Machine Learning Approach For Predictive Maintenance Aircraft Engine Using IBM Watson Studio

Team Members

1. Ashish Santoshkumar Nair 19BCG10028
2. Satvik Shetty 19BAI10051
3. Sairam Pimple 19BCY10174
4. Ranjan Kudesia 19BCG10037

Introduction

1.1 Overview

The project uses a Machine Learning Approach for predictive Maintenance Aircraft Engine Using IBM Watson Studio.
For this project a combination of Python,Data Analysis, Exploratory Data Analysis, Data Preprocessing Techniques, Classification and Regression Algorithms

1.2 Purpose

This project can be used to predict aircraft engine failures before it happens.This can save a lot of lives, money and time.
Finding out the possibility of a system failure can be achieved with this project.

Literature Survey

2.1 Existing Problem:

An engine failure occurs when an engine unexpectedly stops producing power due to a malfunction other than fuel exhaustion. It often applies for aircraft, but other turbine engines can fail, like ground-based turbines used in power plants or combined diesel and gas vessels and vehicles.

Engine failure is highly risky and needs a lot of time for repair. Unexpected failure leads to loss of money and time. Predicting the failure prior, will save time, effort, money and sometimes even lives.

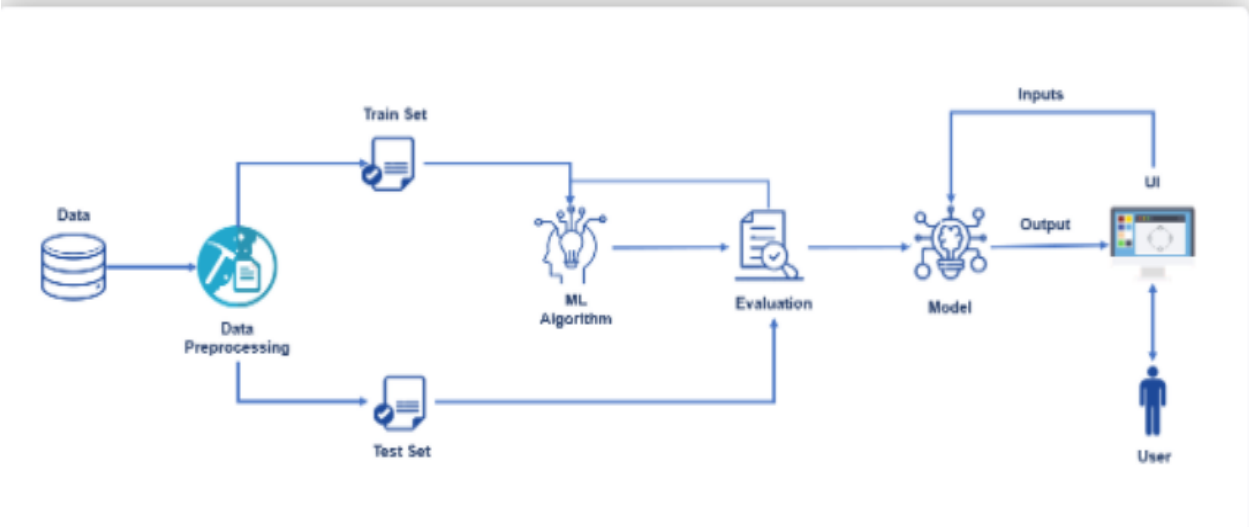
2.2 Proposed Solution:

The project aims to predict the failure of an engine by using Machine Learning to save loss of time & money thus improving productivity.

3.Theoretical Analysis

3.1 Block Diagram

The below diagram is the technical architecture of the whole project



3.2 Hardware/Software requirements:

1. Jupyter Notebook
2. Spyder
3. Visual studios
4. IBM Cloud
    a. IBM Watson studios
5. Flask

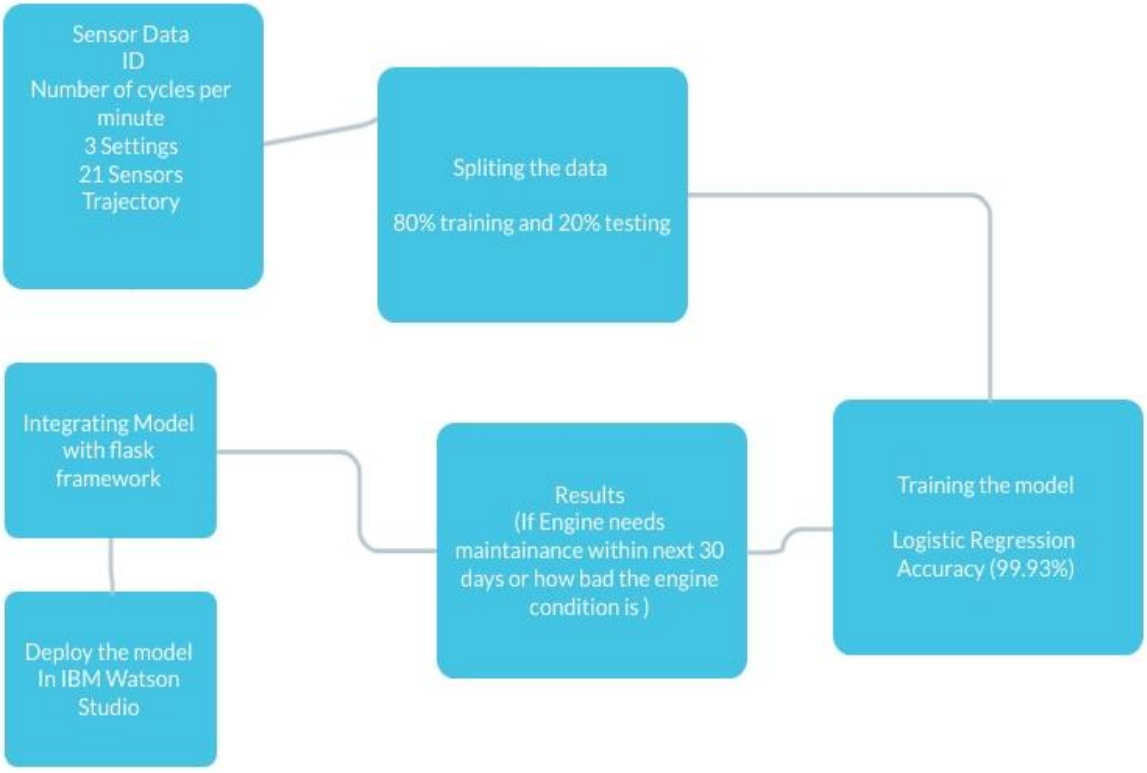
6. Packages:

a. Pandas
b. numpy
c. scikit -learn
d. matplotlib & seaborn
e. Pickle

4.Experimental Investigations:

1. Here, we are considering a dataset which has operational setting data and the data came from different sensors of an engine
2. The dataset contains 20631 data points inside the train dataset and we have 3 different datasets for train, test and truth.
3. We first built a model in a jupyter notebook.
4. The data is split into training and testing set in 8:2 ratio.
5. The model is trained using Logistic Regression .
6. Logistic Regression yields the most accurate result with around 99.93 percent accuracy .
7. Then we integrate the machine learning model with the flask to build the web application.
8. The model is then deployed using IBM Watson Studio.

5.Flowchart:



6.Result:

Once the proposed– model is implemented, you will be welcomed by a UI where the input received from 21 different engine sensors and 3 different settings can be either manually typed in and see if the engine needs a maintenance or not or we also have automatic system from which data from all the sensors can be collected which are available on our dataset and predict the condition of the engine and see if the engine needs a maintenance or not.

7.Advantages:

1. Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

2. No human intervention needed (automation)

With ML, you don't need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

3. Continuous Improvement

As ML Algorithms gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

4. Handling multidimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multidimensional and multi-variety, and they can do this in dynamic or uncertain environments.

Disadvantages:

1. Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

2. Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

4. High error-susceptibility

Machine learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

8.Applications:

This solution can be applied anywhere there is a use of an aircraft engine.

Commercial planes,passenger planes or even fighter planes-this prediction system can be used.Its application depends on the fact that the machine uses an aircraft engine.

