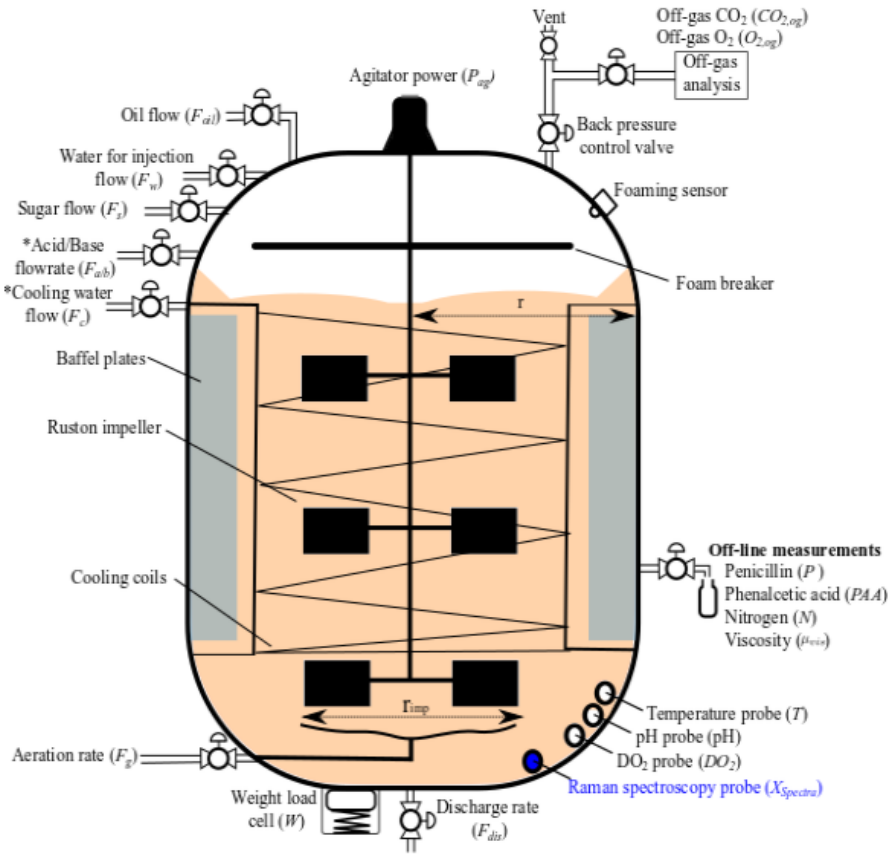
Biopharmaceutical manufacturing of penicillin : An Analysis

Abstract

Industrial-Scale Penicillin Simulation: IndPenSim

- The simulation was developed using a mechanistic model and validated using historical data collected from an industrial-scale penicillin fermentation process (the batch data is available for download). Each batch was carried out in a 100,000 litre bioreactor using an industrial strain of Penicillium chrysogenum, a schematic of the bioreactor is given below.



Index

INTRODUCTION

What is penicillin?

Penicillins are a group of antibacterial drugs that attack a wide range of bacteria. They were the first drugs of this type that doctors used. The discovery and manufacture of penicillins have changed the face of medicine, as these drugs have saved millions of lives.

Penicillium fungi are the source of penicillin, which people can take orally or via injection.

People across the globe now widely use penicillins to treat various infections and diseases.

About our data set?

This data was generated using a 100,000 litre penicillin fermentation system referenced as IndPenSim.

IndPenSim is the first to include a realistic simulated Raman spectroscopy device for the purpose of developing, evaluating and implementation of advanced and innovative control solutions applicable to biotechnology facilities. This data set generated by IndPenSim represents the biggest data set available for advanced data analytics and contains 100 batches with all available process and Raman spectroscopy measurements (~2.5 GB). This data is highly suitable for the development of big data analytics, machine learning (ML) or artificial intelligence (AI) algorithms applicable to the biopharmaceutical industry. The 100 batches are controlled using different control strategies and different batch lengths representing a typical Biopharmaceutical manufacturing facility:

Batches 1-30: Controlled by recipe driven approach

Batches 31-60: Controlled by operators

Batches 61-90: Controlled by an Advanced Process Control (APC) solution using the Raman spectroscopy

Batches 91-100: Contain faults resulting in process deviations.

CODE

PRE - PROCESSING

```
#installing necessary libraries
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import os
from urllib.request import urlretrieve
import zipfile
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import requests
import pandas as pd

#connecting drive from google collab here

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

link =
'https://drive.google.com/file/d/1yKgjU-dJiQ4YZ8K-j-MOebLOjCZu_U6U/view?us
p=sharing'
id = '1yKgjU-dJiQ4YZ8K-j-MOebLOjCZu_U6U'
```

```
#slicing and concatenation

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('100_Batches_IndPenSim_V3.csv')
df=pd.read_csv('100_Batches_IndPenSim_V3.csv')
```

Checking the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113935 entries, 0 to 113934
Columns: 2239 entries, Time (h) to 201
dtypes: float64(2225), int64(14)
memory usage: 1.9 GB
```

```
#To check missing values
df.isna().sum()
```

```
Time (h)                          0
Aeration rate(Fg:L/h)             0
Agitator RPM(RPM:RPM)             0
Sugar feed rate(Fs:L/h)           0
Acid flow rate(Fa:L/h)            0
                                 ...
205                               0
204                               0
203                               0
202                          113935
201                          113935
Length: 2239, dtype: int64
```

```
#Selecting the desired columns
newdf = df.loc[:,['Time (h)', 'Aeration rate(Fg:L/h)', 'Agitator RPM(RPM:RPM)', 'Sugar feed rate(Fs:L/h)', 'Acid flow rate(Fa:L/h)', 'Base flow rate(Fb:L/h)', 'Heating/cooling water flow rate(Fc:L/h)', 'Heating water flow rate(Fh:L/h)', 'Water for injection/dilution(Fw:L/h)', 'Air head pressure(pressure:bar)', 'Dumped broth flow(Fremoved:L/h)', 'Substrate concentration(S:g/L)', 'Dissolved oxygen
```

```
concentration(DO2:mg/L)', 'Penicillin concentration(P:g/L)', 'Vessel
Volume(V:L)', 'Vessel Weight(Wt:Kg)', 'pH(pH:pH)', 'Temperature(T:K)',
'Generated heat(Q:kJ)', 'carbon dioxide percent in off-gas(CO2outgas:%)',
'PAA flow(Fpaa:PAA flow (L/h))', 'PAA concentration
offline(PAA_offline:PAA (g L^{-1}))', 'Oil flow(Foil:L/hr)', 'NH_3
concentration off-line(NH3_offline:NH3 (g L^{-1}))', 'Oxygen Uptake
Rate(OUR:(g min^{-1}))', 'Oxygen in percent in off-gas(O2:O2  (%))',
'Offline Penicillin concentration(P_offline:P(g L^{-1}))', 'Offline
Biomass concentratio(X_offline:X(g L^{-1}))', 'Carbon evolution
rate(CER:g/h)', 'Ammonia shots(NH3_shots:kgs)',
'Viscosity(Viscosity_offline:centPoise)', 'Fault reference(Fault_ref:Fault
ref)', '0 - Recipe driven 1 - Operator controlled(Control_ref:Control
ref)', '1- No Raman spec', '2-PAT control(PAT_ref:PAT ref)', 'Batch
reference(Batch_ref:Batch ref)', 'Batch ID', 'Fault flag', ' 1-Raman spec
recorded']]
```

Missing Values in the desired columns

```
#Check for missing values
newdf.isna().sum()
```

```
Time (h)                                                                0
Aeration rate(Fg:L/h)                                                   0
Agitator RPM(RPM:RPM)                                                   0
Sugar feed rate(Fs:L/h)                                                 0
Acid flow rate(Fa:L/h)                                                  0
Base flow rate(Fb:L/h)                                                  0
Heating/cooling water flow rate(Fc:L/h)                                 0
Heating water flow rate(Fh:L/h)                                         0
Water for injection/dilution(Fw:L/h)                                    0
Air head pressure(pressure:bar)                                         0
Dumped broth flow(Fremoved:L/h)                                         0
Substrate concentration(S:g/L)                                          0
Dissolved oxygen concentration(DO2:mg/L)                                0
Penicillin concentration(P:g/L)                                         0
Vessel Volume(V:L)                                                      0
Vessel Weight(Wt:Kg)                                                    0
pH(pH:pH)                                                               0
Temperature(T:K)                                                        0
Generated heat(Q:kJ)                                                    0
carbon dioxide percent in off-gas(CO2outgas:%)                          0
PAA flow(Fpaa:PAA flow (L/h))                                           0
PAA concentration offline(PAA_offline:PAA (g L^{-1}))              111873
Oil flow(Foil:L/hr)                                                     0
NH_3 concentration off-line(NH3_offline:NH3 (g L^{-1}))            111873
Oxygen Uptake Rate(OUR:(g min^{-1}))                                    0
Oxygen in percent in off-gas(O2:O2  (%))                                0
Offline Penicillin concentration(P_offline:P(g L^{-1}))            111873
Offline Biomass concentratio(X_offline:X(g L^{-1}))                111873
Carbon evolution rate(CER:g/h)                                          0
Ammonia shots(NH3_shots:kgs)                                            0
Viscosity(Viscosity_offline:centPoise)                             111873
Fault reference(Fault_ref:Fault ref)                                    0
0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)      0
1- No Raman spec                                                        0
2-PAT control(PAT_ref:PAT ref)                                          0
Batch reference(Batch_ref:Batch ref)                                    0
Batch ID                                                                0
```

These values are the quantity of a particular element used in the process of penicillin generation. We can not replace these values with mean or mode as this would mean change in the ratio of ingredients used in the process. Hence the missing values are assumed to be 0.

```
#Replacing the missing values with 0
newdf[['PAA concentration offline(PAA_offline:PAA (g L^{-1}))', 'NH_3
concentration off-line(NH3_offline:NH3 (g L^{-1}))', 'Offline Penicillin
concentration(P_offline:P(g L^{-1}))', 'Offline Biomass
```

```
concentratio(X_offline:X(g L^{-1}))',
'Viscosity(Viscosity_offline:centPoise)']] = newdf[['PAA concentration
offline(PAA_offline:PAA (g L^{-1}))', 'NH_3 concentration
off-line(NH3_offline:NH3 (g L^{-1}))', 'Offline Penicillin
concentration(P_offline:P(g L^{-1}))', 'Offline Biomass
concentratio(X_offline:X(g L^{-1}))',
'Viscosity(Viscosity_offline:centPoise)']].fillna(0)
```

```
#checking missing values again
newdf.isna().sum()
```

```
Time (h)                                                          0
Aeration rate(Fg:L/h)                                             0
Agitator RPM(RPM:RPM)                                             0
Sugar feed rate(Fs:L/h)                                           0
Acid flow rate(Fa:L/h)                                            0
Base flow rate(Fb:L/h)                                            0
Heating/cooling water flow rate(Fc:L/h)                           0
Heating water flow rate(Fh:L/h)                                   0
Water for injection/dilution(Fw:L/h)                              0
Air head pressure(pressure:bar)                                   0
Dumped broth flow(Fremoved:L/h)                                   0
Substrate concentration(S:g/L)                                    0
Dissolved oxygen concentration(DO2:mg/L)                          0
Penicillin concentration(P:g/L)                                   0
Vessel Volume(V:L)                                                0
Vessel Weight(Wt:Kg)                                              0
pH(pH:pH)                                                         0
Temperature(T:K)                                                  0
Generated heat(Q:kJ)                                              0
carbon dioxide percent in off-gas(CO2outgas:%)                    0
PAA flow(Fpaa:PAA flow (L/h))                                     0
PAA concentration offline(PAA_offline:PAA (g L^{-1}))             0
Oil flow(Foil:L/hr)                                               0
NH_3 concentration off-line(NH3_offline:NH3 (g L^{-1}))           0
Oxygen Uptake Rate(OUR:(g min^{-1}))                              0
Oxygen in percent in off-gas(O2:O2  (%))                          0
Offline Penicillin concentration(P_offline:P(g L^{-1}))           0
Offline Biomass concentratio(X_offline:X(g L^{-1}))               0
Carbon evolution rate(CER:g/h)                                    0
Ammonia shots(NH3_shots:kgs)                                      0
Viscosity(Viscosity_offline:centPoise)                            0
Fault reference(Fault_ref:Fault ref)                              0
0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)0
1- No Raman spec                                                  0
2-PAT control(PAT_ref:PAT ref)                                    0
Batch reference(Batch_ref:Batch ref)                              0
Batch ID                                                          0
```

```
newdf.describe()
```

	Time (h)	Aeration rate(Fg:L/h)	Agitator RPM(RPM:RPM)	Sugar feed rate(Fs:L/h)	Acid flow rate(Fa:L/h)	Base flow rate(Fb:L/h)	Heating/cooling water flow rate(Fc:L/h)	Heating water flow rate(Fh:L/h)	Water for injection/dilution(Fw:L/h)	pressure(
count	113935.000000	113935.000000	113935.0	113935.000000	113935.000000	113935.000000	113935.000000	113935.000000	113935.000000	
mean	114.750656	65.246360	100.0	76.663764	0.073209	61.334389	74.346341	20.763025	154.811954	
std	66.990504	11.690215	0.0	25.680134	0.552788	44.972713	108.022600	50.230266	155.601474	
min	0.200000	20.000000	100.0	2.000000	0.000000	0.000000	0.000100	0.000100	0.000000	
25%	57.000000	60.000000	100.0	72.000000	0.000000	35.766000	11.157000	0.000100	0.000000	
50%	114.000000	65.000000	100.0	80.000000	0.000000	55.407000	34.384000	0.159010	100.000000	
75%	171.000000	75.000000	100.0	90.000000	0.000000	76.271500	94.904500	11.640500	250.000000	
max	290.000000	75.000000	100.0	150.000000	12.996000	225.000000	1500.000000	1500.000000	500.000000	

Creating new dataframe for the second dataset

```
#creating new dataframe for the second dataset
df2= pd.read_csv('/content/100_Batches_IndPenSim_Statistics.csv')
df2.head()
```

	Batch ref	Penicllin_harvested_during_batch(kg)	Penicllin_harvested_end_of_batch (kg)	Penicllin_yield_total (kg)	Fault ref(0-NoFault 1-Fault)
0	1	1066400.0	1720000.0	2786400.0	0
1	2	985910.0	1340100.0	2326000.0	0
2	3	1416100.0	1259200.0	2675300.0	0
3	4	815700.0	1071000.0	1886700.0	0
4	5	1128500.0	2434400.0	3562900.0	0

```
#creating new column in newdf with NaN values
newdf["Penicllin_yield_total (kg)"] = np.nan
```

```
newdf.columns
```

```
Index(['Time (h)', 'Aeration rate(Fg:L/h)', 'Agitator RPM(RPM:RPM)',
       'Sugar feed rate(Fs:L/h)', 'Acid flow rate(Fa:L/h)',
       'Base flow rate(Fb:L/h)', 'Heating/cooling water flow rate(Fc:L/h)',
       'Heating water flow rate(Fh:L/h)',
       'Water for injection/dilution(Fw:L/h)',
       'Air head pressure(pressure:bar)', 'Dumped broth flow(Fremoved:L/h)',
       'Substrate concentration(S:g/L)',
       'Dissolved oxygen concentration(DO2:mg/L)',
       'Penicillin concentration(P:g/L)', 'Vessel Volume(V:L)',
       'Vessel Weight(Wt:Kg)', 'pH(pH:pH)', 'Temperature(T:K)',
       'Generated heat(Q:kJ)',
       'carbon dioxide percent in off-gas(CO2outgas:%)',
       'PAA flow(Fpaa:PAA flow (L/h))',
       'PAA concentration offline(PAA_offline:PAA (g L^{-1}))',
       'Oil flow(Foil:L/hr)',
       'NH_3 concentration off-line(NH3_offline:NH3 (g L^{-1}))',
       'Oxygen Uptake Rate(OUR:(g min^{-1}))',
       'Oxygen in percent in off-gas(O2:O2  (%))',
       'Offline Penicillin concentration(P_offline:P(g L^{-1}))',
       'Offline Biomass concentratio(X_offline:X(g L^{-1}))',
       'Carbon evolution rate(CER:g/h)', 'Ammonia shots(NH3_shots:kgs)',
       'Viscosity(Viscosity_offline:centPoise)',
       'Fault reference(Fault_ref:Fault ref)',
       '0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)',
       '1- No Raman spec', '2-PAT control(PAT_ref:PAT ref)',
       'Batch reference(Batch_ref:Batch ref)', 'Batch ID', 'Fault flag',
       ' 1-Raman spec recorded', 'Penicllin_yield_total (kg)'],
      dtype='object')
```

```
#Mapping the values from two datasets
for i in range(0,len(newdf)):
  x=newdf.loc[i]
  y=int(x['2-PAT control(PAT_ref:PAT ref)'])
  z=df2.loc[y-1]
  newdf.loc[i,'Penicllin_yield_total (kg)']=z['Penicllin_yield_total (kg)']
```

```
newdf.isna().sum()
```

```
Time (h)                                                                0
Aeration rate(Fg:L/h)                                                   0
Agitator RPM(RPM:RPM)                                                   0
Sugar feed rate(Fs:L/h)                                                 0
Acid flow rate(Fa:L/h)                                                  0
Base flow rate(Fb:L/h)                                                  0
Heating/cooling water flow rate(Fc:L/h)                                 0
Heating water flow rate(Fh:L/h)                                         0
Water for injection/dilution(Fw:L/h)                                    0
Air head pressure(pressure:bar)                                         0
Dumped broth flow(Fremoved:L/h)                                         0
Substrate concentration(S:g/L)                                          0
Dissolved oxygen concentration(DO2:mg/L)                                0
Penicillin concentration(P:g/L)                                         0
Vessel Volume(V:L)                                                      0
Vessel Weight(Wt:Kg)                                                    0
pH(pH:pH)                                                               0
Temperature(T:K)                                                        0
Generated heat(Q:kJ)                                                    0
carbon dioxide percent in off-gas(CO2outgas:%)                          0
PAA flow(Fpaa:PAA flow (L/h))                                           0
PAA concentration offline(PAA_offline:PAA (g L^{-1}))                   0
Oil flow(Foil:L/hr)                                                     0
NH_3 concentration off-line(NH3_offline:NH3 (g L^{-1}))                 0
Oxygen Uptake Rate(OUR:(g min^{-1}))                                    0
Oxygen in percent in off-gas(O2:O2  (%))                                0
Offline Penicillin concentration(P_offline:P(g L^{-1}))                 0
Offline Biomass concentratio(X_offline:X(g L^{-1}))                     0
Carbon evolution rate(CER:g/h)                                          0
Ammonia shots(NH3_shots:kgs)                                            0
Viscosity(Viscosity_offline:centPoise)                                  0
Fault reference(Fault_ref:Fault ref)                                    0
0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)      0
1- No Raman spec                                                        0
2-PAT control(PAT_ref:PAT ref)                                          0
```

Download the new .csv file

```
newdf.shape
```

```
(113935, 40)
```

```
newdf.to_csv(r'C:\Users\Admin\Desktop\final.csv', index=False)
```

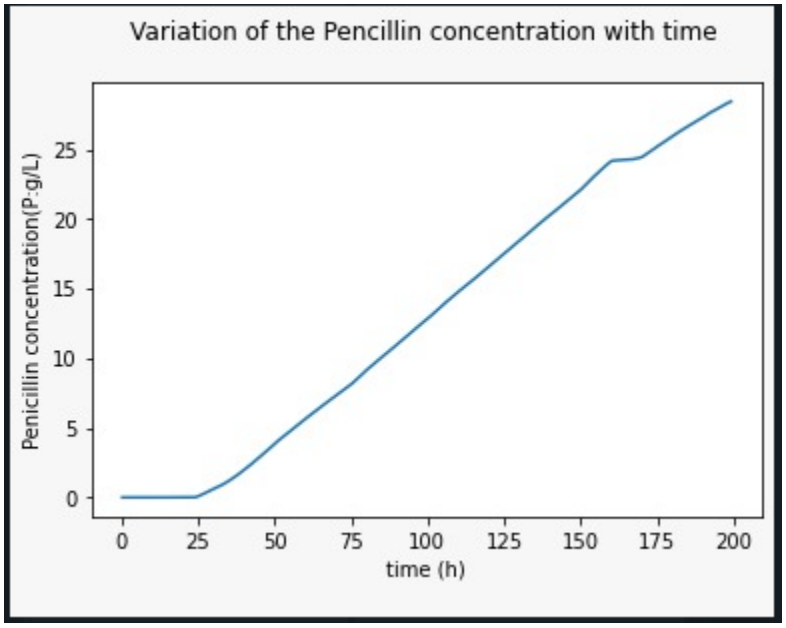
VISUALISATION

1) **Penicillin concentration with time**

```
g1=data['Time (h)']
g1=g1.iloc[0:1000:5]


g2=data['Penicillin concentration(P:g/L)']
g2=g2.iloc[0:1000:5]


plt.plot(g1,g2)
plt.title("Variation of the Penicillin concentration with time \n")
plt.xlabel("time (h)\n")
plt.ylabel("Penicillin concentration(P:g/L)")
```



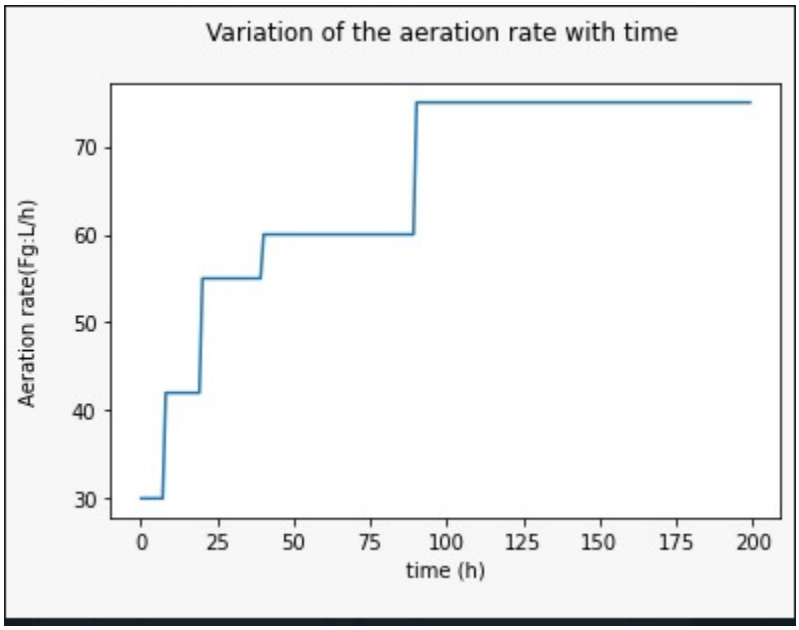
2) **Aeration rate with time**

Aeration is the process by which air is circulated through, mixed with or dissolved in a liquid or substance

```
g1=data['Time (h)']
g1=g1.iloc[0:1000:5]


g2=data['Aeration rate(Fg:L/h) ']
g2=g2.iloc[0:1000:5]


plt.plot(g1,g2)
plt.title("Variation of the Aeration rate with time \n")
plt.xlabel("time (h)\n")
plt.ylabel("Aeration rate(Fg:L/h)")
```

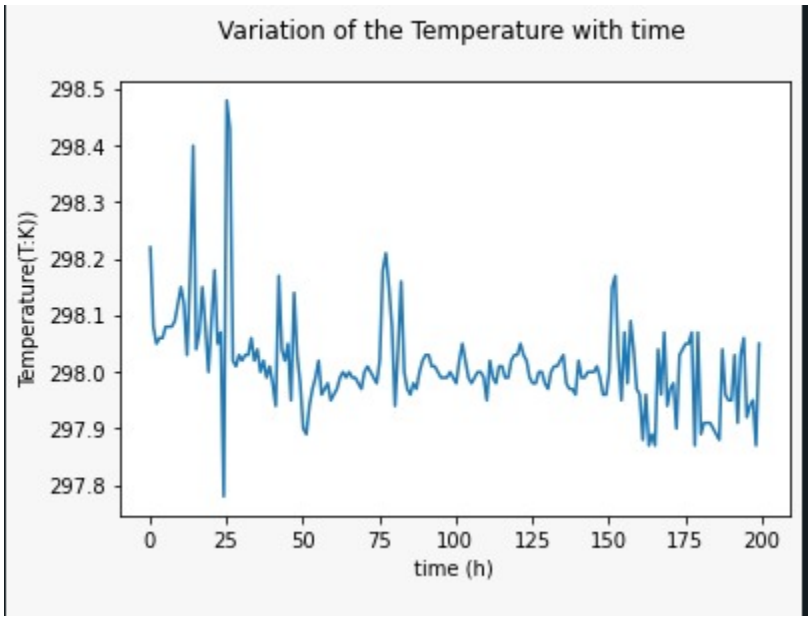


3) Temperature with time

