ME5001 Project

Group 5

Team Members:
Vishnu M - ME17B184
Vishnu Sajan - ME17B172
Safwan P C - ME17B163

In [94]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
```

Data Preperation

In [95]:
```
df1=pd.read_csv(r'C:\Users\REJILA\Downloads\Training\Training\Small\Sample01.csv') # reading one dataset to understand the dataset
```

In [96]:
```
df1.head()
```

Out[96]:

	Time(s)	Flow_Rate(ml/m)	Upstream_Pressure(psi)	Downstream_Pressure(psi)
0	0.0	9.935067	-0.30000	-1.10625
1	0.1	4.110983	0.61250	-0.34375
2	0.2	5.098116	1.05625	-0.08125
3	0.3	10.231207	1.20000	0.84375
4	0.4	8.355655	0.93125	0.79375

In [97]:
```
particlesize_file=pd.read_excel(r'C:\Users\REJILA\Downloads\Training\Training\Training Operation Profiles of Samples.xlsx') #reading excel file having the particle size and solid ratio of training dataset
```

In [98]:
```
particlesize_file.head()
```

Out[98]:

	Sample	Particle Size (micron)	Solid Ratio(%)
0	1	45-53	0.400
1	2	45-53	0.400
2	3	45-53	0.400
3	4	45-53	0.400
4	5	45-53	0.425

In [99]:
```
particlesize_file.columns
```

Out[99]:
```
Index(['Sample', 'Particle Size (micron)', 'Solid Ratio(%)'], dtype='object')
```

In [100]:
```
training_samples={} #dictionary to store the training datasets
```

Reading all the training datasets, adding the columns of particle size and solid ratio and storing them in a dictionary

In [101]:
```
for i in range(1,10): #reading dataset with particle size small
    # particle size is taken as the average of lower and upper limit (eg: 45-53 = (45+53)/2 = 49)
    df=pd.read_csv(r'C:\Users\REJILA\Downloads\Training\Training\Small\Sample0'+str(i)+'.csv')
    df['Particle_size']=(int(particlesize_file['Particle Size (micron)'][i-1][0:2])+int(particlesize_file['Particle Size (micron)'][i-1][3:5]))/2
    df['Solid Ratio(%)']=particlesize_file['Solid Ratio(%)'][i-1]
    training_samples[i]=df
for i in range(10,13):
    df=pd.read_csv(r'C:\Users\REJILA\Downloads\Training\Training\Small\Sample'+str(i)+'.csv')
    df['Particle_size']=(int(particlesize_file['Particle Size (micron)'][i-1][0:2])+int(particlesize_file['Particle Size (micron)'][i-1][3:5]))/2
    df['Solid Ratio(%)']=particlesize_file['Solid Ratio(%)'][i-1]
    training_samples[i]=df

for i in range(33,45):#reading dataset with particle size large
    df=pd.read_csv(r'C:\Users\REJILA\Downloads\Training\Training\Large\Sample'+str(i)+'.csv')
    df['Particle_size']=(int(particlesize_file['Particle Size (micron)'][i-1-20][0:2])+int(particlesize_file['Particle Size (micron)'][i-1-20][3:5]))/2
    df['Solid Ratio(%)']=particlesize_file['Solid Ratio(%)'][i-1-20]
    training_samples[i-20]=df
```

In [102]:
```
validation_samples={} #dictionary to store the validation datasets
```

In [103]:
```
particlesizevalid_file=pd.read_excel(r'C:\Users\REJILA\Downloads\Validation (1)\Validation\Validation Operation Profiles of Samples.xlsx')
```

Reading all the validation datasets, adding the columns of particle size and solid ratio and storing them in a dictionary

In [104]:
```
for i in range(13,17):#reading dataset with particle size small
    df=pd.read_csv(r'C:\Users\REJILA\Downloads\Validation (1)\Validation\Small\Sample'+str(i)+'.csv')
    df['Particle_size']=(int(particlesizevalid_file['Particle Size (micron)'][i-13][0:2])+int(particlesizevalid_file['Particle Size (micron)'][i-13][3:5]))/2
    df['Solid Ratio(%)']=particlesizevalid_file['Solid Ratio(%)'][i-13]
    validation_samples[i-12]=df