9.Conclusion:

Engine failure is highly risky and needs a lot of time for repair. Unexpected failure leads to loss of money and time. Predicting the failure prior, will save time, effort, money and sometimes even lives. The failure can be detected by installing the sensors and keeping a track of the values. The failure detection and predictive maintenance can be for any device, out of which we will be dealing with the engine failure for a threshold number of days.
The project aims to predict the failure of an engine by using Machine Learning to save loss of time & money thus improving productivity

This solution helps save so much time and money by simply just entering the values we get from the different sensors which are already in place for a commercial plane and this data simply putting in our website can give you a 99.93 percent accuracy about the condition of your aircraft engine. Whether the engine is in good condition or not or if it needs a maintenance in the next 30 days to get the engine back in good condition and this can help save lives of countless people by preventing any engine malfunction midflight.

10.Future Scope:

Currently our model is using sensor data received from airplanes engine and training the model using these datasets to predict if the engine is in good condition or not. We have to manually enter all the sensor data received from all these sensors in our website for manual prediction. Although this is very accurate and trustworthy , in future we can even make it better by configuring all the different sensors in such a way that all the data is directly sent to our website and we can monitor the engine constantly at particular intervals of time and we will immediately come to know if some problem is there with the engine and get it fixed as soon as possible. .

11.Bibiliography:

1.https://data-flair.training/blogs/advantages-and-disadvantages-of-machine-learning/

https://analyticsindiamag.com/machine-learning-for-predictive-maintenance-key-approaches-techniques-to-consider/

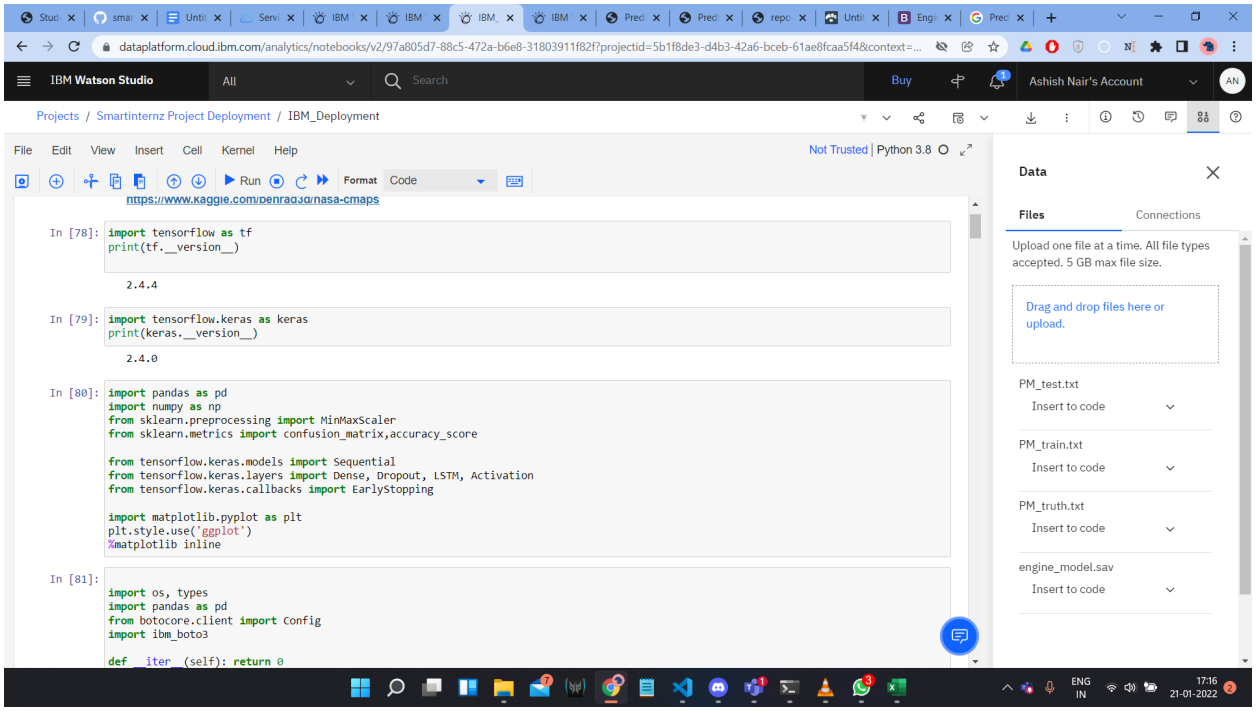
https://ieeexplore.ieee.org/document/9289466

https://aerospace.honeywell.com/us/en/learn/about-us/blogs/moving-beyond-the-hype-of-predictive-maintenance

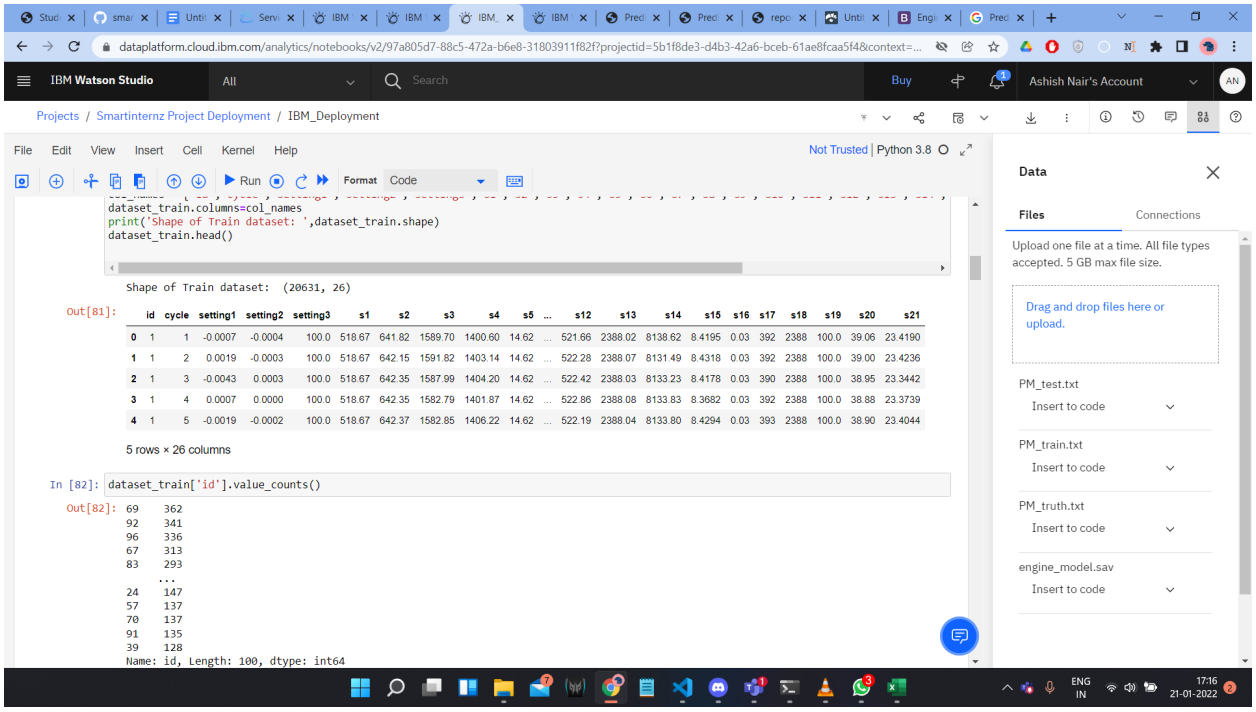
https://www.infoq.com/articles/machine-learning-techniques-predictive-maintenance/

Appendix

a. Source Code



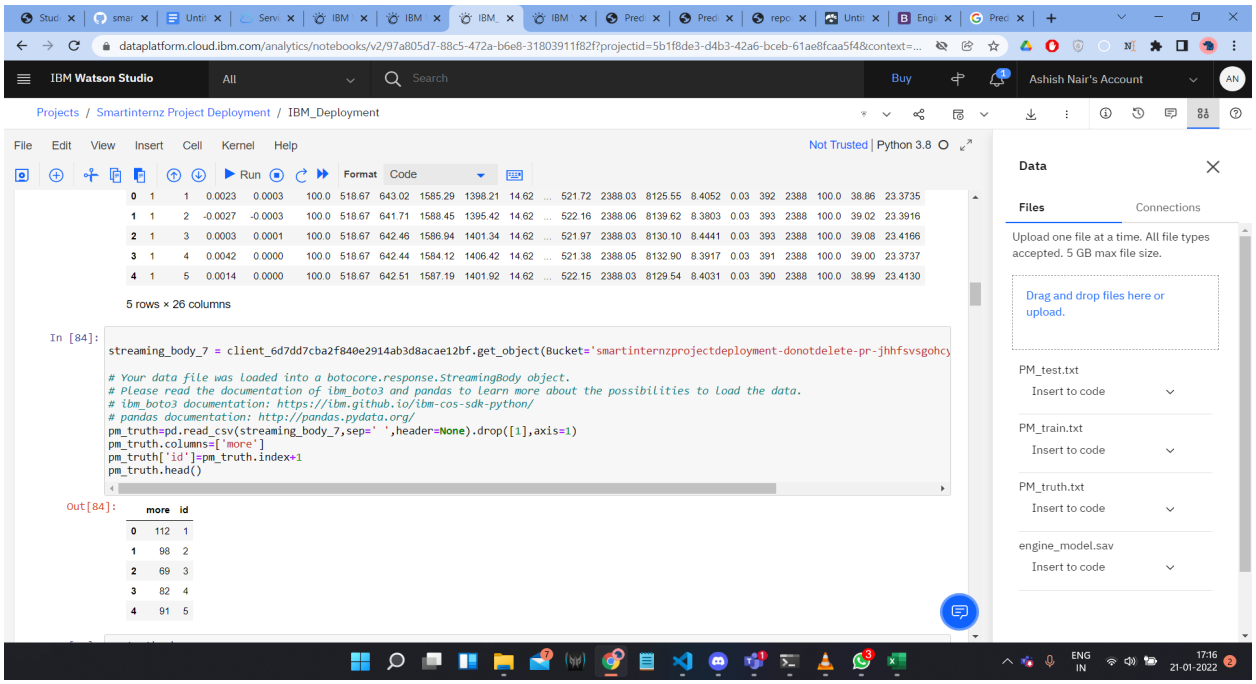
```
https://www.kaggle.com/penrag93/nasa-cmaps

In [78]:
import tensorflow as tf
print(tf.__version__)

    2.4.4

In [79]:
import tensorflow.keras as keras
print(keras.__version__)

    2.4.0

In [80]:
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix,accuracy_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Activation
from tensorflow.keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline

In [81]:

import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0
```



```
dataset_train.columns=col_names
print('Shape of Train dataset: ',dataset_train.shape)
dataset_train.head()

    Shape of Train dataset:  (20631, 26)

Out[81]:
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

```
    5 rows × 26 columns

In [82]:
dataset_train['id'].value_counts()

Out[82]: 69    362
         92    341
         96    336
         67    313
         83    293
              ...
         24    147
         57    137
         70    137
         91    135
         39    128
         Name: id, Length: 100, dtype: int64
```



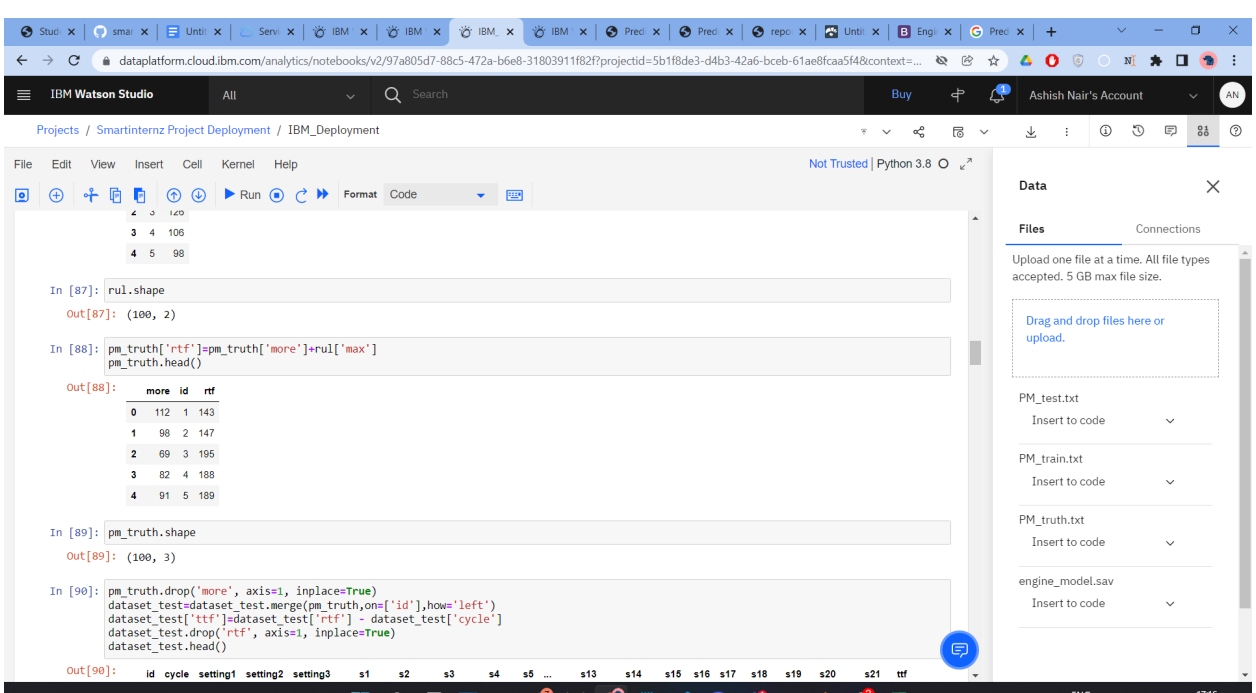
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	521.97	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	521.38	2388.05	8132.90	8.3917	0.03	391	2388	100.0	39.00	23.3737
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	522.15	2388.03	8129.54	8.4031	0.03	390	2388	100.0	38.99	23.4130

5 rows × 26 columns

In [84]:
```
streaming_body_7 = client_6d7dd7cba2f840e2914ab3d8acae12bf.get_object(Bucket='smartinternzprojectdeployment-donotdelete-pr-jhhfsvsgohcy

# Your data file was loaded into a botocore.response.StreamingBody object.
# Please read the documentation of ibm_boto3 and pandas to learn more about the possibilities to load the data.
# ibm_boto3 documentation: https://ibm.github.io/ibm-cos-sdk-python/
# pandas documentation: http://pandas.pydata.org/
pm_truth=pd.read_csv(streaming_body_7,sep=' ',header=None).drop([1],axis=1)
pm_truth.columns=['more']
pm_truth['id']=pm_truth.index+1
pm_truth.head()
```

Out[84]:

	more	id
0	112	1
1	98	2
2	69	3
3	82	4
4	91	5



3	4	106
4	5	98

In [87]:
```
rul.shape
```

Out[87]: (100, 2)

In [88]:
```
pm_truth['rtf']=pm_truth['more']+rul['max']
pm_truth.head()
```

Out[88]:

	more	id	rtf
0	112	1	143
1	98	2	147
2	69	3	195
3	82	4	188
4	91	5	189

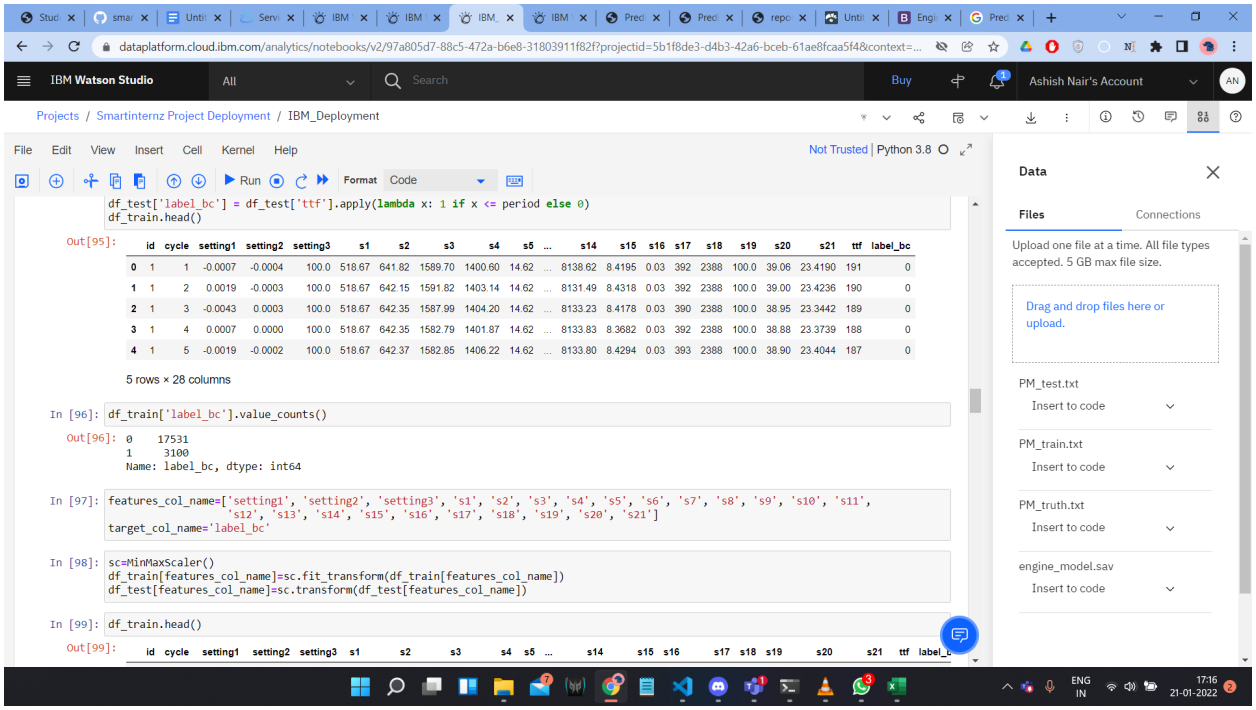
In [89]:
```
pm_truth.shape
```

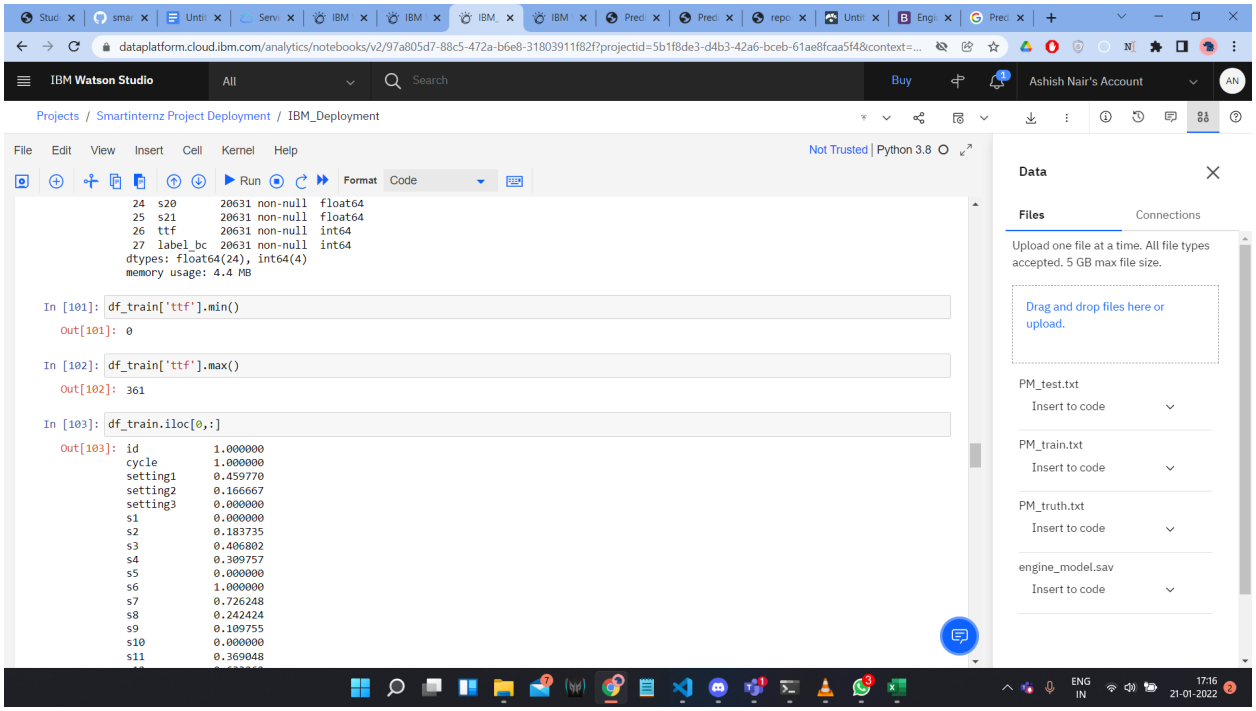
Out[89]: (100, 3)

In [90]:
```
pm_truth.drop('more', axis=1, inplace=True)
dataset_test=dataset_test.merge(pm_truth,on=['id'],how='left')
dataset_test['ttf']=dataset_test['rtf'] - dataset_test['cycle']
dataset_test.drop('rtf', axis=1, inplace=True)
dataset_test.head()
```

Out[90]:

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s13	s14	s15	s16	s17	s18	s19	s20	s21	ttf


IBM Watson Studio
Projects / Smartinternz Project Deployment / IBM_Deployment
File Edit View Insert Cell Kernel Help
Not Trusted | Python 3.8
Run Format Code
df_test['label_bc'] = df_test['ttf'].apply(lambda x: 1 if x <= period else 0)
df_train.head()
Out[95]:
id cycle setting1 setting2 setting3 s1 s2 s3 s4 s5 ... s14 s15 s16 s17 s18 s19 s20 s21 ttf label_bc
0 1 1 -0.0007 -0.0004 100.0 518.67 641.82 1589.70 1400.60 14.62 ... 8138.62 8.4195 0.03 392 2388 100.0 39.06 23.4190 191 0
1 1 2 0.0019 -0.0003 100.0 518.67 642.15 1591.82 1403.14 14.62 ... 8131.49 8.4318 0.03 392 2388 100.0 39.00 23.4236 190 0
2 1 3 -0.0043 0.0003 100.0 518.67 642.35 1587.99 1404.20 14.62 ... 8133.23 8.4178 0.03 390 2388 100.0 38.95 23.3442 189 0
3 1 4 0.0007 0.0000 100.0 518.67 642.35 1582.79 1401.87 14.62 ... 8133.83 8.3682 0.03 392 2388 100.0 38.88 23.3739 188 0
4 1 5 -0.0019 -0.0002 100.0 518.67 642.37 1582.85 1406.22 14.62 ... 8133.80 8.4294 0.03 393 2388 100.0 38.90 23.4044 187 0
5 rows × 28 columns
In [96]: df_train['label_bc'].value_counts()
Out[96]: 0 17531
1 3100
Name: label_bc, dtype: int64
In [97]: features_col_name=['setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21']
target_col_name='label_bc'
In [98]: sc=MinMaxScaler()
df_train[features_col_name]=sc.fit_transform(df_train[features_col_name])
df_test[features_col_name]=sc.transform(df_test[features_col_name])
In [99]: df_train.head()
Out[99]: id cycle setting1 setting2 setting3 s1 s2 s3 s4 s5 ... s14 s15 s16 s17 s18 s19 s20 s21 ttf label_bc
Data
Files
Connections
Upload one file at a time. All file types accepted. 5 GB max file size.
Drag and drop files here or upload.
PM_test.txt
Insert to code
PM_train.txt
Insert to code
PM_truth.txt
Insert to code
engine_model.sav
Insert to code


IBM Watson Studio
Projects / Smartinternz Project Deployment / IBM_Deployment
File Edit View Insert Cell Kernel Help
Not Trusted | Python 3.8
Run Format Code
24 s20 20631 non-null float64
25 s21 20631 non-null float64
26 ttf 20631 non-null int64
27 label_bc 20631 non-null int64
dtypes: float64(24), int64(4)
memory usage: 4.4 MB
In [101]: df_train['ttf'].min()
Out[101]: 0
In [102]: df_train['ttf'].max()
Out[102]: 361
In [103]: df_train.iloc[0,:]
Out[103]: id 1.000000
cycle 1.000000
setting1 0.459770
setting2 0.166667
setting3 0.000000
s1 0.000000
s2 0.183735
s3 0.406802
s4 0.309757
s5 0.000000
s6 1.000000
s7 0.726248
s8 0.242424
s9 0.109755
s10 0.000000
s11 0.369048
Data
Files
Connections
Upload one file at a time. All file types accepted. 5 GB max file size.
Drag and drop files here or upload.
PM_test.txt
Insert to code
PM_train.txt
Insert to code
PM_truth.txt
Insert to code
engine_model.sav
Insert to code

dataplatform.cloud.ibm.com/analytics/notebooks/v2/97a805d7-88c5-472a-b6e8-31803911f82f?projectid=5b1f8de3-d4b3-42a6-bceb-61ae8fcaa5f4&context=...

IBM Watson Studio | All | Search | Buy | Ashish Nair's Account

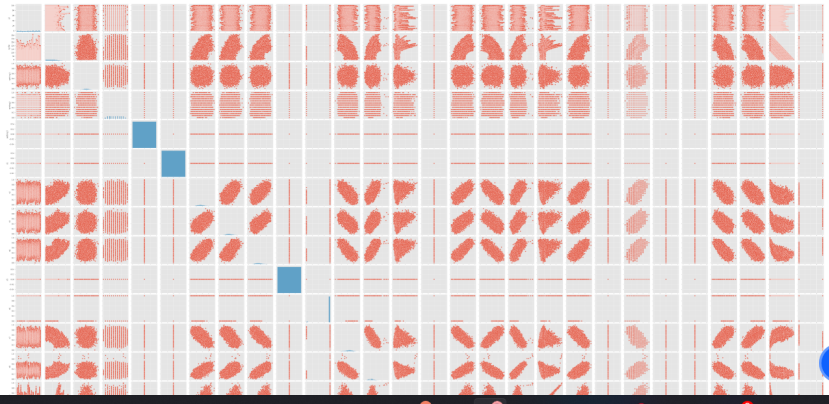
Projects / Smartinternz Project Deployment / IBM_Deployment

File Edit View Insert Cell Kernel Help | Not Trusted | Python 3.8

Run | Format | Code

