```
import glob
import pandas as pd
import time
import datetime as dt
import os
import numpy as np
import openpyxl
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import pathlib
from pathlib import Path
import pickle
import glob
from datetime import datetime
warnings.filterwarnings("ignore")
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
import math
import plotly.express as px
import sys
from scipy.stats import skew, kurtosis
from scipy.stats import spearmanr
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
from sklearn import preprocessing
from sklearn.model_selection import KFold
import lightgbm as lgb
import optuna
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, expla
from sklearn.metrics import mean_tweedie_deviance
import sys
import umap.umap_ as umap
def mean_absolute_percentage_error(y_true, y_predicted):
    y_true, y_predicted = np.array(y_true), np.array(y_predicted)
    return np.mean(np.abs((y_true - y_predicted) / y_true)) * 100
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import RFE

from sklearn.model_selection import train_test_split, KFold
from sklearn.feature_selection import RFE
```

```
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from tslearn.preprocessing import TimeSeriesScalerMeanVariance
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from datetime import timedelta
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
os.chdir('/content/Raw_Data')


def directional_accuracy(actuals, forecasts):
    correct_directions = np.sum(np.sign(actuals[1:] - actuals[:-1]) == np.sign(forec
    total_directions = len(actuals) - 1
    return correct_directions / total_directions


extension = 'xlsx'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]


Precip_Raw_Data = pd.DataFrame()
i=0
for fn in all_filenames:
    i=i+1
    tmp_df = pd.read_excel(fn,skiprows=7,parse_dates=['Historic Data From : ','Histo
    tmp_df.rename(columns = {'Historic Data From : ':'Timestamp_From', 'Historic Dat
    tmp_df['Date']=tmp_df['Timestamp_From'].dt.date
    tmp_df['Date'] = pd.to_datetime(tmp_df.Date)
    tmp_df.columns = tmp_df.columns.str.replace('-Precipitation_Section', '')
    tmp_df.columns = tmp_df.columns.str.replace(' ', '_')
    print("Reading file ",i," :",fn," | ",tmp_df.shape)
    Precip_Raw_Data = pd.concat([Precip_Raw_Data, tmp_df], ignore_index=True)
    del (tmp_df)
```

```
Reading file  1  : compare-parameter-11_Sep_2023_13_13_04.xlsx  |  (717, 34)
Reading file  2  : compare-parameter-11_Sep_2023_13_13_44.xlsx  |  (24, 34)
Reading file  3  : compare-parameter-11_Sep_2023_13_03_37.xlsx  |  (713, 34)
Reading file  4  : compare-parameter-20_Dec_2023_22_47_14.xlsx  |  (512, 34)
Reading file  5  : compare-parameter-23_Jan_2024_15_24_32.xlsx  |  (720, 34)
Reading file  6  : compare-parameter-11_Sep_2023_13_15_02.xlsx  |  (720, 34)
Reading file  7  : compare-parameter-11_Sep_2023_13_14_28.xlsx  |  (718, 34)
Reading file  8  : compare-parameter-16_Oct_2023_11_54_09.xlsx  |  (720, 34)
Reading file  9  : compare-parameter-28_Nov_2023_17_20_45.xlsx  |  (168, 34)
Reading file  10  : compare-parameter-11_Sep_2023_13_09_52.xlsx  |  (720, 34)
Reading file  11  : compare-parameter-11_Sep_2023_13_04_24.xlsx  |  (672, 34)
Reading file  12  : compare-parameter-11_Sep_2023_13_17_50.xlsx  |  (720, 34)
Reading file  13  : compare-parameter-11_Sep_2023_13_04_01.xlsx  |  (24, 34)
Reading file  14  : compare-parameter-11_Sep_2023_13_15_27.xlsx  |  (24, 34)
Reading file  15  : compare-parameter-11_Sep_2023_13_18_14.xlsx  |  (24, 34)
Reading file  16  : compare-parameter-24_Nov_2023_09_56_08.xlsx  |  (528, 34)
Reading file  17  : compare-parameter-11_Sep_2023_13_10_20.xlsx  |  (24, 34)
```

```
Reading file  18  : compare-parameter-11_Sep_2023_13_11_44.xlsx  |  (720, 34)
Reading file  19  : compare-parameter-30_Oct_2023_15_23_59.xlsx  |  (696, 34)
```

```
Precip_Raw_Data=Precip_Raw_Data.sort_values(by = ['Timestamp_From','Timestamp_To'])
Precip_Raw_Data.reset_index(drop = True,inplace =True)
```

```
Precip_Raw_Data.shape
```

```
(9164, 34)
```

```
Precip_Raw_Data.to_csv("/content/Precip_Raw_Data.csv",index=False)
```

```
Agg_data=Precip_Raw_Data.copy()
```

```
Agg_data['Timestamp_To'].max()
```

```
Timestamp('2024-01-19 22:00:00')
```

```
Agg_data.tail()
```

	Timestamp_From	Timestamp_To	Special_Filling_temp	Special_Filling_Flow	F
9159	2024-01-19 17:00:00	2024-01-19 18:00:00	97.01	137.22	
9160	2024-01-19 18:00:00	2024-01-19 19:00:00	96.00	141.38	
9161	2024-01-19 19:00:00	2024-01-19 20:00:00	96.38	149.86	
9162	2024-01-19 20:00:00	2024-01-19 21:00:00	96.02	140.27	
9163	2024-01-19 21:00:00	2024-01-19 22:00:00	95.01	174.21	

```
Agg_data.drop(['Timestamp_From','Date','Fine_Seed_Charge','Batch_Circulation_Hrs','T
```

```
final_Agg_data=Agg_data[['Timestamp_To','Special_Filling_Flow','Fine_Seed_Tonnage','
                         'Fine_Seed_SSA','Fine_Seed_3_5u','Fine_Seed_45u','Fine_Seed_
                         'Coarse_Seed_1_Tonnage','Coarse_Seed_2_Flow','HAT_3_5u','HAT
```

```
def replace_outliers_with_median(data):
    Q1 = data.quantile(0.25)
```

```
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    median = data.median()
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data = data.apply(lambda x: x.where((x.between(lower_bound[x.name], upper_bound[

    return data



treated_agg_data = replace_outliers_with_median(final_Agg_data)


min_timestamp = treated_agg_data['Timestamp_To'].min()
print(min_timestamp)
timestamp_1hr_before = min_timestamp - pd.Timedelta(hours=1)
timestamp_2hr_before = min_timestamp - pd.Timedelta(hours=2)
```

```
2023-01-01 01:00:00
```

```
new_rows_dynamic = [
    {col: timestamp_1hr_before if col == 'Timestamp_To' else pd.NA for col in treate
    {col: timestamp_2hr_before if col == 'Timestamp_To' else pd.NA for col in treate
]

new_rows_df_dynamic = pd.DataFrame(new_rows_dynamic)


data_extended_dynamic = pd.concat([new_rows_df_dynamic, treated_agg_data], ignore_in
data_extended_dynamic.sort_values(by='Timestamp_To', inplace=True)
data_extended_dynamic.reset_index(drop=True, inplace=True)


columns_to_aggregate = ['Special_Filling_temp','Special_Filling_Flow','First_Agglo_T
                        'Fine_Seed_Tonnage','Fine_Seed_Flow','Fine_Seed_Density',
                       'PHE_Inlet_Temp','PHE_Outlet_Temp','Regular_Filling_Flow','Ta
                        'Coarse_Seed_1_Tonnage','Coarse_Seed_2_Tonnage','Coarse_Seed
                        'Coarse_Seed_2_Density',]
columns_to_resample = ['Fine_Seed_SSA','Fine_Seed_3_5u','Fine_Seed_45u','Fine_Seed_D
                       'Filling_Concentration','Filling_Ratio','Feed_Hydrate_D50','F


resample_frequency = '8H'
agg_functions = {
    'min': 'min',
    'max': 'max',
    'median': 'median',
    'mean': 'mean',
    'std': 'std',
    'var': 'var',
```

```
        'var_coeff': lambda x: x.std() / x.mean() if x.mean() != 0 else 0
    }


    df=data_extended_dynamic.copy()
    df['Timestamp_To'] = df['Timestamp_To'] + pd.Timedelta(hours=1)
    df.set_index('Timestamp_To', inplace=True)


    resampled_df = pd.DataFrame()

    for feature in df.columns:
        if feature in columns_to_aggregate:
            for agg_name, agg_func in agg_functions.items():
                new_feature_name = f'{feature}_{agg_name}'
                resampled_df[new_feature_name] = df[feature].resample(resample_frequency
        elif feature in columns_to_resample:
            resampled_df[feature] = df[feature].resample(resample_frequency).mean()


    resampled_df.reset_index(inplace=True)
    resampled_df['Timestamp_To'] = resampled_df['Timestamp_To'] + pd.Timedelta(hours=6)


    resampled_df = resampled_df.interpolate(method='linear',
                                            limit_direction='forward')
    resampled_df = resampled_df.interpolate(method='linear',
                                            limit_direction='backward')


    final_df=resampled_df.copy()


    columns_to_drop=[col for col in final_df.columns if col != "Feed_Hydrate_D50" and co


    final_df.set_index('Timestamp_To', inplace=True)


    df_lag = final_df.copy()
    base_columns = [
        "Fine_Seed_Tonnage",
        "Fine_Seed_Flow",
        "Fine_Seed_Density",
        "Fine_Seed_SSA",
        "Fine_Seed_3_5u",
        "Fine_Seed_45u",
        "Fine_Seed_D50",
        "PHE_Inlet_Temp",
        "Regular_Filling_Flow",
        "Coarse_Seed_1_Tonnage",
        "Coarse_Seed_2_Flow",
        "HAT_3_5u",
        "HAT_45u",
```

```
    "HAT_D50",
    "Filling_Ratio"
]


special_lag_ranges = {
    "Fine_Seed_SSA": range(47, 84),
    "Fine_Seed_3_5u": range(47, 84),
    "Coarse_Seed_1_Tonnage": range(39, 61),
    "Coarse_Seed_2_Flow": range(39, 61)
}

filtered_columns = [col for col in final_df.columns for base_col in base_columns if


for column_name in filtered_columns:

    base_column_name = next(base_col for base_col in base_columns if base_col in col

    lag_range = special_lag_ranges.get(base_column_name, range(39, 84))

    for shift_amount in lag_range:
        feature_name = f'{column_name}_shift_{shift_amount}'
        df_lag[feature_name] = final_df[column_name].shift(shift_amount)



df_lag = df_lag.drop(columns=columns_to_drop)


df_lag_cleaned = df_lag.dropna()


best_cols = '/content/precip_final_cols.pkl'

with open(best_cols, 'rb') as file:
    final_cols = pickle.load(file)


final_cols
```

```
['Fine_Seed_Tonnage_min_shift_43',
 'Fine_Seed_Tonnage_min_shift_50',
 'Fine_Seed_Tonnage_min_shift_59',
 'Fine_Seed_Tonnage_min_shift_61',
 'Fine_Seed_Tonnage_max_shift_42',
 'Fine_Seed_Tonnage_max_shift_62',
 'Fine_Seed_Tonnage_median_shift_42',
 'Fine_Seed_Flow_max_shift_62',
 'Fine_Seed_Density_mean_shift_74',
 'Fine_Seed_Density_mean_shift_76',
 'Fine_Seed_Density_mean_shift_82',
 'Fine_Seed_Density_mean_shift_83',
 'Fine_Seed_Density_var_coeff_shift_60',
```

```
        'Fine_Seed_SSA_shift_57',
        'Fine_Seed_SSA_shift_70',
        'Fine_Seed_45u_shift_39',
        'Fine_Seed_45u_shift_45',
        'Fine_Seed_45u_shift_55',
        'Fine_Seed_45u_shift_57',
        'Fine_Seed_45u_shift_77',
        'Fine_Seed_45u_shift_81',
        'Fine_Seed_45u_shift_83',
        'Fine_Seed_D50_shift_61',
        'Fine_Seed_D50_shift_66',
        'Fine_Seed_D50_shift_82',
        'PHE_Inlet_Temp_min_shift_41',
        'PHE_Inlet_Temp_min_shift_73',
        'PHE_Inlet_Temp_std_shift_42',
        'PHE_Inlet_Temp_std_shift_52',
        'PHE_Inlet_Temp_std_shift_54',
        'PHE_Inlet_Temp_std_shift_59',
        'PHE_Inlet_Temp_std_shift_62',
        'PHE_Inlet_Temp_std_shift_78',
        'Regular_Filling_Flow_min_shift_42',
        'Regular_Filling_Flow_min_shift_45',
        'Regular_Filling_Flow_min_shift_50',
        'Regular_Filling_Flow_max_shift_39',
        'Regular_Filling_Flow_max_shift_40',
        'Regular_Filling_Flow_max_shift_42',
        'Regular_Filling_Flow_max_shift_43',
        'Regular_Filling_Flow_max_shift_44',
        'Regular_Filling_Flow_max_shift_45',
        'Regular_Filling_Flow_max_shift_48',
        'Regular_Filling_Flow_max_shift_51',
        'Regular_Filling_Flow_max_shift_57',
        'Regular_Filling_Flow_max_shift_70',
        'Regular_Filling_Flow_max_shift_77',
        'Regular_Filling_Flow_max_shift_82',
        'Regular_Filling_Flow_median_shift_40',
        'Regular_Filling_Flow_median_shift_41',
        'Regular_Filling_Flow_median_shift_42',
        'Regular_Filling_Flow_median_shift_44',
        'Regular_Filling_Flow_median_shift_52',
        'Regular_Filling_Flow_median_shift_57',
        'Regular_Filling_Flow_mean_shift_39',
        'Regular_Filling_Flow_std_shift_39',
        'Regular_Filling_Flow_std_shift_41',
        'Regular_Filling_Flow_std_shift_42',
```

```
filtered_cols = [col for col in final_cols if col != "Feed_Hydrate_D50" and int(col.

filtered_cols.append("Feed_Hydrate_D50")


len(filtered_cols)
```

```
70
```

```
df_final_cleaned=df_lag_cleaned[filtered_cols]


def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).m
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).m
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).m
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16)
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float3
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df


df_final_cleaned_optimized = reduce_mem_usage(df_final_cleaned)
```

```
Memory usage of dataframe is 0.58 MB
Memory usage after optimization is: 0.15 MB
Decreased by 73.9%
```

```
target_column ='Feed_Hydrate_D50'
```

```
max_date = df_final_cleaned.index.max()


cutoff_date = max_date - timedelta(days=15)


train_df = df_final_cleaned[df_final_cleaned.index <= cutoff_date]
test_df = df_final_cleaned[df_final_cleaned.index > cutoff_date]


print(train_df.shape, test_df.shape)
```

```
(1024, 70) (45, 70)
```

```
print(train_df.index.max(),test_df.index.min(),test_df.index.max())
```

```
2024-01-04 22:00:00 2024-01-05 06:00:00 2024-01-19 22:00:00
```

```
X_train = train_df.drop([target_column],axis=1).copy()
y_train = train_df[[target_column]].copy()


X_test= test_df.drop([target_column],axis=1).copy()
y_test=test_df[[target_column]].copy()


print('X_train shape',X_train.shape)
print('y_train shape',y_train.shape)
print('X_test shape',X_test.shape)
print('y_test shape',y_test.shape)
```

```
X_train shape (1024, 69)
y_train shape (1024, 1)
X_test shape (45, 69)
y_test shape (45, 1)
```

```
best_params={'max_depth': 10,
 'max_bin': 487,
 'n_estimators': 2292,
 'num_leaves': 415,
 'min_child_weight': 8.432817690045072,
 'learning_rate': 0.04218575822063664,
 'subsample': 0.5008507518604671,
 'colsample_bytree': 0.4055821572424323,
 'reg_lambda': 1.9337373228621815}


reg = lgb.LGBMRegressor(objective='regression',
                        boosting_type = 'gbdt',
                        **best_params)
```

```
    reg.fit(X_train.values, y_train.values.ravel(), callbacks=[lgb.log_evaluation(period
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testin
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 18919
[LightGBM] [Info] Number of data points in the train set: 1024, number of used f
[LightGBM] [Info] Start training from score 94.450195
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
preds = reg.predict(X_test.values)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
Mape_test = mean_absolute_percentage_error(y_test[target_column],preds)
print(Mape_test)
MSE_test = mean_squared_error(y_test[target_column],preds)
RMSE_test = np.sqrt(MSE_test)
print(RMSE_test)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
7.304279308409371
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
8.893302585335313
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
test1=y_test.copy()
test1['Feed_Hydrate_D50_pred']=preds
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
test1.head(50)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Timestamp_To	Feed_Hydrate_D50	Feed_Hydrate_D50_pred
2024-01-05 06:00:00	106.0000	105.847750
2024-01-05 14:00:00	107.0000	106.458166
2024-01-05 22:00:00	110.0000	106.868174
2024-01-06 06:00:00	111.0000	103.479417
2024-01-06 14:00:00	108.0000	106.519631
2024-01-06 22:00:00	97.8125	106.857678
2024-01-07 06:00:00	104.0000	107.660279
2024-01-07 14:00:00	111.0000	107.938649
2024-01-07 22:00:00	113.0000	105.259103
2024-01-08 06:00:00	117.0000	105.002562
2024-01-08 14:00:00	112.0000	106.608919
2024-01-08 22:00:00	114.0000	107.545367
2024-01-09 06:00:00	116.0000	106.041116
2024-01-09 14:00:00	118.0000	108.063649
2024-01-09 22:00:00	112.0000	107.334643
2024-01-10 06:00:00	114.0000	106.568858
2024-01-10 14:00:00	116.0000	106.656223
2024-01-10 22:00:00	117.0000	107.653197
2024-01-11 06:00:00	113.0000	104.113773
2024-01-11 14:00:00	114.0000	105.552659
2024-01-11 22:00:00	115.0000	107.095707
2024-01-12 06:00:00	93.6875	106.194136
2024-01-12 14:00:00	115.0000	103.891453
2024-01-12 22:00:00	114.0000	108.379359
2024-01-13 06:00:00	117.0000	105.773227
2024-01-13 14:00:00	109.1250	108.244674
2024-01-13 22:00:00	113.8125	108.599447
2024-01-14 06:00:00	115.3750	106.151477
2024-01-14 14:00:00	93.6875	109.496948

2024-01-14 22:00:00	116.6250	105.612936
2024-01-15 06:00:00	93.6875	109.215820
2024-01-15 14:00:00	115.1250	108.042109
2024-01-15 22:00:00	93.6875	109.522890
2024-01-16 06:00:00	117.8125	106.059060
2024-01-16 14:00:00	93.6875	106.348973
2024-01-16 22:00:00	115.5000	105.277175
2024-01-17 06:00:00	115.3125	106.405036
2024-01-17 14:00:00	115.1250	108.604016
2024-01-17 22:00:00	93.6875	106.498216
2024-01-18 06:00:00	118.1250	107.686630
2024-01-18 14:00:00	116.6250	107.019242
2024-01-18 22:00:00	111.1875	108.529254
2024-01-19 06:00:00	117.3750	109.298385
2024-01-19 14:00:00	115.0000	108.177428
2024-01-19 22:00:00	109.0000	107.559549

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

def find_best_hyperparams_LGB(trainX, trainY, num_train, n_trials, cores):

    tscv_folds = TimeSeriesSplit(n_splits=num_train)

    def objective(trial):
        # Hyperparameter settings
        params = {
            'max_depth': trial.suggest_int('max_depth', 2, 12),
            'max_bin': trial.suggest_int('max_bin', 10, 500),
            'n_estimators': trial.suggest_int('n_estimators', 10, 5000),
            'num_leaves': trial.suggest_int('num_leaves', 5, 4075),
            'min_child_weight': trial.suggest_loguniform('min_child_weight', 1, 500)
            'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.9),
            'subsample': trial.suggest_loguniform('subsample', 0.4, 1),
            'colsample_bytree': trial.suggest_loguniform('colsample_bytree', 0.4, 1)
            'reg_lambda': trial.suggest_loguniform('reg_lambda', 0.01, 50),
            'random_state': 100,
            'objective': 'regression',
            'boosting_type': 'gbdt',
            'metric': 'rmse',
            'verbosity': 0
        }

        rmse_scores = []
        da_scores = []

        for train_index, test_index in tscv_folds.split(trainX):
            Major_X, minor_X = trainX.iloc[train_index, :], trainX.iloc[test_index,
            Major_Y, minor_Y = trainY.iloc[train_index], trainY.iloc[test_index]

            # Train model
            reg = lgb.LGBMRegressor(**params)
            reg.fit(Major_X.values, Major_Y.values.ravel(), eval_set=[(minor_X.value

            # Predictions and evaluation
            preds = reg.predict(minor_X)
            rmse_scores.append(np.sqrt(mean_squared_error(minor_Y, preds)))
            da_scores.append(directional_accuracy(minor_Y.values.ravel(), preds))

        # Combine RMSE and directional accuracy
        avg_rmse = np.mean(rmse_scores)
        avg_da = np.mean(da_scores)
        return avg_rmse * (1 - avg_da)

    study = optuna.create_study(direction='minimize')
    study.optimize(objective, n_trials=n_trials, n_jobs=cores[1])

    best_params = study.best_params
    return [study.best_value, best_params]

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
def find_best_hyperparams_LGB(trainX, trainY, num_train, n_trials, cores):

    tscv_folds = TimeSeriesSplit(n_splits=num_train)

    def objective(trial):
        # Hyperparameter settings
        params = {
            'max_depth': trial.suggest_int('max_depth', 2, 12),
            'max_bin': trial.suggest_int('max_bin', 10, 500),
            'n_estimators': trial.suggest_int('n_estimators', 10, 5000),
            'num_leaves': trial.suggest_int('num_leaves', 5, 4075),
            'min_child_weight': trial.suggest_loguniform('min_child_weight', 1, 500)
            'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.9),
            'subsample': trial.suggest_loguniform('subsample', 0.4, 1),
            'colsample_bytree': trial.suggest_loguniform('colsample_bytree', 0.4, 1)
            'reg_lambda': trial.suggest_loguniform('reg_lambda', 0.01, 50),
            'random_state': 100,
            'objective': 'regression',
            'boosting_type': 'gbdt',
            'metric': 'rmse',
            'verbosity': 0
        }

        rmse_scores = []
        da_scores = []

        for train_index, test_index in tscv_folds.split(trainX):
            Major_X, minor_X = trainX.iloc[train_index, :], trainX.iloc[test_index,
            Major_Y, minor_Y = trainY.iloc[train_index], trainY.iloc[test_index]

            # Train model
            reg = lgb.LGBMRegressor(**params)
            # Use callbacks for early stopping
            reg.fit(Major_X.values, Major_Y.values.ravel(), eval_set=[(minor_X.value

            # Predictions and evaluation
            preds = reg.predict(minor_X)
            rmse_scores.append(np.sqrt(mean_squared_error(minor_Y, preds)))
            da_scores.append(directional_accuracy(minor_Y.values.ravel(), preds))

        # Combine RMSE and directional accuracy
        avg_rmse = np.mean(rmse_scores)
        avg_da = np.mean(da_scores)
        return avg_rmse * (1 - avg_da)

    study = optuna.create_study(direction='minimize')
    study.optimize(objective, n_trials=n_trials, n_jobs=cores[1])

    best_params = study.best_params
    return [study.best_value, best_params]
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
best_params
```

```
{'max_depth': 10,
 'max_bin': 487,
 'n_estimators': 2292,
 'num_leaves': 415,
 'min_child_weight': 0.43281769045072,
 'learning_rate': 0.042185758220663664,
 'subsample': 0.5008507518604671,
 'colsample_bytree': 0.40558215724243233,
 'reg_lambda': 1.0337873228621815}
```

```
best_params={'max_depth': 2,
 'max_bin': 252,
 'n_estimators': 2304,
 'num_leaves': 3774,
 'min_child_weight': 100.07002845692435,
 'learning_rate': 0.012399429737718884,
 'subsample': 0.4710719295644458,
 'colsample_bytree': 0.8123534274930037,
 'reg_lambda': 0.1269832447852094}
```

```
reg = lgb.LGBMRegressor(objective='regression',
                        boosting_type = 'gbdt',
                        **best_params)
```

```
reg.fit(X_train.values , y_train.values.ravel(), callbacks=[lgb.log_evaluation(perio
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testin
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Total Bins 16074
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Number of data points in the train set: 1024, number of used f
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 94.450195
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```