for i in range(45,49):#reading dataset with particle size large
    df=pd.read_csv(r'C:\Users\REJILA\Downloads\Validation (1)\Validation\Large\Sample'+str(i)+'.csv')
    df['Particle_size']=(int(particlesizevalid_file['Particle Size (micron)'][i-41][0:2])+int(particlesizevalid_file['Particle Size (micron)'][i-41][3:5]))/2
    df['Solid Ratio(%)']=particlesizevalid_file['Solid Ratio(%)'][i-41]
    validation_samples[i-40]=df
```

In [ ]:

In [105]:
```
training_samples[5].head() #sample of the changed dataset
```

Out[105]:

	Time(s)	Flow_Rate(ml/m)	Upstream_Pressure(psi)	Downstream_Pressure(psi)	Particle_size	Solid Ratio(%)
0	0.0	4.604549	0.42500	0.43750	49.0	0.425
1	0.1	6.973668	0.95625	1.05625	49.0	0.425
2	0.2	5.098116	1.15000	0.73125	49.0	0.425
3	0.3	3.518703	1.25000	0.38125	49.0	0.425
4	0.4	1.445724	0.98750	-0.02500	49.0	0.425

Finding remaining useful life(RUL) of training datasets from the conditions given

In [106]:
```
for i in range(1,25): #finding RUL
    x=np.array(training_samples[i][abs(training_samples[i]["Upstream_Pressure(psi)"]-training_samples[i]["Downstream_Pressure(psi)"])>20]['Time(s)'])[0]
    training_samples[i]['Remaining_Useful_Time'] = x-training_samples[i]["Time(s)"]
for i in range(1,25): # Checking the time instant where there is a sudden jump in the flowrate and removing the rows above that instant
    c=0
    for j in range(1,1000):
        if training_samples[i]['Flow_Rate(ml/m)'][j]-training_samples[i]['Flow_Rate(ml/m)'][j-1]>100:
            c=j
            break
    training_samples[i]=training_samples[i][j:]
    training_samples[i].reset_index(inplace=True)
    training_samples[i]=training_samples[i].drop(['index'],axis=1)
```

Finding remaining useful life(RUL) of validation datasets from the conditions given

In [107]:
```
for i in range(1,9):#finding RUL
    x=np.array(validation_samples[i][abs(validation_samples[i]["Upstream_Pressure(psi)"]-validation_samples[i]["Downstream_Pressure(psi)"])>20]['Time(s)'])[0]
    validation_samples[i]['Remaining_Useful_Time'] = x-validation_samples[i]["Time(s)"]
for i in range(1,9):# Checking the time instant where there is a sudden jump in the flowrate and removing the rows above that instant
    c=0
    for j in range(1,1000):
        if validation_samples[i]['Flow_Rate(ml/m)'][j]-validation_samples[i]['Flow_Rate(ml/m)'][j-1]>100:
            c=j
            break
    validation_samples[i]=validation_samples[i][j:]
    validation_samples[i].reset_index(inplace=True)
    validation_samples[i]=validation_samples[i].drop(['index'],axis=1)
```

In [108]:
```
validation_samples[1].head()
```

Out[108]:

	Time(s)	Flow_Rate(ml/m)	Upstream_Pressure(psi)	Downstream_Pressure(psi)	Particle_size	Solid Ratio(%)	Remaining_Useful_Time
0	4.1	505.673256	1.15625	0.90000	49.0	0.475	208.1
1	4.2	399.655173	1.48125	0.80625	49.0	0.475	208.0
2	4.3	438.745639	1.49375	0.80625	49.0	0.475	207.9
3	4.4	490.767548	1.12500	-0.13125	49.0	0.475	207.8
4	4.5	523.639077	0.63125	-0.40000	49.0	0.475	207.7

In [109]:
```
from sklearn.preprocessing import StandardScaler
```

Combining all the training datasets into a single dataset- final_train

In [110]:
```
final_train=pd.concat(training_samples.values())
```

Combining all the validation datasets into a single dataset- final_valid

In [111]:
```
final_valid=pd.concat(validation_samples.values())
```

In [112]:
```
final_train['pressure_difference']=final_train['Upstream_Pressure(psi)']-final_train['Downstream_Pressure(psi)']
final_valid['pressure_difference']=final_valid['Upstream_Pressure(psi)']-final_valid['Downstream_Pressure(psi)']
```

In [113]:
```
final_train.head()
```

Out[113]:

	Time(s)	Flow_Rate(ml/m)	Upstream_Pressure(psi)	Downstream_Pressure(psi)	Particle_size	Solid Ratio(%)	Remaining_Useful_Time	pressure_
0	7.7	172.515871	-0.13750	0.20625	49.0	0.4	265.7	
1	7.8	578.030105	-0.93125	-0.89375	49.0	0.4	265.6	
2	7.9	597.575338	-0.14375	-1.20625	49.0	0.4	265.5	
3	8.0	631.927566	-0.20000	-1.13750	49.0	0.4	265.4	
4	8.1	653.447065	0.04375	-0.89375	49.0	0.4	265.3	

In [114]:
```
#final_train=final_train.drop(['Upstream_Pressure(psi)','Downstream_Pressure(psi)'],axis=1)
#final_valid=final_valid.drop(['Upstream_Pressure(psi)','Downstream_Pressure(psi)'],axis=1)
```

In [115]:
```
X_valid=final_valid.drop(['Remaining_Useful_Time','Time(s)'],axis=1)
y_valid=final_valid['Remaining_Useful_Time']
```

Test train split

In [116]:
```
from sklearn.model_selection import train_test_split
X=final_train.drop(['Remaining_Useful_Time','Time(s)'],axis=1)
Y=final_train['Remaining_Useful_Time']
```

In [117]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.33, random_state=42)
```

