rmse:1.63064	test-rmse:1.85576
[27]	train-rmse:1.62429	test-rmse:1.85651
[28]	train-rmse:1.61911	test-rmse:1.85895
[29]	train-rmse:1.61058	test-rmse:1.85850
[30]	train-rmse:1.60529	test-rmse:1.85711
[31]	train-rmse:1.59811	test-rmse:1.85618
[32]	train-rmse:1.59046	test-rmse:1.85421
[33]	train-rmse:1.58659	test-rmse:1.85388
[34]	train-rmse:1.57859	test-rmse:1.85205
[35]	train-rmse:1.57345	test-rmse:1.85254
```

```
[36]	train-rmse:1.56641	test-rmse:1.85009
[37]	train-rmse:1.56037	test-rmse:1.85182
[38]	train-rmse:1.55424	test-rmse:1.84907
[39]	train-rmse:1.54858	test-rmse:1.84628
[40]	train-rmse:1.54294	test-rmse:1.84543
[41]	train-rmse:1.53811	test-rmse:1.84415
[42]	train-rmse:1.53265	test-rmse:1.84480
[43]	train-rmse:1.52692	test-rmse:1.84173
[44]	train-rmse:1.52067	test-rmse:1.84379
[45]	train-rmse:1.51287	test-rmse:1.84192
[46]	train-rmse:1.50851	test-rmse:1.84221
[47]	train-rmse:1.50263	test-rmse:1.84083
[48]	train-rmse:1.49762	test-rmse:1.84013
[49]	train-rmse:1.49137	test-rmse:1.83967
[50]	train-rmse:1.48665	test-rmse:1.84386
[51]	train-rmse:1.48244	test-rmse:1.84160
[52]	train-rmse:1.47806	test-rmse:1.84307
[53]	train-rmse:1.47377	test-rmse:1.83998
[54]	train-rmse:1.46739	test-rmse:1.83995
[55]	train-rmse:1.46109	test-rmse:1.84036
[56]	train-rmse:1.45640	test-rmse:1.83925
[57]	train-rmse:1.45145	test-rmse:1.84008
```

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:1.91473	test-rmse:1.98394
[1]	train-rmse:1.84329	test-rmse:1.95427
[2]	train-rmse:1.77811	test-rmse:1.93046
[3]	train-rmse:1.71860	test-rmse:1.90982
[4]	train-rmse:1.66479	test-rmse:1.89332
[5]	train-rmse:1.61466	test-rmse:1.87687
[6]	train-rmse:1.56999	test-rmse:1.86128
[7]	train-rmse:1.52944	test-rmse:1.84963
[8]	train-rmse:1.49090	test-rmse:1.84683
[9]	train-rmse:1.45609	test-rmse:1.84133
[10]	train-rmse:1.42424	test-rmse:1.84498
[11]	train-rmse:1.39199	test-rmse:1.83752
[12]	train-rmse:1.36442	test-rmse:1.83296
[13]	train-rmse:1.34046	test-rmse:1.83927
[14]	train-rmse:1.31642	test-rmse:1.83828
[15]	train-rmse:1.29375	test-rmse:1.83726
[16]	train-rmse:1.25895	test-rmse:1.82343
[17]	train-rmse:1.22615	test-rmse:1.81810
[18]	train-rmse:1.19630	test-rmse:1.81181
[19]	train-rmse:1.17092	test-rmse:1.80766
[20]	train-rmse:1.14810	test-rmse:1.80687
[21]	train-rmse:1.12494	test-rmse:1.80619
[22]	train-rmse:1.11308	test-rmse:1.81397
[23]	train-rmse:1.08967	test-rmse:1.81092
[24]	train-rmse:1.06830	test-rmse:1.80862
[25]	train-rmse:1.05870	test-rmse:1.81753
[26]	train-rmse:1.04070	test-rmse:1.82236
[27]	train-rmse:1.03263	test-rmse:1.82949
[28]	train-rmse:1.01780	test-rmse:1.82736
[29]	train-rmse:1.00866	test-rmse:1.83130
[30]	train-rmse:0.99366	test-rmse:1.83235
[31]	train-rmse:0.98755	test-rmse:1.84128
[32]	train-rmse:0.97357	test-rmse:1.84437
[33]	train-rmse:0.96113	test-rmse:1.84764
[34]	train-rmse:0.95580	test-rmse:1.85546
[35]	train-rmse:0.94522	test-rmse:1.85730
[36]	train-rmse:0.93782	test-rmse:1.85898
```

```
[37]	train-rmse:0.92823	test-rmse:1.86215
[38]	train-rmse:0.92028	test-rmse:1.86222
[39]	train-rmse:0.91451	test-rmse:1.86546
[40]	train-rmse:0.90944	test-rmse:1.86837
[41]	train-rmse:0.90148	test-rmse:1.86991
[42]	train-rmse:0.89847	test-rmse:1.87395
[43]	train-rmse:0.89583	test-rmse:1.87977
[44]	train-rmse:0.88917	test-rmse:1.87990
[45]	train-rmse:0.88458	test-rmse:1.88262
[46]	train-rmse:0.88092	test-rmse:1.88395
[47]	train-rmse:0.87764	test-rmse:1.88569
[48]	train-rmse:0.87364	test-rmse:1.88496
[49]	train-rmse:0.87093	test-rmse:1.88772
[50]	train-rmse:0.86671	test-rmse:1.89007
```

```
# 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:1.91473	test-rmse:1.98394
[1]	train-rmse:1.84329	test-rmse:1.95427
[2]	train-rmse:1.77811	test-rmse:1.93046
[3]	train-rmse:1.71860	test-rmse:1.90982
[4]	train-rmse:1.66479	test-rmse:1.89332
[5]	train-rmse:1.61466	test-rmse:1.87687
[6]	train-rmse:1.56999	test-rmse:1.86128
[7]	train-rmse:1.52944	test-rmse:1.84963
[8]	train-rmse:1.49090	test-rmse:1.84683
[9]	train-rmse:1.45609	test-rmse:1.84133
[10]	train-rmse:1.42424	test-rmse:1.84498
[11]	train-rmse:1.39199	test-rmse:1.83752
[12]	train-rmse:1.36442	test-rmse:1.83296
[13]	train-rmse:1.34046	test-rmse:1.83927
[14]	train-rmse:1.31642	test-rmse:1.83828
[15]	train-rmse:1.29375	test-rmse:1.83726
[16]	train-rmse:1.25895	test-rmse:1.82343
[17]	train-rmse:1.22615	test-rmse:1.81810
[18]	train-rmse:1.19630	test-rmse:1.81181
[19]	train-rmse:1.17092	test-rmse:1.80766
[20]	train-rmse:1.14810	test-rmse:1.80687
[21]	train-rmse:1.12494	test-rmse:1.80619
[22]	train-rmse:1.11308	test-rmse:1.81397
[23]	train-rmse:1.08967	test-rmse:1.81092
[24]	train-rmse:1.06830	test-rmse:1.80862
[25]	train-rmse:1.05870	test-rmse:1.81753
[26]	train-rmse:1.04070	test-rmse:1.82236
[27]	train-rmse:1.03263	test-rmse:1.82949
[28]	train-rmse:1.01780	test-rmse:1.82736
[29]	train-rmse:1.00866	test-rmse:1.83130
[30]	train-rmse:0.99366	test-rmse:1.83235
[31]	train-rmse:0.98755	test-rmse:1.84128
[32]	train-rmse:0.97357	test-rmse:1.84437
[33]	train-rmse:0.96113	test-rmse:1.84764
[34]	train-rmse:0.95580	test-rmse:1.85546
[35]	train-rmse:0.94522	test-rmse:1.85730
[36]	train-rmse:0.93782	test-rmse:1.85898
[37]	train-rmse:0.92823	test-rmse:1.86215
[38]	train-rmse:0.92028	test-rmse:1.86222
[39]	train-rmse:0.91451	test-rmse:1.86546
[40]	train-rmse:0.90944	test-rmse:1.86837
[41]	train-rmse:0.90148	test-rmse:1.86991
[42]	train-rmse:0.89847	test-rmse:1.87395
[43]	train-rmse:0.89583	test-rmse:1.87977
[44]	train-rmse:0.88917	test-rmse:1.87990
[45]	train-rmse:0.88458	test-rmse:1.88262
[46]	train-rmse:0.88092	test-rmse:1.88395
[47]	train-rmse:0.87764	test-rmse:1.88569
[48]	train-rmse:0.87364	test-rmse:1.88496
[49]	train-rmse:0.87093	test-rmse:1.88772
[50]	train-rmse:0.86671	test-rmse:1.89007
```

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:1.46712	test-rmse:1.82803
[1]	train-rmse:1.21512	test-rmse:1.83124
[2]	train-rmse:1.03167	test-rmse:1.89367
[3]	train-rmse:0.89959	test-rmse:1.93611
[4]	train-rmse:0.74198	test-rmse:1.91896
[5]	train-rmse:0.67852	test-rmse:1.93280
[6]	train-rmse:0.56360	test-rmse:1.94524
[7]	train-rmse:0.47803	test-rmse:1.96074
[8]	train-rmse:0.41980	test-rmse:1.98434
[9]	train-rmse:0.38515	test-rmse:2.00290
[10]	train-rmse:0.34168	test-rmse:2.00562
[11]	train-rmse:0.31076	test-rmse:2.00561
[12]	train-rmse:0.27368	test-rmse:2.00729
[13]	train-rmse:0.25266	test-rmse:2.00764
[14]	train-rmse:0.23145	test-rmse:2.00373
[15]	train-rmse:0.20843	test-rmse:2.00407
[16]	train-rmse:0.18681	test-rmse:2.00982
[17]	train-rmse:0.17541	test-rmse:2.00823
[18]	train-rmse:0.15662	test-rmse:2.01492
[19]	train-rmse:0.14097	test-rmse:2.00973
[20]	train-rmse:0.12647	test-rmse:2.01161
[21]	train-rmse:0.12227	test-rmse:2.01266
[22]	train-rmse:0.11506	test-rmse:2.01025
[23]	train-rmse:0.10944	test-rmse:2.01001
[24]	train-rmse:0.09750	test-rmse:2.01131
[25]	train-rmse:0.09075	test-rmse:2.01349
[26]	train-rmse:0.08352	test-rmse:2.01706
[27]	train-rmse:0.07612	test-rmse:2.01784
[28]	train-rmse:0.07050	test-rmse:2.01965
[29]	train-rmse:0.06642	test-rmse:2.02089
[30]	train-rmse:0.05929	test-rmse:2.02030
```

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:1.54611	test-rmse:1.83261
[1]	train-rmse:1.21437	test-rmse:1.77350
[2]	train-rmse:0.96740	test-rmse:1.73570
[3]	train-rmse:0.79335	test-rmse:1.74744
[4]	train-rmse:0.66900	test-rmse:1.74173
[5]	train-rmse:0.56030	test-rmse:1.74965
[6]	train-rmse:0.46679	test-rmse:1.77566
[7]	train-rmse:0.41638	test-rmse:1.78293
```

```
[8]	train-rmse:0.34870	test-rmse:1.79056
[9]	train-rmse:0.29881	test-rmse:1.80614
[10]	train-rmse:0.25472	test-rmse:1.81797
[11]	train-rmse:0.23385	test-rmse:1.82658
[12]	train-rmse:0.21009	test-rmse:1.82933
[13]	train-rmse:0.19919	test-rmse:1.82828
[14]	train-rmse:0.18560	test-rmse:1.82773
[15]	train-rmse:0.17054	test-rmse:1.83153
[16]	train-rmse:0.15554	test-rmse:1.83142
[17]	train-rmse:0.13335	test-rmse:1.83289
[18]	train-rmse:0.11370	test-rmse:1.83559
[19]	train-rmse:0.10114	test-rmse:1.83739
[20]	train-rmse:0.09314	test-rmse:1.83613
[21]	train-rmse:0.07741	test-rmse:1.83502
[22]	train-rmse:0.06734	test-rmse:1.83766
[23]	train-rmse:0.05770	test-rmse:1.83961
[24]	train-rmse:0.04833	test-rmse:1.84044
[25]	train-rmse:0.04400	test-rmse:1.84082
[26]	train-rmse:0.03995	test-rmse:1.84152
[27]	train-rmse:0.03498	test-rmse:1.84284
[28]	train-rmse:0.03154	test-rmse:1.84356
[29]	train-rmse:0.02885	test-rmse:1.84414
[30]	train-rmse:0.02513	test-rmse:1.84352
[31]	train-rmse:0.02356	test-rmse:1.84398
[32]	train-rmse:0.02059	test-rmse:1.84420
```

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: (0.05190546764578219, 'bad'),
PBIAS: (8.591316937572678, 'good'),
KGE: (-0.01798982196702381, 'good')

Root Mean Squared Error: 1.8235765571291478
Normalized Root Mean Squared Error: 0.3393331888963803
```

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: (-0.1636895844365409, 'bad'),
PBIAS: (9.536289397180818, 'good'),
KGE: (0.15415693131224129, 'good')

Root Mean Squared Error: 2.0203041368368067
Normalized Root Mean Squared Error: 0.37594047950070836
```

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: 1.8778755992091596
Normalized Root Mean Squared Error: 0.3494372160791142
```

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: 1.8811301123499795
Normalized Root Mean Squared Error: 0.3500428195664272
```

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: (0.03341306908490893, 'bad'),
PBIAS: (10.512910853283234, 'good'),
KGE: (-0.03854732119448778, 'good')

Root Mean Squared Error: 1.9732885466855479
Normalized Root Mean Squared Error: 0.36719176529317976
```

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: (0.11321714547296846, 'bad'),
PBIAS: (10.665690801608564, 'good'),
KGE: (0.38974323879693695, 'good')

Root Mean Squared Error: 1.8900738765764884
Normalized Root Mean Squared Error: 0.3517070853324318
```

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: (0.15573931242290218, 'bad'),
PBIAS: (7.7673086505846225, 'good'),
KGE: (0.4420497909223884, 'good')

Root Mean Squared Error: 1.8442017055350528
Normalized Root Mean Squared Error: 0.3431711398464929
```

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: 2.0152388363385962
Normalized Root Mean Squared Error: 0.37499792265325577
```

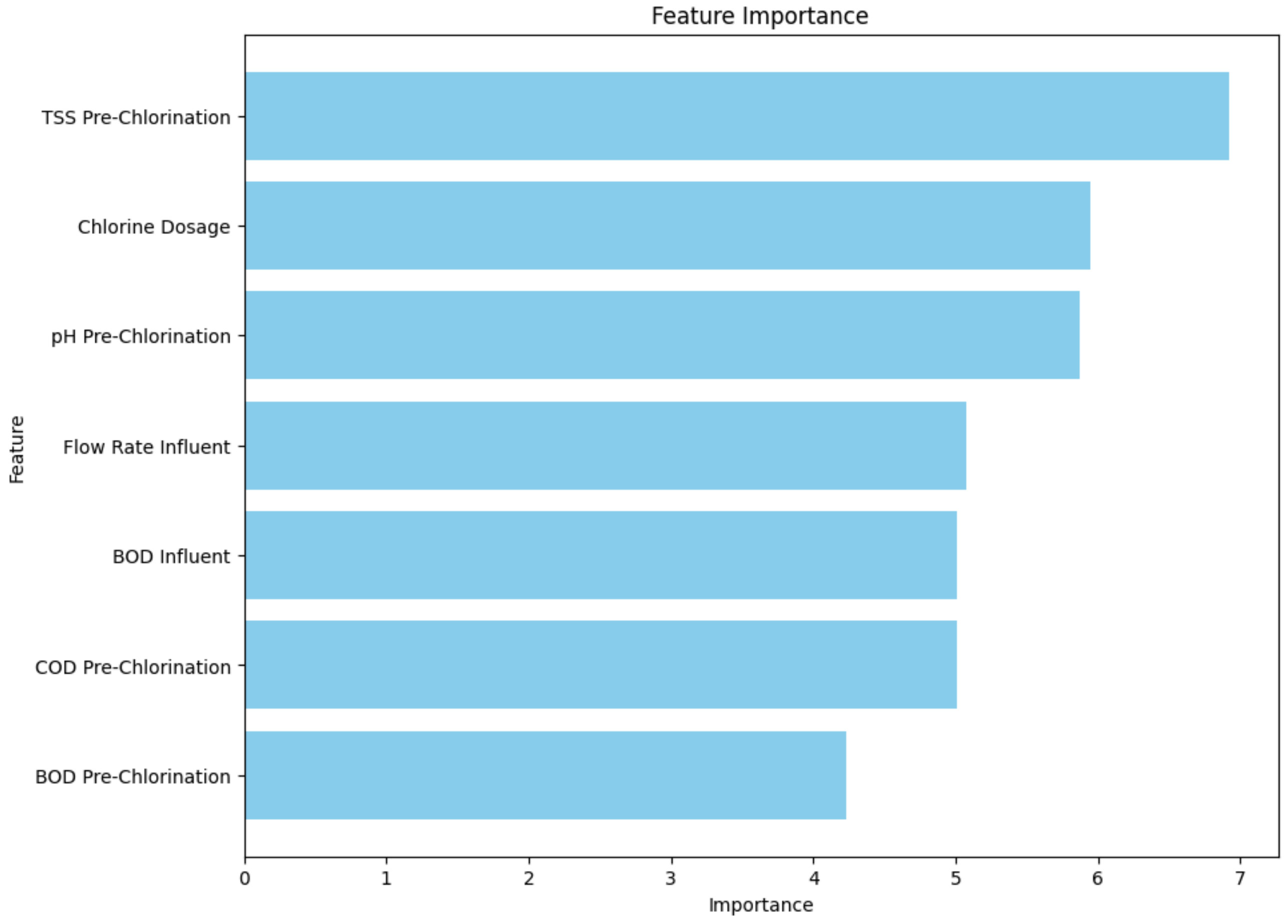
Naive Model 2

```
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: 2.019107272084111
Normalized Root Mean Squared Error: 0.3757177655534259
```

Feature Importance

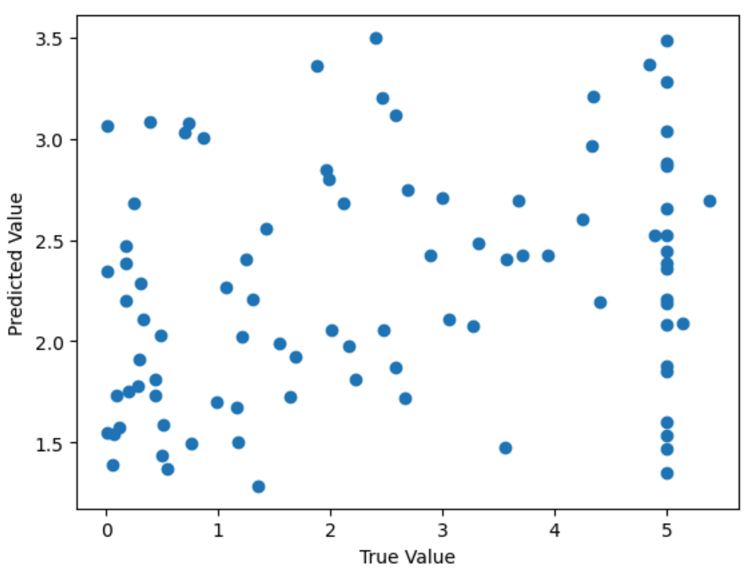
```
# Get feature importance
importance_CAS = final_model_CAS.get_score(importance_type='gain')

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

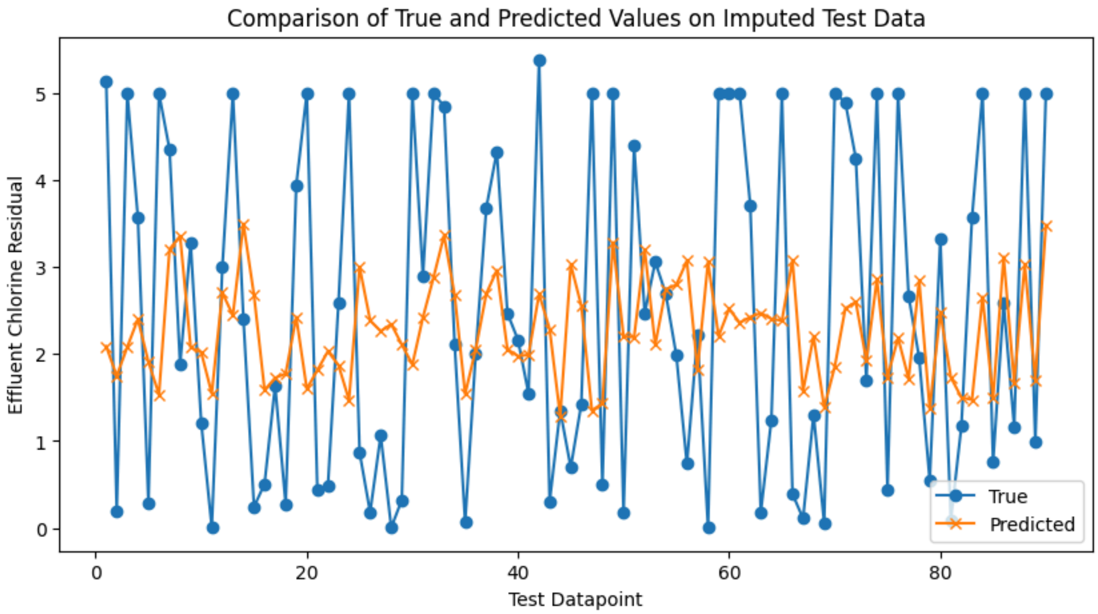
Optimized XGBoost on Imputed Test Dataset

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



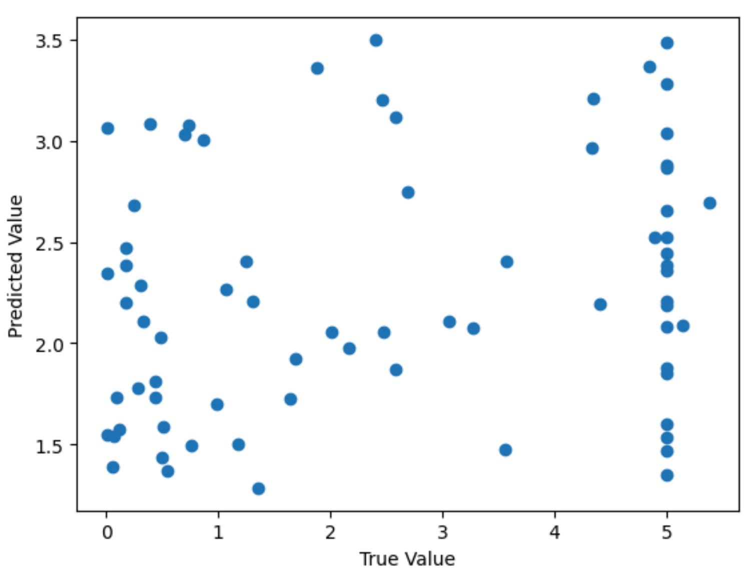

```
# Create an x-axis range based on the length of the series/array
x = range(1, len(y_test_CAS) + 1)
```

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

# Show plot
plt.show()
```



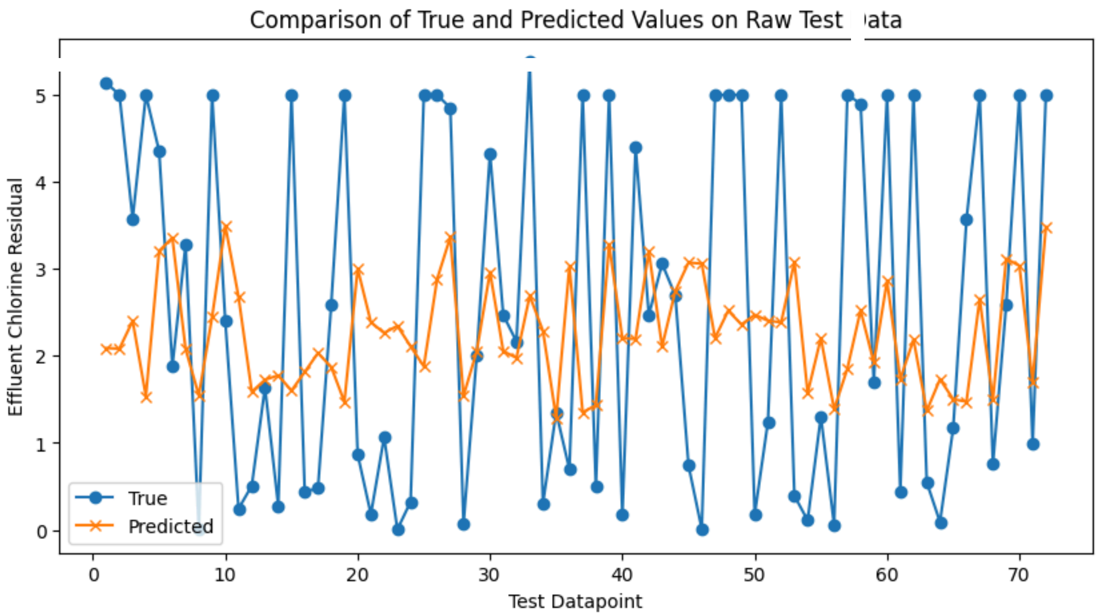
Optimized XGBoost on Non-Imputed (Raw) Test Dataset

```
# without imputation
plt.scatter(y_test_orig_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('Effluent Chlorine Residual')
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)
y_pred_oob_CAS = extend_with_nan(y_pred_oob_CAS, max_length)
y_pred_naive_CAS = extend_with_nan(y_pred_naive_CAS, max_length)
y_pred_naive_orig_CAS = extend_with_nan(y_pred_naive_orig_CAS, max_length)
```