```
g1=data['Time (h)']
g1=g1.iloc[0:100:5]


g2=data['Temperature(T:K)']
g2=g2.iloc[0:100:5]


plt.plot(g1,g2)
plt.title("Variation of temperature with time \n")
plt.xlabel("Time (h) \n")
plt.ylabel("Temperature(T:K) \n")
```

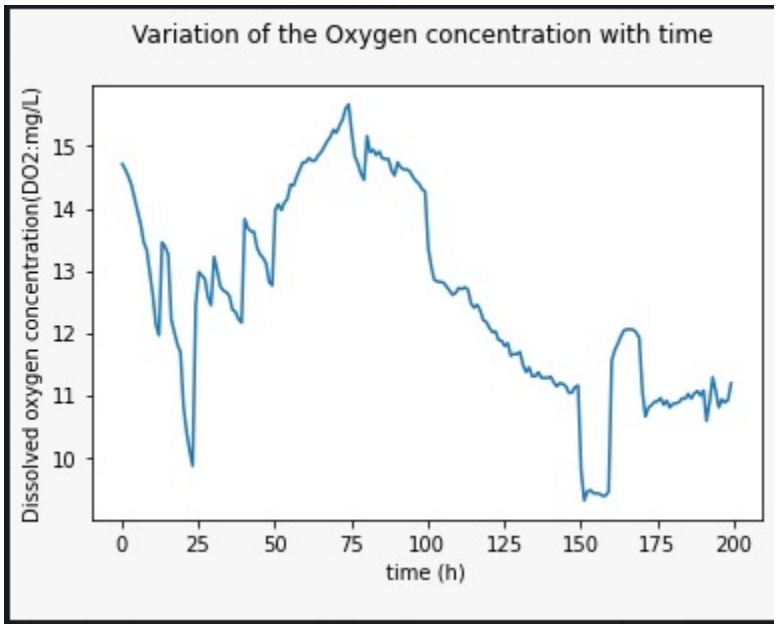


4) Oxygen concentration with time

```
g1=data['Time (h)']
g1=g1.iloc[0:100:5]


g2=data['Dissolved oxygen concentration(DO2:mg/L)']
g2=g2.iloc[0:100:5]


plt.plot(g1,g2)
plt.title("Oxygen concentration with time \n")
plt.xlabel("Time (h) \n")
plt.ylabel("Dissolved oxygen concentration(DO2:mg/L) \n")
```

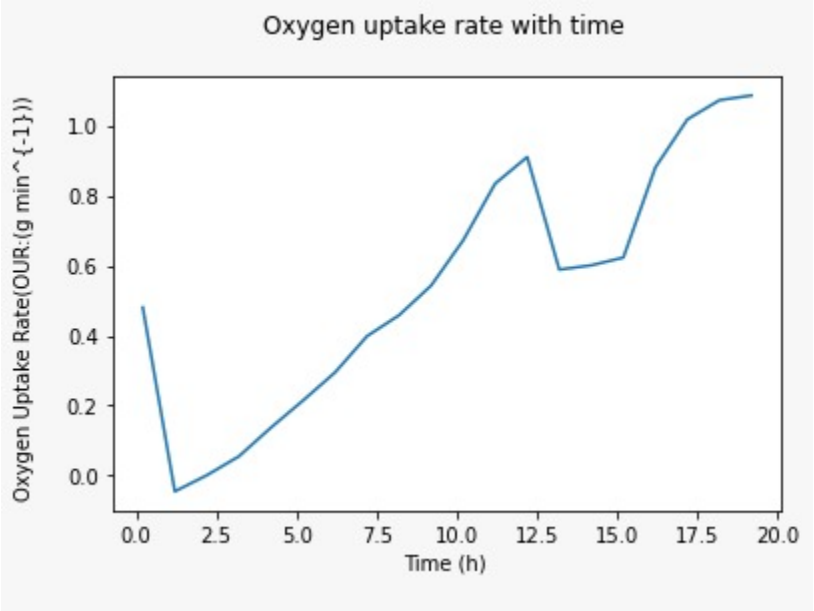


5) Oxygen uptake rate with time

```
g1=data['Time (h)']
g1=g1.iloc[0:100:5]


g2=data['Oxygen Uptake Rate(OUR:(g min^{-1}))']
g2=g2.iloc[0:100:5]


plt.plot(g1,g2)
plt.title("Oxygen uptake rate with time \n")
plt.xlabel("Time (h) \n")
plt.ylabel("Oxygen Uptake Rate(OUR:(g min^{-1}))\n")
```

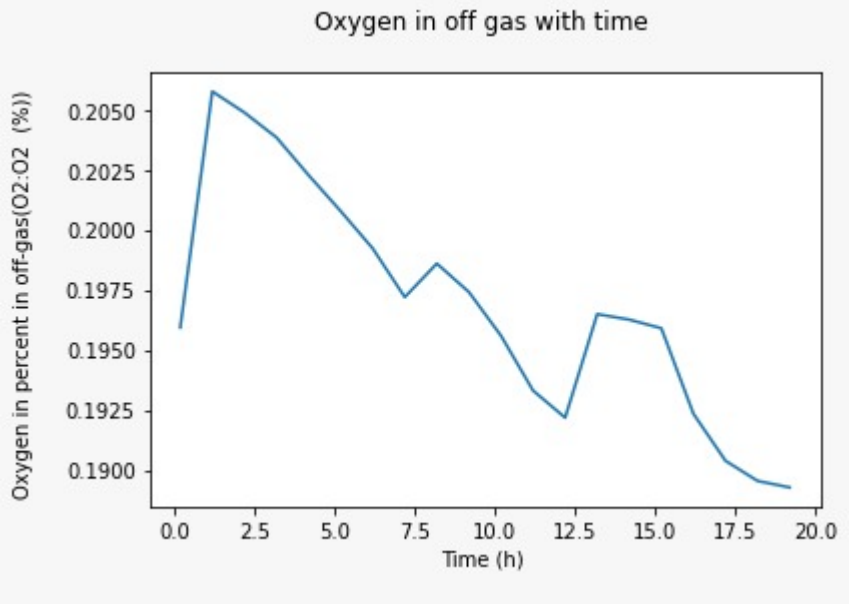


6) Variation of oxygen percent in off gas with time

```
g1=data['Time (h)']
g1=g1.iloc[0:100:5]


g2=data['Oxygen in percent in off-gas(O2:O2  (%))']
g2=g2.iloc[0:100:5]


plt.plot(g1,g2)
plt.title("Oxygen in off gas with time \n")
plt.xlabel("Time (h) \n")
plt.ylabel("Oxygen in percent in off-gas(O2:O2  (%)) \n")
```



QUERIES

1. To find the number of fault batches with the operator driven method

```
test.filter(test['0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)']== 1).groupby('Fault reference(Fault_ref:Fault ref)').count().show()
```

(2) Spark Jobs

```
+-----------------------------------+-----+
|Fault reference(Fault_ref:Fault ref)|count|
+-----------------------------------+-----+
|                                  0|34385|
+-----------------------------------+-----+
```

Command took 2.38 seconds -- by tejusdinesh80@gmail.com at 12/8/2020, 9:46:03 PM on BigData_proj

#command
test.filter(test['0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)']==1).groupby('Fault reference(Fault_ref:Fault ref)').count().show()

We can see that there are 0 fault batches when the production was operator driven.

2. To find the number of fault batches with the operator recipe method

#command
test.filter(test['0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)']==0).groupby('Fault reference(Fault_ref:Fault ref)').count().show()

```
test.filter(test['0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)']== 0).groupby('Fault reference(Fault_ref:Fault ref)').count().show()
```

(2) Spark Jobs

```
+-----------------------------------+-----+
|Fault reference(Fault_ref:Fault ref)|count|
+-----------------------------------+-----+
|                                  0|78294|
|                                  1| 1256|
+-----------------------------------+-----+
```

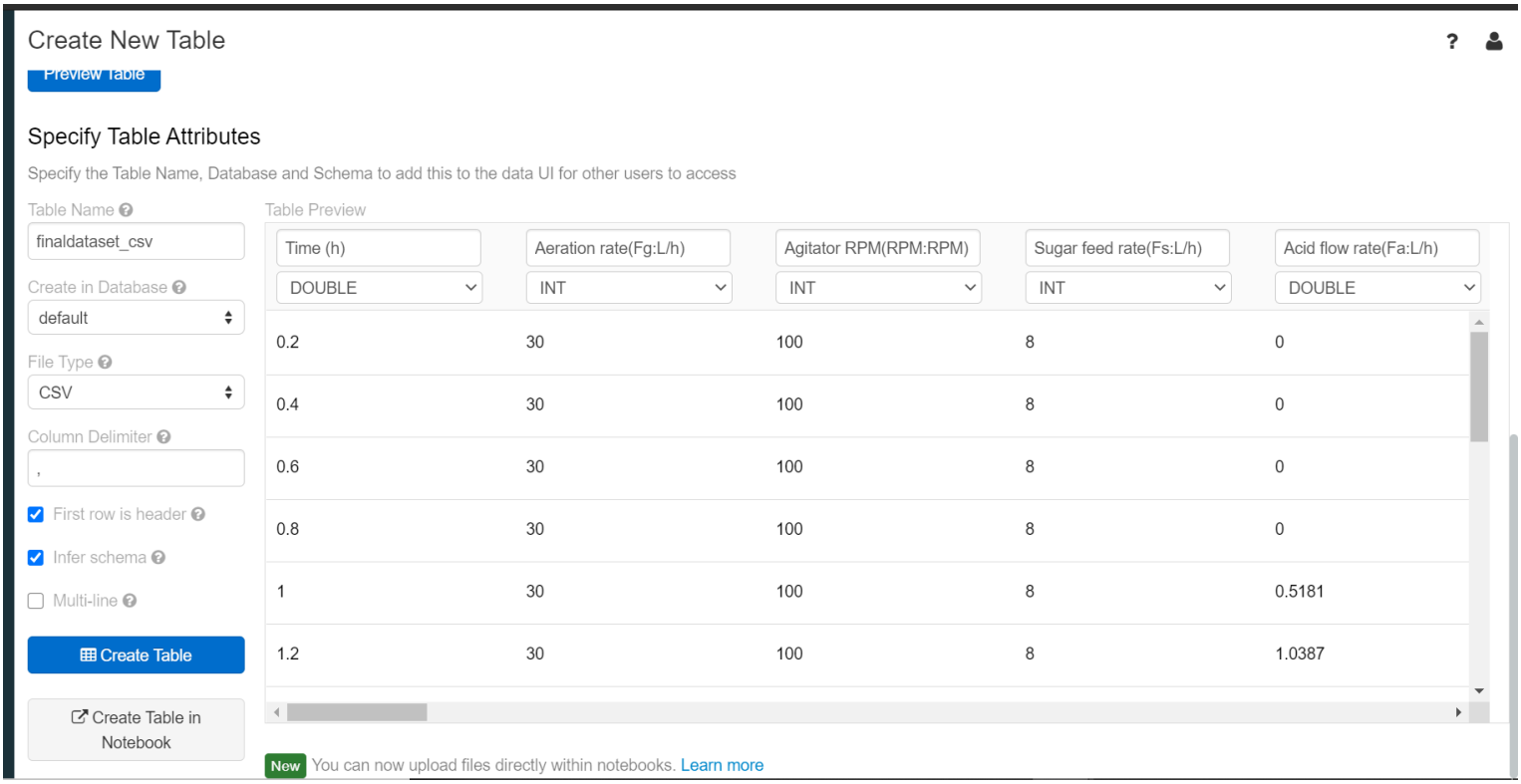
Command took 1.85 seconds -- by tejusdinesh80@gmail.com at 12/8/2020, 9:46:23 PM on BigData_proj

It is evident that there are about 1250 entries where the production of penicillin was faulty.

We can hence conclude that the operator driven method is more reliable than recipe driven in the production of penicillin.

Design of Machine Learning model to predict Penicillin yield

We have used databricks platform for the analysis purpose. Below is the screenshot of uploaded the dataset into the cluster.



Importing necessary packages

```
1  from pyspark.ml.regression import RandomForestRegressor
2  import pandas as pd
3  from pyspark.sql import SparkSession
4  from pyspark.sql import SQLContext
5  from pyspark.ml.feature import VectorAssembler
6  from pyspark.ml import Pipeline
7  from pyspark.ml.feature import VectorAssembler
8  from pyspark.ml.tuning import ParamGridBuilder
9  import numpy as np
10 from pyspark.ml.tuning import CrossValidator
11 from pyspark.ml.evaluation import RegressionEvaluator
12 import matplotlib.pyplot as plt
```

1

Command took 0.47 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:26:14 PM on My Cluster

Starting sparksession and reading the datasets

```
spark = SparkSession.builder.appName('Rand_for').getOrCreate()
```

Command took 0.05 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:26:19 PM on My Cluster

Cmd 3

```
data_stat= spark.read.csv('/FileStore/tables/100_Batches_IndPenSim_Statistics.csv',inferSchema=True,
                      header=True)
data = spark.read.csv('/FileStore/tables/finaldataset.csv',inferSchema=True,
                      header=True)
```

▸ (4) Spark Jobs
▸ data_stat: pyspark.sql.dataframe.DataFrame = [Batch ref: integer, Penicllin_harvested_during_batch(kg): double ... 3 more fields]
▸ data: pyspark.sql.dataframe.DataFrame = [Time (h): double, Aeration rate(Fg:L/h): integer ... 38 more fields]

Command took 5.66 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:26:22 PM on My Cluster

Features vector

The assembler is a vector with the columns that act as features for the prediction. Spark ML's Random Forest class requires that the features are formatted as a single vector. So the first stage is the VectorAssembler. This takes a list of columns that will be included in the new 'features' column. Creating a random forest Regressor model. The only inputs for the Random Forest model are the label and features. Parameters are assigned in the tuning piece.

```
feature_list = []
for col in data.columns:
    if col == 'Penicllin_yield_total (kg)':
        continue
    else:
        feature_list.append(col)

assembler = VectorAssembler(inputCols=feature_list, outputCol="features")
```

Command took 0.10 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:26:31 PM on My Cluster

Randomforest Regressor model object

```
1 rf = RandomForestRegressor(labelCol="Penicllin_yield_total (kg)", featuresCol="features")
```

```
Command took 0.07 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:27:14 PM on My Cluster
```

Two-stage workflow into an ML pipeline

A machine learning **pipeline** is used to help automate machine learning workflows

```
1 pipeline = Pipeline(stages=[assembler, rf])
```

```
Command took 0.04 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:27:15 PM on My Cluster
```

Setting up the parameters, i.e number of trees, and the depth of the tree.

- **numTrees** – Number of trees in the forest. This parameter is usually the most important setting
- **maxDepth** – Max number of levels in each decision tree

```
1 paramGrid = ParamGridBuilder() \
2     .addGrid(rf.numTrees, [int(x) for x in np.linspace(start = 20, stop = 25, num = 3)]) \
3     .addGrid(rf.maxDepth, [int(x) for x in np.linspace(start = 5, stop = 10, num = 3)]) \
4     .build()
```

```
Command took 0.04 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:27:37 PM on My Cluster
```

To evaluate our model and the corresponding “grid” of parameter variables, we use two folds cross-validation. This method randomly partitions the

original sample into two subsamples and uses them for training and validation. In this, we assign our pipeline to the **estimator** argument, our parameter grid to the ***estimatorParamMaps*** argument, and we import Spark ML's ***RegressionEvaluator*** for the evaluator argument.

```
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(labelCol="Penicllin_yield_total (kg)", predictionCol="prediction",
metricName="rmse"),
                          numFolds=2)
```

```
Command took 0.06 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:28:48 PM on My Cluster
```

Splitting of data into train and test

```
(trainingData, testData) = data.randomSplit([0.8, 0.2])
```

```
▸ trainingData:  pyspark.sql.dataframe.DataFrame = [Time (h): double, Aeration rate(Fg:L/h): integer ... 38 more fields]
▸ testData:  pyspark.sql.dataframe.DataFrame = [Time (h): double, Aeration rate(Fg:L/h): integer ... 38 more fields]
Command took 0.08 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:28:50 PM on My Cluster
```

Training the model

```
cvModel = crossval.fit(trainingData)
```

```
▸ (54) Spark Jobs
/databricks/spark/python/pyspark/ml/util.py:762: UserWarning: Cannot find mlflow module. To enable MLfl
  warnings.warn(_MLflowInstrumentation._NO_MLFLOW_WARNING)
```

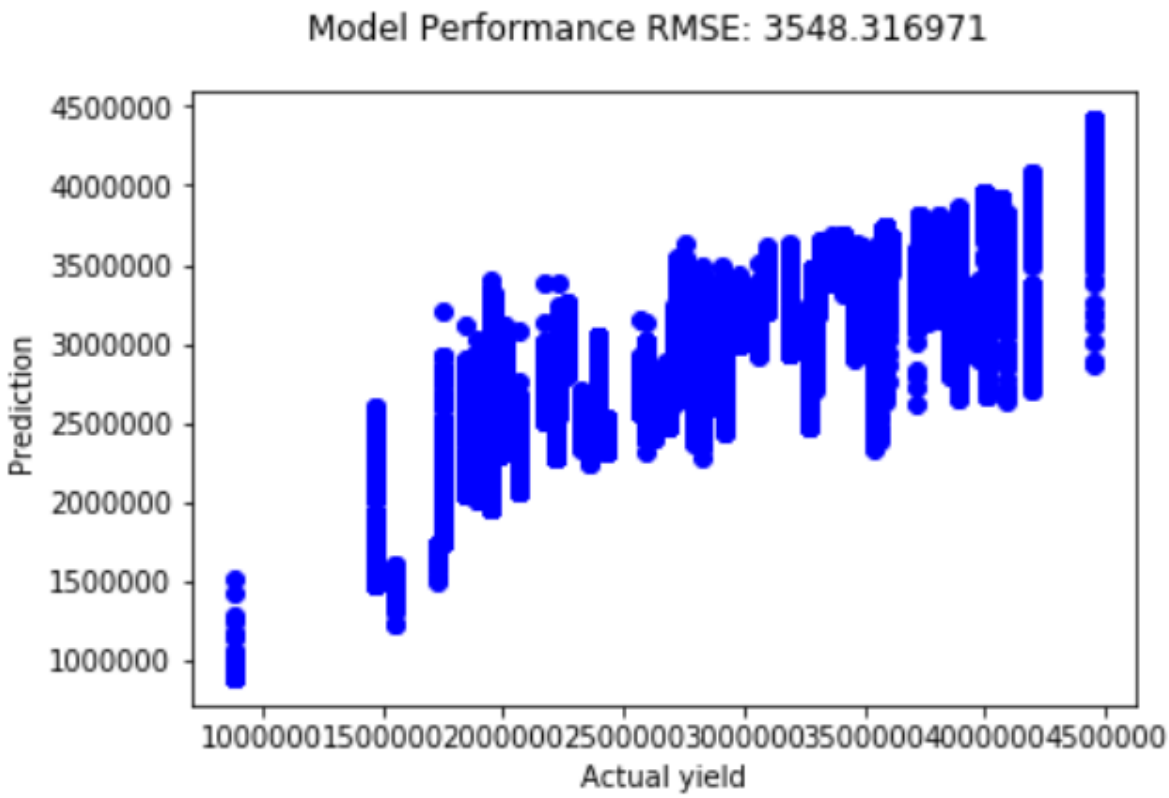
The model is fit using the ***CrossValidator*** we created. This triggers Spark to assess the features and "grow" numerous decision trees using random samples of the training data. The results are recorded for each permutation of the hyperparameters.

Testing the model

```
predictions = cvModel.transform(testData)
```

Evaluating the mode

```
evaluator = RegressionEvaluator(labelCol="Penicllin_yield_total (kg)", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
rfPred = cvModel.transform(data)
rfResult = rfPred.toPandas()
# rfResult
plt.plot(rfResult['Penicllin_yield_total (kg)'], rfResult.prediction, 'bo')
plt.xlabel('Price')
plt.ylabel('Prediction')
plt.suptitle("Model Performance RMSE: %f" % rmse)
plt.show()
```

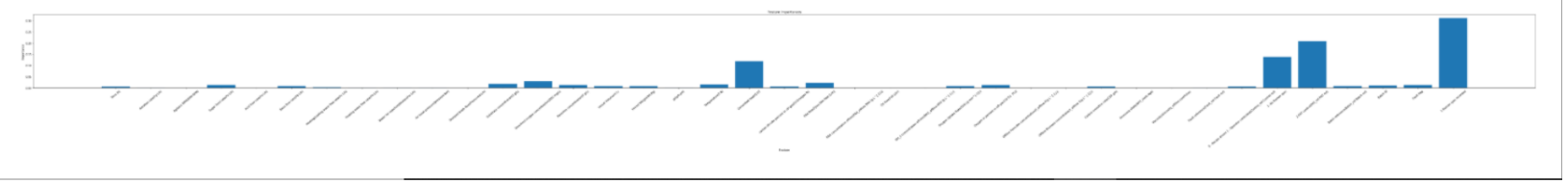


Finding the best model and the important features

```
bestPipeline = cvModel.bestModel
bestModel = bestPipeline.stages[1]

importances = bestModel.featureImportances
x_values = list(range(len(importances)))
# x_values
plt.figure(figsize=(100, 5))
plt.bar(x_values, importances, orientation = 'vertical')
plt.xticks(x_values, feature_list, rotation=40)
plt.ylabel('Importance')
plt.xlabel('Feature')
plt.title('Feature Importances')
```

Out[13]:



Although the graph is unclear because of the size of the graph, it can be observed that the following columns played a major role prediction of penicillin yield

- Temperature(T:K)
- Generated heat(Q:kJ)
- Sugar feed rate(Fs:L/h)
- Base flow rate(Fb:L/h)
- Substrate concentration(S:g/L)
- Dissolved oxygen concentration(DO2:mg/L)
- Penicillin concentration(P:g/L)
- PAA flow(Fpaa:PAA flow (L/h))
- NH_3 concentration off-line(NH3_offline:NH3 (g L^{-1}))
- Oxygen Uptake Rate(OUR:(g min^{-1}))
- 1-Raman spec recorded
- 0 - Recipe driven 1 - Operator controlled(Control_ref:Control ref)
- 1- No Raman spec
- 2-PAT control(PAT_ref:PAT ref)

Investigating which parameters performed best. Our ‘Best Model’ object has a series of “get” parameter functions that select out the parameter values which have the highest performance.

```
1 print('numTrees - ', bestModel.getNumTrees)
2 print('maxDepth - ', bestModel.getOrDefault('maxDepth'))
```

```
numTrees -  22
maxDepth -  10
```

Command took 0.04 seconds -- by tejusdinesh80@gmail.com at 12/9/2020, 12:37:35 PM on My Cluster

With the default number of trees(20) and depth(5) the rmse value was found to be 5547.836039742647

Conclusion:

- Based on our results we believed that the approach that is the best suitable for the production of penicillin is the Operator controlled approach with raman spectroscopy.
- Lot of factors involved in production may seem to have a very insignificant role but they’re rather crucial.
- Due to this reason the best parameter method (ruling out the unwanted parameters) is not possible.