Data scaling

In [118]:
```
scaler = StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_valid_scaled=scaler.transform(X_valid)
```

In [119]:
```
X_test_scaled=scaler.transform(X_test)
```

In [120]:
```
scaler2=StandardScaler()
y_train_scaled=scaler2.fit_transform(y_train.values.reshape(-1,1))
y_valid_scaled=scaler2.transform(y_valid.values.reshape(-1,1))
```

In [121]:
```
y_test_scaled=scaler2.transform(y_test.values.reshape(-1,1))
```

Model Training

Gradient Boosting Regression

In [60]:
```
from sklearn.metrics import mean_squared_error
from sklearn import datasets, ensemble
params = {'n_estimators': 500,
          'max_depth': 10,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train_scaled, y_train_scaled)

mse = mean_squared_error(y_test_scaled, reg.predict(X_test_scaled))
print("The mean squared error (MSE) on validation set: {:.4f}".format(mse))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, reg.predict(X_valid_scaled))))
```

```
C:\Users\REJILA\Anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:1454: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
The mean squared error (MSE) on validation set: 0.0665
The mean squared error (MSE) on test set: 0.0614
```

In [61]:
```
from sklearn.metrics import r2_score
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, reg.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, reg.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.9335
The r2_score on test set: 0.9107
```

Random Forest Regression

In [62]:
```
from sklearn.ensemble import RandomForestRegressor
regr = RandomForestRegressor(n_estimators=500,max_depth=10, random_state=0)
regr.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, regr.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, regr.predict(X_valid_scaled))))
```

```
C:\Users\REJILA\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
The mean squared error (MSE) on validation set: 0.0790
The mean squared error (MSE) on test set: 0.0717
```

In [63]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, regr.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, regr.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.9211
The r2_score on test set: 0.8958
```

Linear Regression

In [64]:
```
from sklearn.linear_model import LinearRegression
f=LinearRegression()
f.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, f.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, f.predict(X_valid_scaled))))
```

```
The mean squared error (MSE) on validation set: 0.4139
The mean squared error (MSE) on test set: 0.2917
```

In [65]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, f.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, f.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.5864
The r2_score on test set: 0.5759
```

XGBoost Regressor

In [66]:
```
import xgboost as xgb
best_xgb_model = xgb.XGBRegressor(colsample_bytree=0.5,
                 gamma=0,
                 learning_rate=0.05,
                 max_depth=5,
                 min_child_weight=1.6,
                 n_estimators=892,
                 reg_alpha=0.75,
                 reg_lambda=0.45,
                 subsample=0.6,
                 seed=42)
best_xgb_model.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, best_xgb_model.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, best_xgb_model.predict(X_valid_scaled))))
```

```
The mean squared error (MSE) on validation set: 0.0707
The mean squared error (MSE) on test set: 0.0616
```

In [67]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, best_xgb_model.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, best_xgb_model.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.9293
The r2_score on test set: 0.9104
```

Catboost Regression

In [92]:
```
from catboost import CatBoostRegressor
cb_model = CatBoostRegressor(iterations=800,
                             learning_rate=0.045,
                             depth=12,
                             eval_metric='RMSE',
                             random_seed = 23,
                             bagging_temperature = 0.2,
                             od_type='Iter',
                             metric_period = 75,
                             od_wait=100)
cb_model.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, cb_model.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, cb_model.predict(X_valid_scaled))))
```

```
0:	learn: 0.9617200	total: 131ms	remaining: 1m 44s
75:	learn: 0.2779253	total: 11.2s	remaining: 1m 47s
150:	learn: 0.2556421	total: 21.5s	remaining: 1m 32s
225:	learn: 0.2468859	total: 31.3s	remaining: 1m 19s
300:	learn: 0.2402787	total: 43.4s	remaining: 1m 12s
375:	learn: 0.2347447	total: 53.8s	remaining: 1m
450:	learn: 0.2299078	total: 1m 3s	remaining: 49.4s
525:	learn: 0.2252885	total: 1m 14s	remaining: 38.6s
600:	learn: 0.2210827	total: 1m 24s	remaining: 28s
675:	learn: 0.2171143	total: 1m 36s	remaining: 17.7s
750:	learn: 0.2137618	total: 1m 46s	remaining: 6.97s
799:	learn: 0.2114347	total: 1m 53s	remaining: 0us
The mean squared error (MSE) on validation set: 0.0649
The mean squared error (MSE) on test set: 0.0595
```

In [93]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, cb_model.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, cb_model.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.9352
The r2_score on test set: 0.9135
```

SGD Regression

In [70]:
```
from sklearn.linear_model import SGDRegressor
clf_ = SGDRegressor()
clf_.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, clf_.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, clf_.predict(X_valid_scaled))))
```

```
The mean squared error (MSE) on validation set: 0.4140
The mean squared error (MSE) on test set: 0.2901
```

```
C:\Users\REJILA\Anaconda3\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [71]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, clf_.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, clf_.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.5864
The r2_score on test set: 0.5783
```

Decision Tree Regression

In [72]:
```
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train_scaled, y_train_scaled)
print("The mean squared error (MSE) on validation set: {:.4f}".format(mean_squared_error(y_test_scaled, decision_tree.predict(X_test_scaled))))
print("The mean squared error (MSE) on test set: {:.4f}".format(mean_squared_error(y_valid_scaled, decision_tree.predict(X_valid_scaled))))
```

```
The mean squared error (MSE) on validation set: 0.1329
The mean squared error (MSE) on test set: 0.1167
```

In [73]:
```
print("The r2_score on validation set: {:.4f}".format(r2_score(y_test_scaled, decision_tree.predict(X_test_scaled))))
print("The r2_score on test set: {:.4f}".format(r2_score(y_valid_scaled, decision_tree.predict(X_valid_scaled))))
```

```
The r2_score on validation set: 0.8672
The r2_score on test set: 0.8303
```

XG boost with hyperparameter tuning

In [29]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
params={
 'max_depth': [5,10,15,20],
 'learning_rate':[0.01,0.05,0.1,0.005,0.02],
 'min_samples_leaf': [1, 2, 4],
 'n_estimators': [200, 600,800, 1000, 1400, 1600, 2000]}
xg_hyper = xgb.XGBRegressor()
xg_ran= RandomizedSearchCV(estimator=xg_hyper, param_distributions = params, n_iter = 5, cv = 3, verbose=2, random_state=42, n_jobs = -1)
xg_ran.fit(X_train_scaled, y_train_scaled)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 15.5min finished
```

Out[29]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_...
                                          tree_method=None,
                                          validate_parameters=False,
                                          verbosity=None),
                   iid='deprecated', n_iter=5, n_jobs=-1,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1,
                                                          0.005, 0.02],
                                        'max_depth': [5, 10, 15, 20],
                                        'min_samples_leaf': [1, 2, 4],
                                        'n_estimators': [200, 600, 800, 1000,
                                                         1400, 1600, 2000]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [31]:
```
from sklearn.metrics import r2_score
r2_score(y_test_scaled, xg_ran.predict(X_test_scaled))
```

Out[31]:
```
0.9289812150014077
```

In [ ]: