In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import MinMaxScaler

# Load the dataset
df = pd.read_csv("C:/Users/tiwar/Desktop/honeywell/81ce1f00-c3f4-4baa-9b57-006fad1875adTEP_Train_Test.csv")

# understanding the data
print("Data Shape:", df.shape)
df.head()
```

```
Data Shape: (26400, 53)
```

Out[1]:

	Time	AFeedStream1	DFeedStream2	EFeedStream3	TotalFeedStream4	RecycleFlowStream8	ReactorFeedRateStream6	ReactorPressurekPagauge	ReactorLevel	ReactorTemperatureDegC	...	EFeedFlowStr
0	1/1/2004 0:00	0.25038	3674.0	4529.0	9.2320	26.889	42.402	2704.3	74.863	120.41	...	5
1	1/1/2004 0:01	0.25109	3659.4	4556.6	9.4264	26.721	42.576	2705.0	75.000	120.41	...	5
2	1/1/2004 0:02	0.25038	3660.3	4477.8	9.4426	26.875	42.070	2706.2	74.771	120.42	...	5
3	1/1/2004 0:03	0.24977	3661.3	4512.1	9.4776	26.758	42.063	2707.2	75.224	120.39	...	5
4	1/1/2004 0:04	0.29405	3679.0	4497.0	9.3381	26.889	42.650	2705.1	75.388	120.39	...	5

5 rows × 53 columns

In [2]:
```
# preparing our training data from that 'normal' period.
CORRECT_DATETIME_COLUMN = 'Time'

df[CORRECT_DATETIME_COLUMN] = pd.to_datetime(df[CORRECT_DATETIME_COLUMN])

train_start_time = pd.to_datetime('2004-01-01 00:00:00')
train_end_time = pd.to_datetime('2004-01-05 23:59:59')

train_df = df[(df[CORRECT_DATETIME_COLUMN] >= train_start_time) & (df[CORRECT_DATETIME_COLUMN] <= train_end_time)]

feature_columns = [col for col in df.columns if col not in [CORRECT_DATETIME_COLUMN, 'ATT_FLAG']]

X_train = train_df[feature_columns]

print("Training data shape:", X_train.shape)
```

```
Training data shape: (7200, 52)
```

In [3]:
```
# Teach the model what 'normal' data looks like.
model = IsolationForest(contamination='auto', random_state=42)
print("Training the model...")
model.fit(X_train)
print("Model training complete.")
```

```
Training the model...
Model training complete.
```

In [4]:
```
# Find and score anomalies across all the data.
X_full = df[feature_columns]

raw_scores = model.decision_function(X_full)

from sklearn.preprocessing import MinMaxScaler # using scaler on the range of (0,100)
scaler = MinMaxScaler(feature_range=(0, 100))

df['Anomaly_Score'] = scaler.fit_transform(-raw_scores.reshape(-1, 1))

df.sort_values(by='Anomaly_Score', ascending=False).head(10)
```

Out[4]:

	Time	AFeedStream1	DFeedStream2	EFeedStream3	TotalFeedStream4	RecycleFlowStream8	ReactorFeedRateStream6	ReactorPressurekPagauge	ReactorLevel	ReactorTemperatureDegC	...	AFeedFlo
23932	2004-01-17 14:52:00	0.57684	3471.0	4323.3	7.1227	26.784	40.225	2502.8	80.571	120.37	...	
23934	2004-01-17 14:54:00	0.69442	3505.5	4172.0	7.1247	26.916	41.072	2454.1	80.581	120.27	...	
23939	2004-01-17 14:59:00	0.79705	3455.1	3898.9	8.4276	26.835	41.743	2483.6	72.163	120.54	...	
23933	2004-01-17 14:53:00	0.57845	3489.4	4274.9	7.0259	27.114	40.090	2492.5	80.750	120.35	...	
23945	2004-01-17 15:05:00	0.77223	3500.7	3730.4	9.4480	26.911	42.651	2521.3	67.776	120.50	...	
23861	2004-01-17 13:41:00	0.70855	3435.7	4110.6	8.4399	26.650	41.675	2545.9	73.768	120.44	...	
23862	2004-01-17 13:42:00	0.71800	3502.4	4003.8	8.7090	26.790	41.875	2547.8	73.544	120.54	...	
23935	2004-01-17 14:55:00	0.69136	3470.8	4144.0	7.4896	26.894	41.049	2428.6	77.848	120.32	...	
23951	2004-01-17 15:11:00	0.55370	3454.7	3771.0	10.3470	26.732	43.345	2586.7	65.194	120.55	...	
23940	2004-01-17 15:00:00	0.83364	3399.7	3795.2	8.5568	26.781	41.770	2489.8	71.547	120.55	...	

10 rows × 54 columns

In [5]:
```
# Pinpoint which features caused each anomaly.
normal_stats = X_train.describe().transpose()

def get_contributing_features(row):

    if row['Anomaly_Score'] < 90:
        return [np.nan] * 7

    z_scores = {}
    for col in feature_columns:
        z = (row[col] - normal_stats.loc[col, 'mean']) / normal_stats.loc[col, 'std']
        z_scores[col] = abs(z)

    sorted_features = sorted(z_scores.items(), key=lambda item: item[1], reverse=True)

    top_features = [feature[0] for feature in sorted_features[:7]]

    while len(top_features) < 7:
        top_features.append('')

    return top_features

print("Identifying contributing features for severe anomalies...")
contributions = df.apply(get_contributing_features, axis=1, result_type='expand')
contributions.columns = [f'top_feature_{i+1}' for i in range(7)]
print("Done.")

final_df = pd.concat([df, contributions], axis=1)
final_df.sort_values(by='Anomaly_Score', ascending=False).head()
```

```
Identifying contributing features for severe anomalies...
Done.
```

Out[5]:

	Time	AFeedStream1	DFeedStream2	EFeedStream3	TotalFeedStream4	RecycleFlowStream8	ReactorFeedRateStream6	ReactorPressurekPagauge	ReactorLevel	ReactorTemperatureDegC	...	ReactorCo
23932	2004-01-17 14:52:00	0.57684	3471.0	4323.3	7.1227	26.784	40.225	2502.8	80.571	120.37	...	
23934	2004-01-17 14:54:00	0.69442	3505.5	4172.0	7.1247	26.916	41.072	2454.1	80.581	120.27	...	
23939	2004-01-17 14:59:00	0.79705	3455.1	3898.9	8.4276	26.835	41.743	2483.6	72.163	120.54	...	
23933	2004-01-17 14:53:00	0.57845	3489.4	4274.9	7.0259	27.114	40.090	2492.5	80.750	120.35	...	
23945	2004-01-17 15:05:00	0.77223	3500.7	3730.4	9.4480	26.911	42.651	2521.3	67.776	120.50	...	

5 rows × 61 columns

In [6]:
```
# Reordering columns to make the output clean
output_columns = df.columns.tolist() + contributions.columns.tolist()
final_df = final_df[output_columns]

# Saving the final dataframe to a CSV file
output_filename = 'anomaly_detection_results.csv'
final_df.to_csv(output_filename, index=False)

print(f"Successfully created the output file: {output_filename}")
```

```
Successfully created the output file: anomaly_detection_results.csv
```

In [ ]: