```
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, explained_variance_score
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')


import pandas as pd

import numpy as np

data = np.genfromtxt('/content/Precip_Raw_Data.csv', delimiter=',', dtype=str, encoding='utf-8')
Agg_data = pd.DataFrame(data[1:], columns=data[0])  # Adjust header if necessary



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

```
'2024-01-19 22:00:00'
```

```
Agg_data.tail()
```

	Timestamp_From	Timestamp_To	Special_Filling_temp	Special_Filling_Flow	First_Agglo_Tank_Temp_T42	Tank_47_Temp	48_Te
9159	2024-01-19 17:00:00	2024-01-19 18:00:00	97.01	137.22	82.49	77.44	
9160	2024-01-19 18:00:00	2024-01-19 19:00:00	96.0	141.38	82.39	77.46	
9161	2024-01-19 19:00:00	2024-01-19 20:00:00	96.38	149.86	82.53	77.56	
9162	2024-01-19 20:00:00	2024-01-19 21:00:00	96.02	140.27	82.31	77.53	
9163	2024-01-19 21:00:00	2024-01-19 22:00:00	95.01	174.21	82.15	77.68	

```
Agg_data.drop(['Timestamp_From','Date','Fine_Seed_Charge','Batch_Circulation_Hrs','Tank_47_Temp','PPT-48_Temperature'], axis = 1


final_Agg_data=Agg_data[['Timestamp_To','Special_Filling_Flow','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','Feed_Hydrat


def replace_outliers_with_median(data):
    # Convert relevant columns to numeric before calculating quantiles
    numeric_cols = data.select_dtypes(include=np.number).columns
    # Convert only numeric_cols to numeric, excluding Timestamp_To
    data[numeric_cols] = data[numeric_cols].apply(pd.to_numeric, errors='coerce')

    # Ensure 'Timestamp_To' is excluded from quantile calculations
    numeric_data = data.select_dtypes(include=np.number)

    Q1 = numeric_data.quantile(0.25)  # Apply quantile to numeric_data
    Q3 = numeric_data.quantile(0.75)  # Apply quantile to numeric_data
    IQR = Q3 - Q1
    median = numeric_data.median()  # Calculate median for numeric_data
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Apply outlier replacement only to numeric columns
    for col in numeric_cols:
        data[col] = data[col].where((data[col].between(lower_bound[col], upper_bound[col])) | data[col].isna(), median[col])

    return data


treated_agg_data = replace_outliers_with_median(final_Agg_data)


min_timestamp = treated_agg_data['Timestamp_To'].min()
print(min_timestamp)
# Convert min_timestamp to datetime object
min_timestamp = pd.to_datetime(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 treated_agg_data.columns},
    {col: timestamp_2hr_before if col == 'Timestamp_To' else pd.NA for col in treated_agg_data.columns}
]

new_rows_df_dynamic = pd.DataFrame(new_rows_dynamic)


data_extended_dynamic = pd.concat([new_rows_df_dynamic, treated_agg_data], ignore_index=True)
# Convert 'Timestamp_To' to datetime objects before sorting
data_extended_dynamic['Timestamp_To'] = pd.to_datetime(data_extended_dynamic['Timestamp_To'])
data_extended_dynamic.sort_values(by='Timestamp_To', inplace=True)
data_extended_dynamic.reset_index(drop=True, inplace=True)
```

```
def initial_data_preprocessing(treated_agg_data):
    # Convert 'Timestamp_To' to datetime objects before finding the minimum
    treated_agg_data['Timestamp_To'] = pd.to_datetime(treated_agg_data['Timestamp_To'])
    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)
    new_rows_dynamic = [
    {col: timestamp_1hr_before if col == 'Timestamp_To' else pd.NA for col in treated_agg_data.columns},
    {col: timestamp_2hr_before if col == 'Timestamp_To' else pd.NA for col in treated_agg_data.columns}]
    new_rows_df_dynamic = pd.DataFrame(new_rows_dynamic)
    data_extended_dynamic = pd.concat([new_rows_df_dynamic, treated_agg_data], ignore_index=True)
    data_extended_dynamic.sort_values(by='Timestamp_To', inplace=True)
    data_extended_dynamic.reset_index(drop=True, inplace=True)
    data_extended_dynamic['Timestamp_To'] = data_extended_dynamic['Timestamp_To'] + pd.Timedelta(hours=1)
    data_extended_dynamic.set_index('Timestamp_To', inplace=True)

    return data_extended_dynamic


df=initial_data_preprocessing(treated_agg_data)
```

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

```
columns_to_aggregate = ['Special_Filling_temp','Special_Filling_Flow','First_Agglo_Tank_Temp_T42',
                        'Fine_Seed_Tonnage','Fine_Seed_Flow','Fine_Seed_Density',
                       'PHE_Inlet_Temp','PHE_Outlet_Temp','Regular_Filling_Flow','Tank__52_Temp','Tank__56_Temp',
                        'Coarse_Seed_1_Tonnage','Coarse_Seed_2_Tonnage','Coarse_Seed_2_Flow','Coarse_Seed_1_Density',
                        'Coarse_Seed_2_Density',]
columns_to_resample = ['Fine_Seed_SSA','Fine_Seed_3_5u','Fine_Seed_45u','Fine_Seed_D50','HAT_3_5u','HAT_45u','HAT_D50',
                       'Filling_Concentration','Filling_Ratio','Feed_Hydrate_D50','Feed_Hydrate_SSA']


def data_aggregation(df):
    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}'
                # Convert the column to numeric before applying aggregation
                # errors='coerce' converts invalid values to NaN
                resampled_df[new_feature_name] = pd.to_numeric(df[feature], errors='coerce').resample(resample_frequency).apply(
        elif feature in columns_to_resample:
            # Convert the column to numeric before applying aggregation
            # errors='coerce' converts invalid values to NaN
            resampled_df[feature] = pd.to_numeric(df[feature], errors='coerce').resample(resample_frequency).mean()
    resampled_df.reset_index(inplace=True)
    resampled_df['Timestamp_To'] = resampled_df['Timestamp_To'] + pd.Timedelta(hours=6)
    return resampled_df


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
}


resampled_df=data_aggregation(df)


resampled_df.tail()
```

	Timestamp_To	Special_Filling_Flow_min	Special_Filling_Flow_max	Special_Filling_Flow_median	Special_Filling_Flow_mea
1147	2024-01-18 14:00:00	137.86	234.90	204.05	195.2950
1148	2024-01-18 22:00:00	126.50	146.69	131.74	132.6725
1149	2024-01-19 06:00:00	123.03	152.11	131.57	133.9587
1150	2024-01-19 14:00:00	125.50	144.22	134.76	133.9950
1151	2024-01-19 22:00:00	137.22	174.21	141.09	145.7337

```
def data_filter(resampled_df):
    resampled_df_sorted = resampled_df.sort_values(by='Timestamp_To', ascending=False)
    for index, row in resampled_df_sorted.head(5).iterrows():
        if pd.isna(row['Feed_Hydrate_D50']):
            resampled_df_sorted = resampled_df_sorted.drop(index)
    resampled_df = resampled_df_sorted.sort_values(by='Timestamp_To')
    return resampled_df


filtered_df=data_filter(resampled_df)


filtered_df=filtered_df[filtered_df['Timestamp_To']<="2024-01-04 22:00:00"]


filtered_df.tail()
```

	Timestamp_To	Special_Filling_Flow_min	Special_Filling_Flow_max	Special_Filling_Flow_median	Special_Filling_Flow_mea
1102	2024-01-03 14:00:00	164.30	232.93	183.920	195.8462
1103	2024-01-03 22:00:00	150.07	187.79	164.255	167.3487
1104	2024-01-04 06:00:00	160.16	206.59	176.500	178.4262
1105	2024-01-04 14:00:00	155.95	206.67	172.235	176.6575
1106	2024-01-04 22:00:00	180.34	215.56	194.300	196.0312

```
print(resampled_df.shape)
print(filtered_df.shape)
```

```
(1152, 66)
(1107, 66)
```

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


final_df=filtered_df.copy()


max_timestamp = final_df['Timestamp_To'].max()
max_timestamp_dt = pd.to_datetime(max_timestamp)
future_timestamp = max_timestamp_dt + timedelta(days=15)
target_date_str = future_timestamp.strftime("%Y-%m-%d %H:%M")


print(future_timestamp)
```

```
2024-01-19 22:00:00
```

```
min_shifts = 45
max_shifts = 83
shifts_per_day = 3


def calculate_lag_dates(target_date_str, min_shifts, max_shifts, shifts_per_day):
    target_date = datetime.strptime(target_date_str, "%Y-%m-%d %H:%M")
    min_lag_days = min_shifts // shifts_per_day
    max_lag_days = max_shifts // shifts_per_day
    max_lag_extra_shifts = max_shifts % shifts_per_day
    start_date_min_lag = target_date - timedelta(days=min_lag_days)
    start_date_max_lag = target_date - timedelta(days=max_lag_days, hours=max_lag_extra_shifts * 8)
    return start_date_min_lag, start_date_max_lag



start_date_min_lag, start_date_max_lag = calculate_lag_dates(target_date_str, min_shifts, max_shifts, shifts_per_day)


print(start_date_min_lag, start_date_max_lag)
```

```
2024-01-04 22:00:00 2023-12-23 06:00:00
```

```
print("Start Date for Minimum Lag:", start_date_min_lag.strftime("%Y-%m-%d %H:%M"))
print("Start Date for Maximum Lag:", start_date_max_lag.strftime("%Y-%m-%d %H:%M"))
```

```
Start Date for Minimum Lag: 2024-01-04 22:00
Start Date for Maximum Lag: 2023-12-23 06:00
```

```
final_df_lag = final_df[(final_df['Timestamp_To'] <= start_date_min_lag) & (final_df['Timestamp_To'] >= start_date_max_lag)]


final_df_lag.shape
```

```
(39, 66)
```

```
final_df_lag.head()
```

```
df=final_df_lag.copy()
df.set_index('Timestamp_To', inplace=True)


for column in df.columns:
    for i in range(45,84):
        nth_last_value = df[column].iloc[-(i - 44)]
        new_column_name = f'{column}_shift_{i}'
        df[new_column_name] = pd.NA
        df.at[df.index[-1],new_column_name] = nth_last_value


final_df_lag.head()
```

```
final_df_lag.tail()
```

	Timestamp_To	Special_Filling_Flow_min	Special_Filling_Flow_max	Special_Filling_Flow_median	Special_Filling_Flow_mea
1102	2024-01-03 14:00:00	164.30	232.93	183.920	195.8462
1103	2024-01-03 22:00:00	150.07	187.79	164.255	167.3487
1104	2024-01-04 06:00:00	160.16	206.59	176.500	178.4262
1105	2024-01-04 14:00:00	155.95	206.67	172.235	176.6575
1106	2024-01-04 22:00:00	180.34	215.56	194.300	196.0312

```
df.head()
```

Timestamp_To	Special_Filling_Flow_min	Special_Filling_Flow_max	Special_Filling_Flow_median	Special_Filling_Flow_mean	Spe
2023-12-23 06:00:00	167.37	202.16	183.960	184.25875	
2023-12-23 14:00:00	159.11	191.34	175.090	175.81250	
2023-12-23 22:00:00	183.58	230.74	213.385	208.41375	
2023-12-24 06:00:00	188.26	233.75	232.010	221.59125	
2023-12-24 14:00:00	209.53	233.20	227.015	225.90625	

5 rows × 2600 columns

```
df.tail()
```

Timestamp_To	Special_Filling_Flow_min	Special_Filling_Flow_max	Special_Filling_Flow_median	Special_Filling_Flow_mean	Spe
2024-01-03 14:00:00	164.30	232.93	183.920	195.84625	
2024-01-03 22:00:00	150.07	187.79	164.255	167.34875	
2024-01-04 06:00:00	160.16	206.59	176.500	178.42625	
2024-01-04 14:00:00	155.95	206.67	172.235	176.65750	
2024-01-04 22:00:00	180.34	215.56	194.300	196.03125	

5 rows × 2600 columns

```
df_lag_cleaned = df.dropna()
```

```
df_lag_cleaned.shape
```

```
(1, 2600)
```

```
df_lag_cleaned.head()
```

```
best_cols = '/content/precip_final_cols (1).pkl'
with open(best_cols, 'rb') as file:
    final_cols = pickle.load(file)
```

```
final_cols
```

```
        'HAT_45u_shift_41',
        'HAT_45u_shift_44',
        'HAT_45u_shift_47',
        'HAT_45u_shift_49',
        'HAT_45u_shift_50',
        'HAT_45u_shift_56',
        'HAT_45u_shift_67',
        'HAT_45u_shift_78',
        'HAT_D50_shift_40',
        'HAT_D50_shift_41',
        'HAT_D50_shift_45',
        'HAT_D50_shift_48',
        'HAT_D50_shift_51',
        'HAT_D50_shift_54',
        'HAT_D50_shift_60',
        'HAT_D50_shift_61',
        'HAT_D50_shift_68',
        'HAT_D50_shift_69',
        'HAT_D50_shift_71',
        'HAT_D50_shift_79',
        'HAT_D50_shift_81',
        'Filling_Ratio_shift_54',
        'Filling_Ratio_shift_64',
        'Filling_Ratio_shift_80',
        'Feed_Hydrate_D50']

filtered_cols = [col for col in final_cols if col != "Feed_Hydrate_D50" and int(col.split('_')[-1]) >= 45]

filtered_cols.append("Feed_Hydrate_D50")


filtered_cols
```

```
    '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_73',
    '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_45',
    'Regular_Filling_Flow_min_shift_50',
    '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_52',
    'Regular_Filling_Flow_median_shift_57',
    'Regular_Filling_Flow_std_shift_49',
    'Regular_Filling_Flow_std_shift_73',
    'Regular_Filling_Flow_std_shift_74',
    'Regular_Filling_Flow_std_shift_75',
    'Regular_Filling_Flow_std_shift_77',
    'Regular_Filling_Flow_std_shift_78',
    'Regular_Filling_Flow_std_shift_82',
    'Regular_Filling_Flow_var_coeff_shift_58',
    'Regular_Filling_Flow_var_coeff_shift_70',
    'Coarse_Seed_2_Flow_max_shift_52',
    'Coarse_Seed_2_Flow_mean_shift_54',
    'HAT_45u_shift_47',
    'HAT_45u_shift_49',
```

```
        'HAT_D50_shift_71',
        'HAT_D50_shift_79',
        'HAT_D50_shift_81',
        'Filling_Ratio_shift_54',
        'Filling_Ratio_shift_64',
        'Filling_Ratio_shift_80',
        'Feed_Hydrate_D50']

df_final_cleaned=df_lag_cleaned[filtered_cols]


df_final_cleaned.shape
```

```
(1, 70)
```

```
target_column ='Feed_Hydrate_D50'
X_unseen = df_final_cleaned.drop([target_column],axis=1).copy()
y_unseen = df_final_cleaned[[target_column]].copy()
best_model = '/content/Light_gbm_model_04-01-2024.pkl'

with open(best_model, 'rb') as file:
    model = pickle.load(file)



X_unseen
```

Timestamp_To	Fine_Seed_Tonnage_min_shift_50	Fine_Seed_Tonnage_min_shift_59	Fine_Seed_Tonnage_min_shift_61	Fine_Seed_Tonna
2024-01-04 22:00:00	48.62	33.95	49.1	

```
list(X_unseen.columns)
```

```
'Fine_Seed_SSA_shift_70',
'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_73',
'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_45',
'Regular_Filling_Flow_min_shift_50',
'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_52',
'Regular_Filling_Flow_median_shift_57',
'Regular_Filling_Flow_std_shift_49',
'Regular_Filling_Flow_std_shift_73',
'Regular_Filling_Flow_std_shift_74',
'Regular_Filling_Flow_std_shift_75',
'Regular_Filling_Flow_std_shift_77',
'Regular_Filling_Flow_std_shift_78',
```

```
        'HAT_D50_shift_45',
        'HAT_D50_shift_48',
        'HAT_D50_shift_51',
        'HAT_D50_shift_54',
        'HAT_D50_shift_60',
        'HAT_D50_shift_61',
        'HAT_D50_shift_68',
        'HAT_D50_shift_69',
        'HAT_D50_shift_71',
        'HAT_D50_shift_79',
        'HAT_D50_shift_81',
        'Filling_Ratio_shift_54',
        'Filling_Ratio_shift_64',
        'Filling_Ratio_shift_80']
```

```
X_unseen.index + timedelta(days=15)
```

```
DatetimeIndex(['2024-01-19 22:00:00'], dtype='datetime64[ns]', name='Timestamp_To', freq=None)
```

```
preds = model.predict(X_unseen.values)
```

```
print(f"we are predicting for {future_timestamp}")
```

```
we are predicting for 2024-01-19 22:00:00
```

```
future_timestamp
```

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

```
preds
```

```
array([108.7532109])
```

```
preds
```

```
array([108.7532109])
```

```
preds=preds.round(2)
```

```
df_unseen_PSD=pd.DataFrame()
df_unseen_PSD['Timestamp_To']=X_unseen.index+ timedelta(days=15)
df_unseen_PSD['PSD_Prediction']=preds.round(2)
```

```
df_unseen_PSD
```

	Timestamp_To	PSD_Prediction
0	2024-01-19 22:00:00	108.75

```
df_unseen_PSD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   Timestamp_To    1 non-null      datetime64[ns]
 1   PSD_Prediction  1 non-null      float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 148.0 bytes
```

```
def get_predict_shift(df):
    hour_now=df['Timestamp_To'].dt.hour[0]
    if hour_now < 14:  # Before 2 PM
        if hour_now < 6:  # Before 6 AM
            shift = "C"  # Previous day's night shift
        else:
            shift = "A"  # Morning shift
    elif hour_now < 22:  # Between 2 PM and 10 PM
        shift = "B"  # Afternoon shift
    else:
        shift = "C"  # Night shift
    predicted_hour=df['Timestamp_To'][0]
```

```
        return shift,predicted_hour


    predict_shift,predicted_hour=get_predict_shift(df_unseen_PSD)
```