```
In [104]: import seaborn as sns
          sns.pairplot(df_train)
```

```
Out[104]: <seaborn.axisgrid.PairGrid at 0x7f18a37ee880>
```



Data

Files | Connections

Upload one file at a time. All file types accepted. 5 GB max file size.

Drag and drop files here or upload.

PM_test.txt
Insert to code

PM_train.txt
Insert to code

PM_truth.txt
Insert to code

engine_model.sav
Insert to code

ENG IN 17:16 21-01-2022

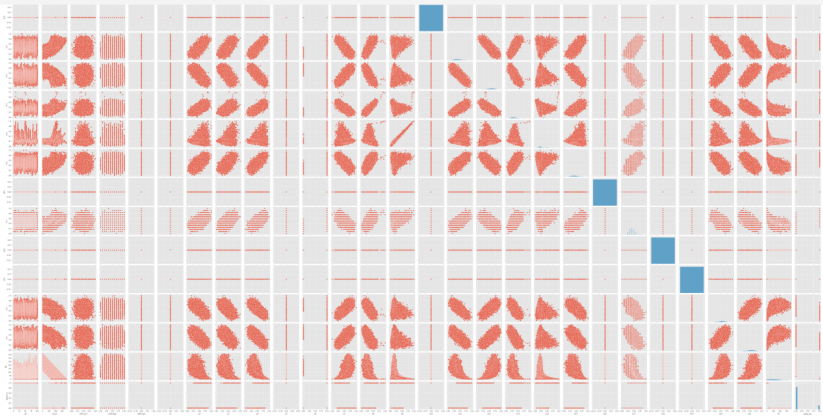
dataplatform.cloud.ibm.com/analytics/notebooks/v2/97a805d7-88c5-472a-b6e8-31803911f82f?projectid=5b1f8de3-d4b3-42a6-bceb-61ae8fcaa5f4&context=...

IBM Watson Studio | All | Search | Buy | Ashish Nair's Account

Projects / Smartinternz Project Deployment / IBM_Deployment

File Edit View Insert Cell Kernel Help | Not Trusted | Python 3.8

Run | Format | Code



```
In [105]: x_train = df_train.iloc[:,:-1].values
          y_train = df_train.iloc[:,-1:].values
```

Data

Files | Connections

Upload one file at a time. All file types accepted. 5 GB max file size.

Drag and drop files here or upload.

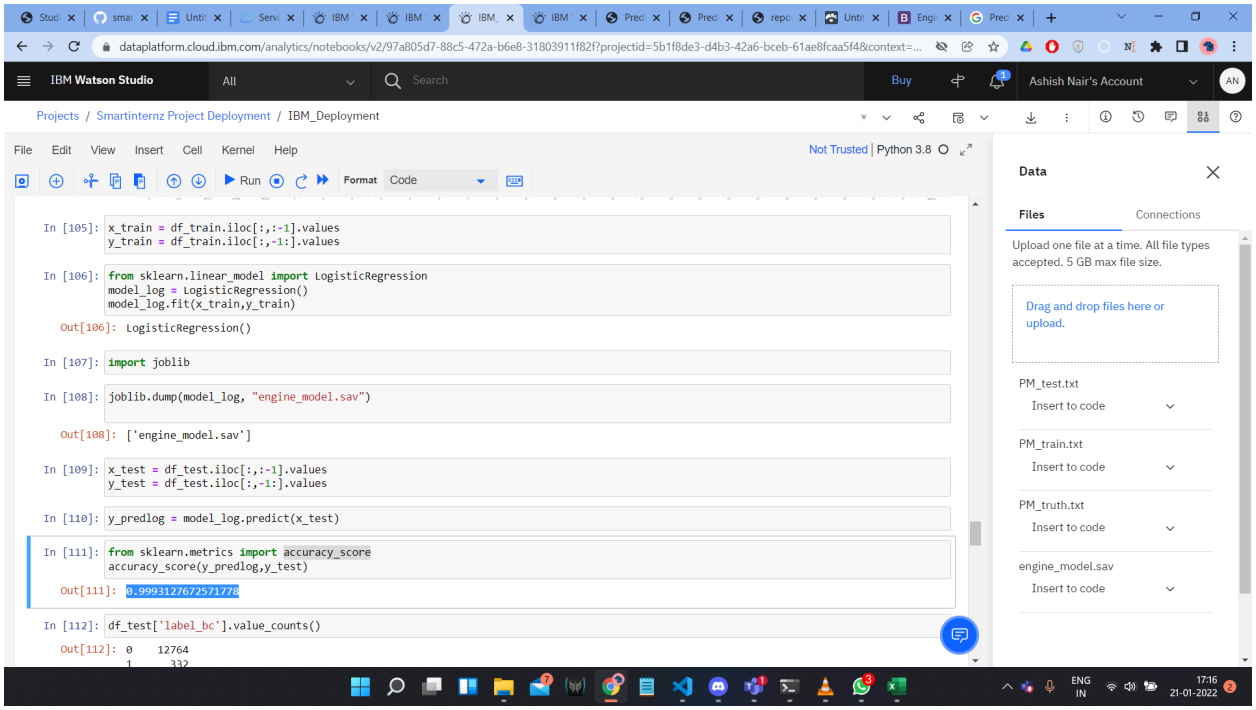
PM_test.txt
Insert to code

PM_train.txt
Insert to code

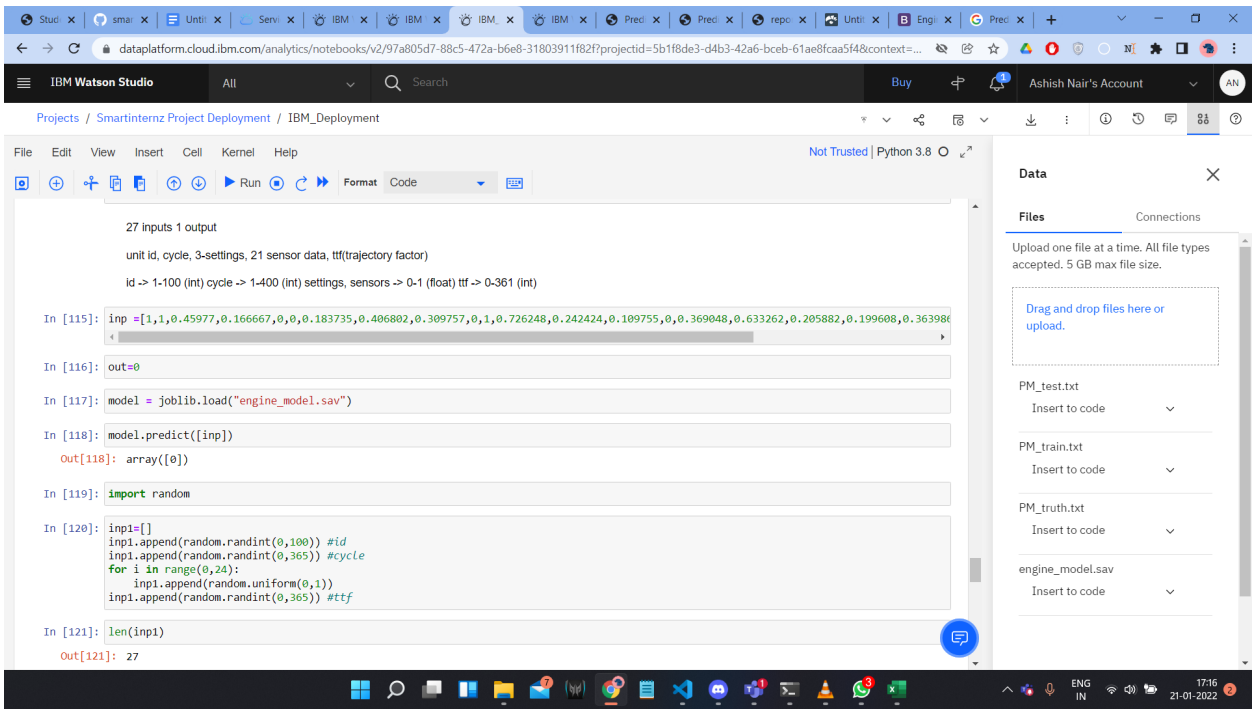
PM_truth.txt
Insert to code

engine_model.sav
Insert to code

ENG IN 17:16 21-01-2022



```
In [105]: x_train = df_train.iloc[:,:-1].values
          y_train = df_train.iloc[:,-1:].values

In [106]: from sklearn.linear_model import LogisticRegression
          model_log = LogisticRegression()
          model_log.fit(x_train,y_train)

Out[106]: LogisticRegression()

In [107]: import joblib

In [108]: joblib.dump(model_log, "engine_model.sav")

Out[108]: ['engine_model.sav']

In [109]: x_test = df_test.iloc[:,:-1].values
          y_test = df_test.iloc[:,-1:].values

In [110]: y_predlog = model_log.predict(x_test)

In [111]: from sklearn.metrics import accuracy_score
          accuracy_score(y_predlog,y_test)

Out[111]: 0.9993127672571778

In [112]: df_test['label_bc'].value_counts()

Out[112]: 0    12764
          1      332
```

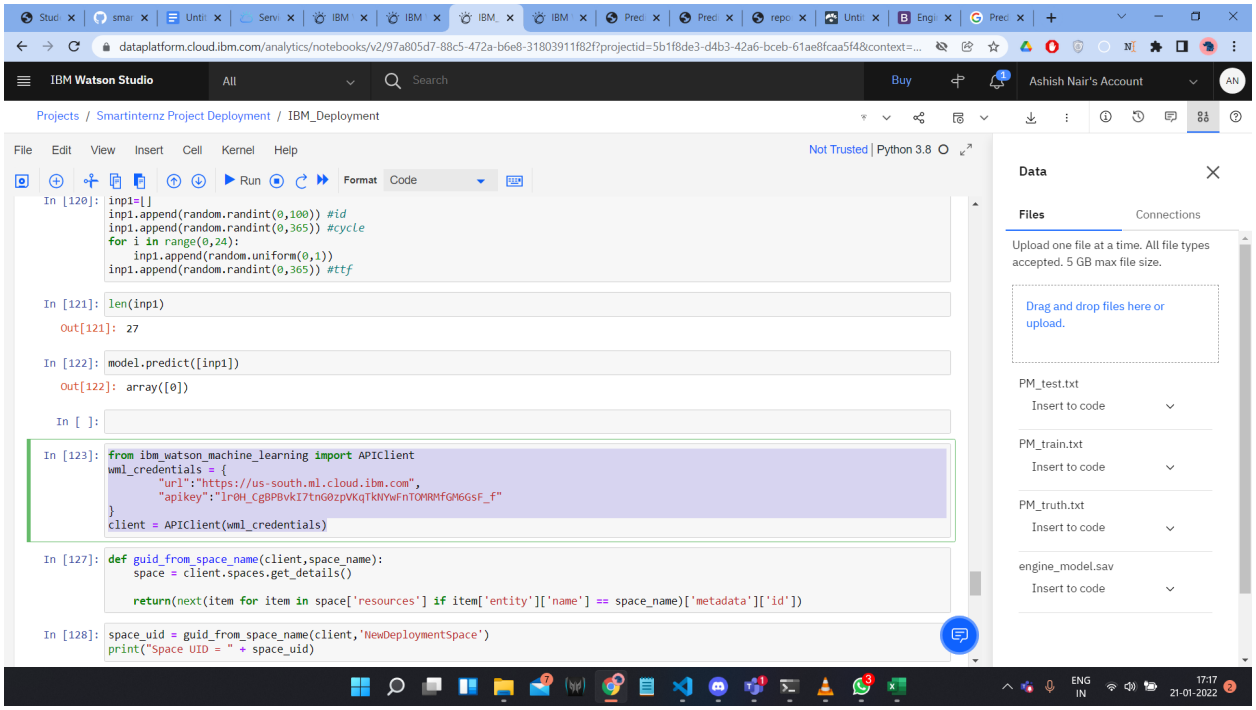


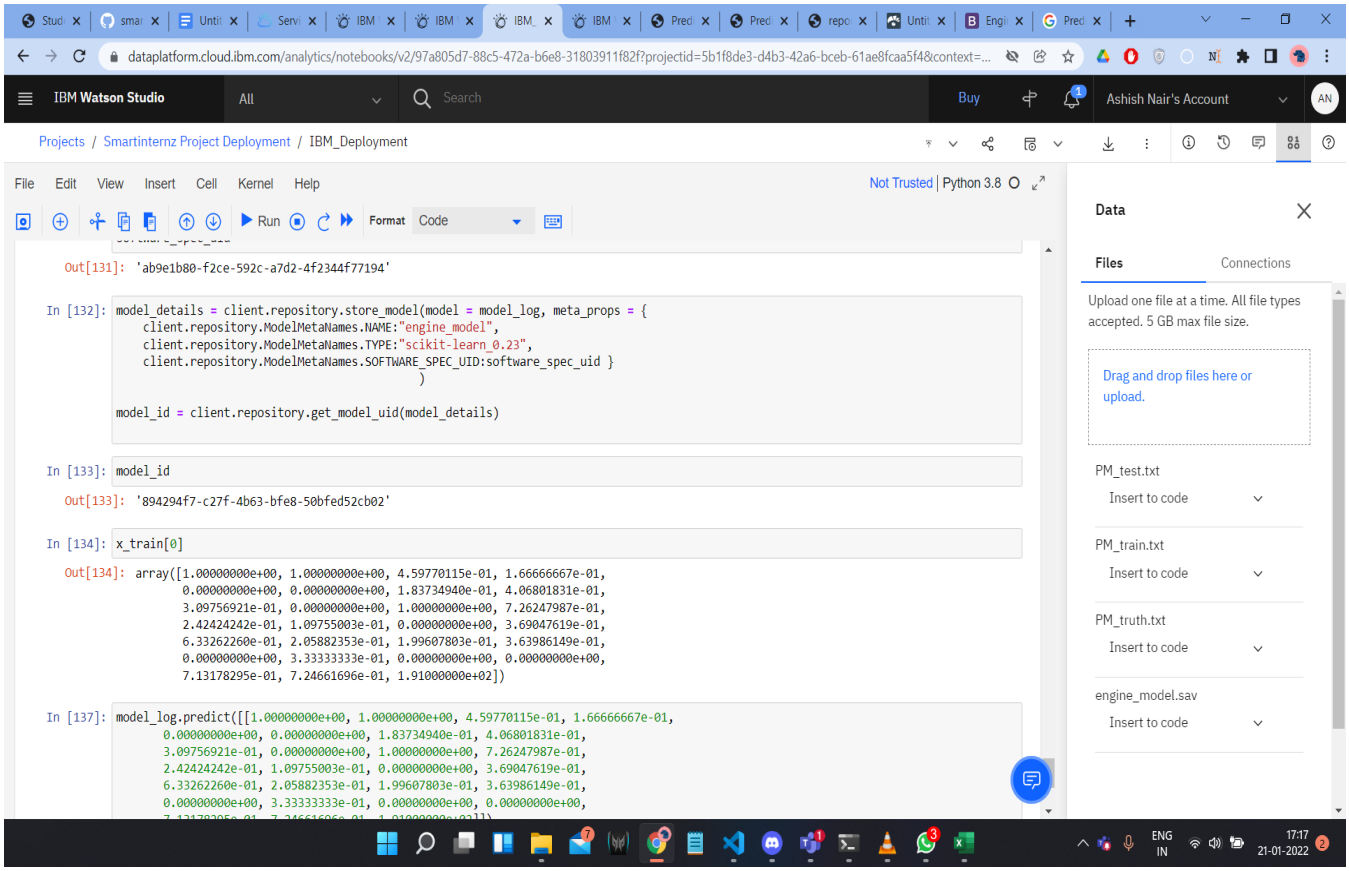
27 inputs 1 output

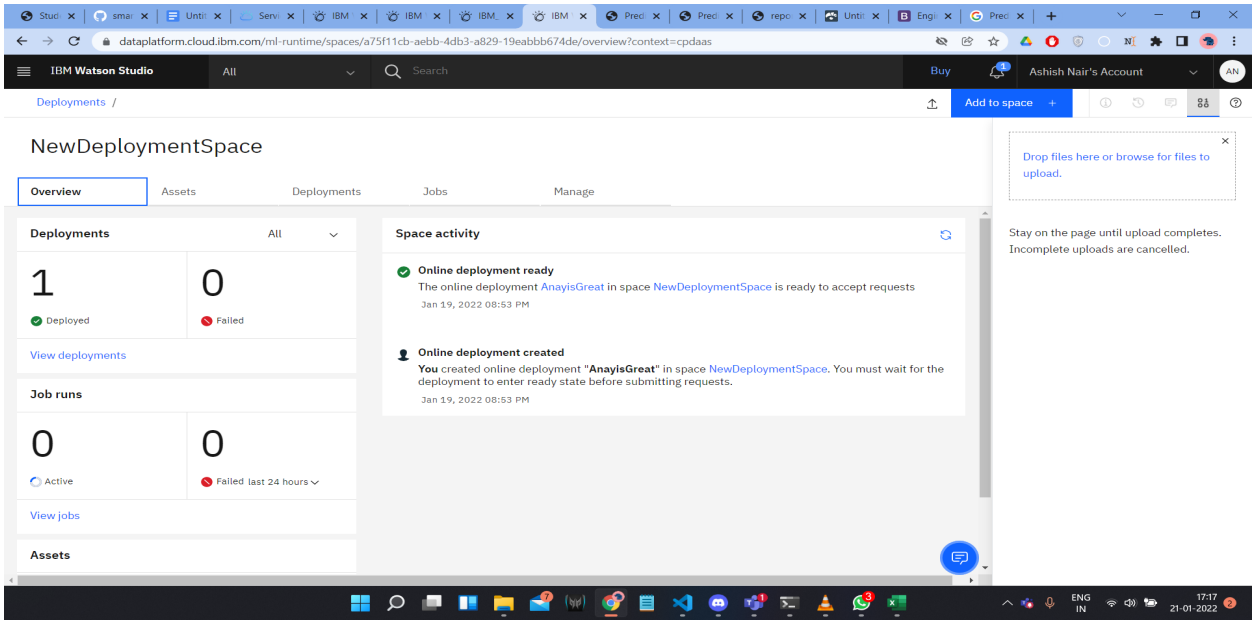
unit id, cycle, 3-settings, 21 sensor data, ttf(trajectory factor)

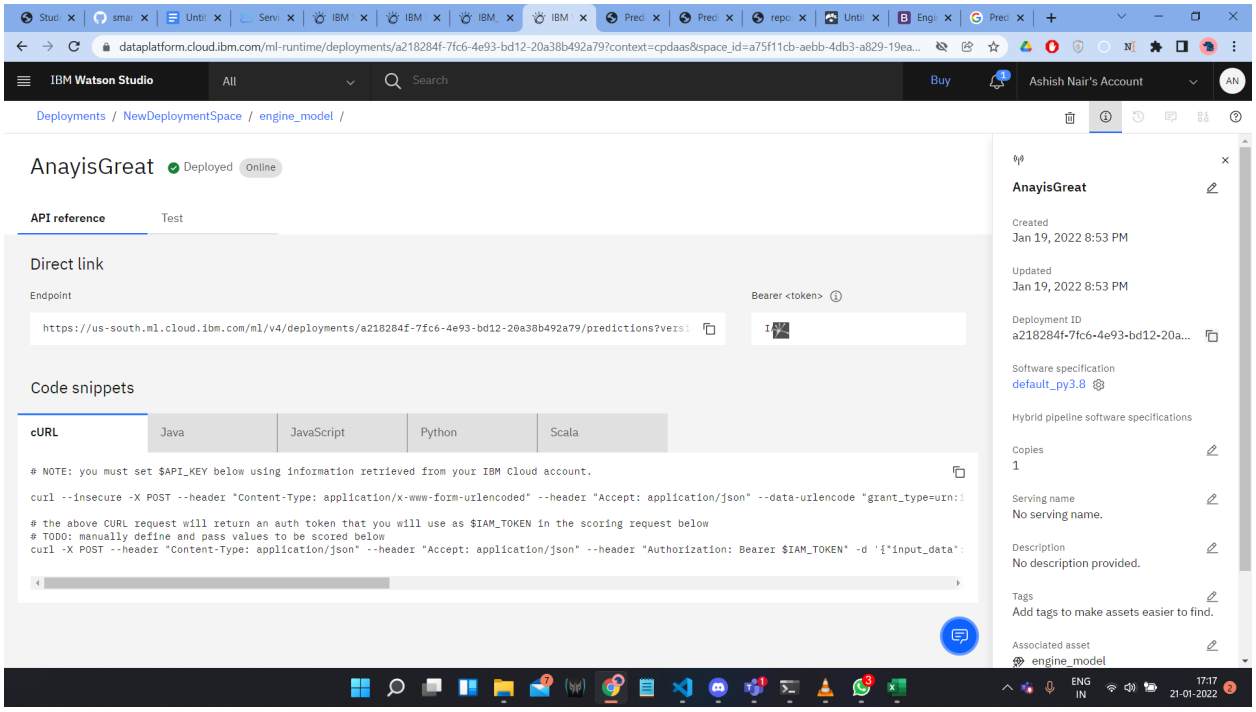
id -> 1-100 (int) cycle -> 1-400 (int) settings, sensors -> 0-1 (float) ttf -> 0-361 (int)

```
In [115]: inp =[1,1,0.45977,0.166667,0,0,0.183735,0.406802,0.309757,0,1,0.726248,0.242424,0.109755,0,0.369048,0.633262,0.205882,0.199608,0.36398

In [116]: out=0

In [117]: model = joblib.load("engine_model.sav")

In [118]: model.predict([inp])

Out[118]: array([0])

In [119]: import random

In [120]: inp1=[]
          inp1.append(random.randint(0,100)) #id
          inp1.append(random.randint(0,365)) #cycle
          for i in range(0,24):
              inp1.append(random.uniform(0,1))
          inp1.append(random.randint(0,365)) #ttf

In [121]: len(inp1)

Out[121]: 27
```

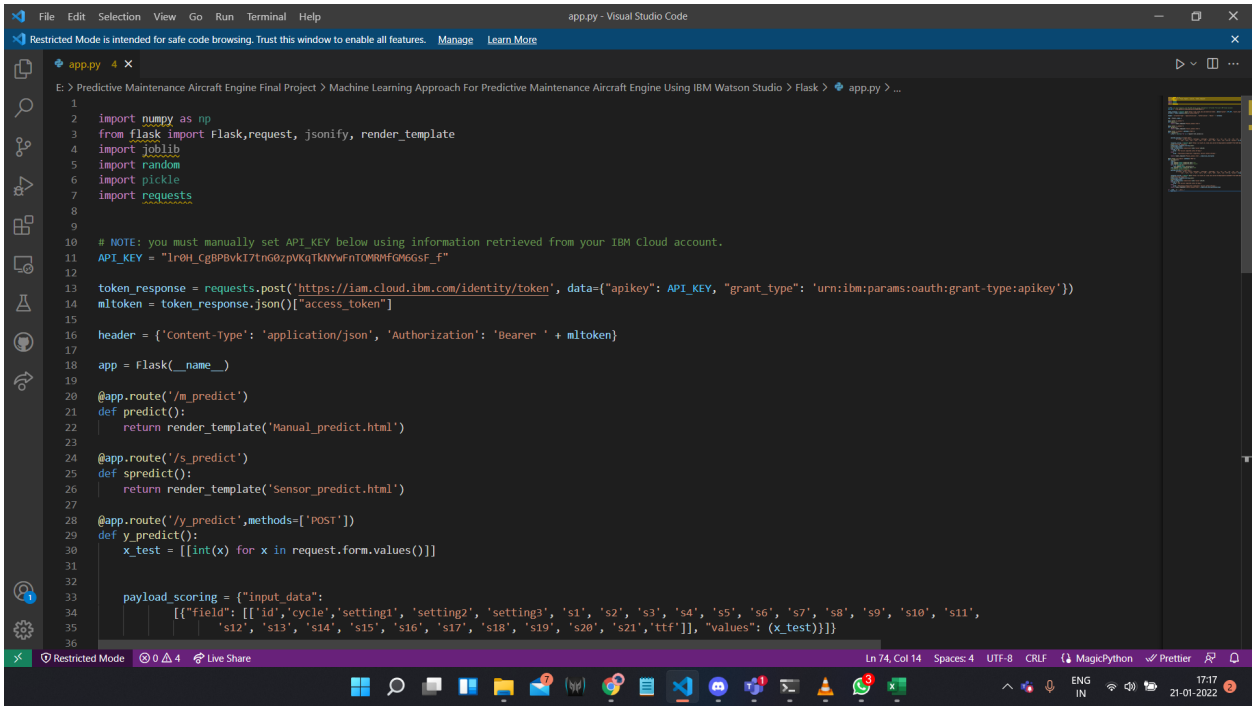

IBM Watson Studio
Projects / Smartinternz Project Deployment / IBM_Deployment
File Edit View Insert Cell Kernel Help
Not Trusted | Python 3.8
Run Format Code
In [120]: inp1=[]
inp1.append(random.randint(0,100)) #id
inp1.append(random.randint(0,365)) #cycle
for i in range(0,24):
inp1.append(random.uniform(0,1))
inp1.append(random.randint(0,365)) #ttf
In [121]: len(inp1)
Out[121]: 27
In [122]: model.predict([inp1])
Out[122]: array([0])
In [123]: from ibm_watson_machine_learning import APIClient
wml_credentials = {
"url":"https://us-south.ml.cloud.ibm.com",
"apikey":"lr0H_CgBPBvkI7tnG0zpVKqTkNYwFnTOMRMfGM6GsF_f"
}
client = APIClient(wml_credentials)
In [127]: def guid_from_space_name(client,space_name):
space = client.spaces.get_details()
return(next(item for item in space['resources'] if item['entity']['name'] == space_name)['metadata']['id'])
In [128]: space_uid = guid_from_space_name(client,'NewDeploymentSpace')
print("Space UID = " + space_uid)
Data
Files
Connections
Upload one file at a time. All file types accepted. 5 GB max file size.
Drag and drop files here or upload.
PM_test.txt
Insert to code
PM_train.txt
Insert to code
PM_truth.txt
Insert to code
engine_model.sav
Insert to code

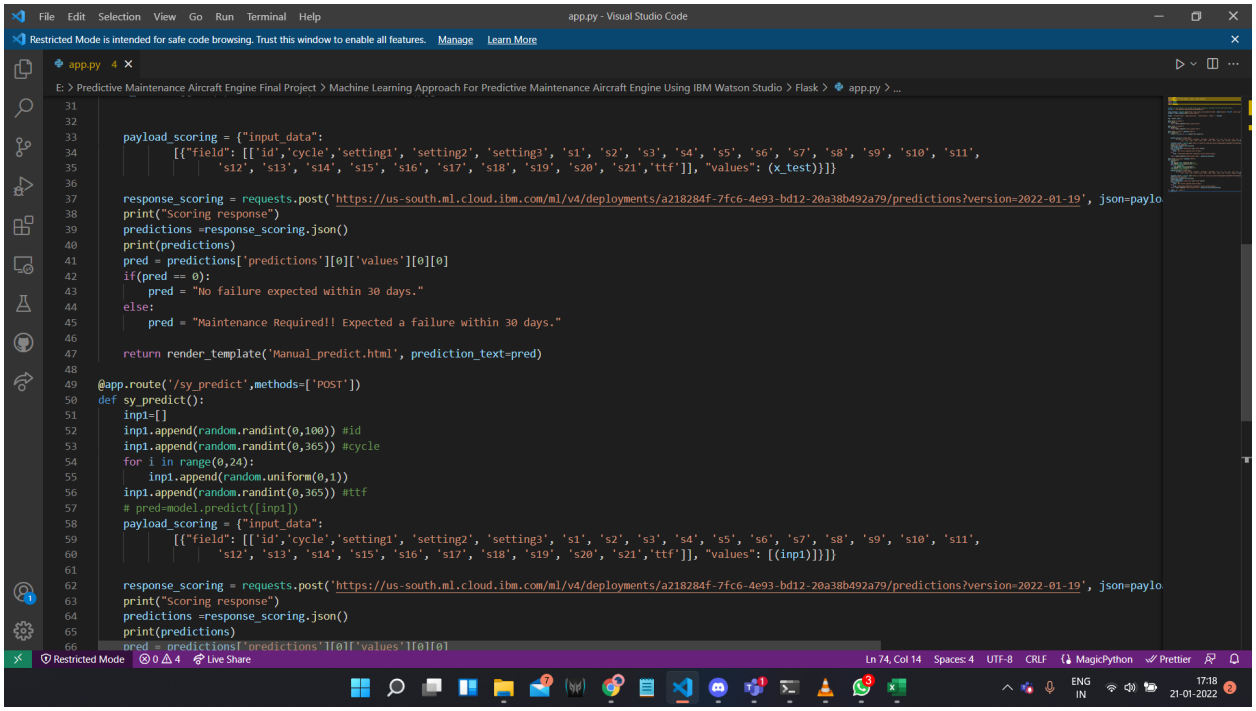

IBM Watson Studio
Projects / Smartinternz Project Deployment / IBM_Deployment
File Edit View Insert Cell Kernel Help
Not Trusted | Python 3.8
Run Format Code
Out[131]: 'ab9e1b80-f2ce-592c-a7d2-4f2344f77194'
In [132]: model_details = client.repository.store_model(model = model_log, meta_props = {
client.repository.ModelMetaNames.NAME:"engine_model",
client.repository.ModelMetaNames.TYPE:"scikit-learn_0.23",
client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid }
)
model_id = client.repository.get_model_uid(model_details)
In [133]: model_id
Out[133]: '894294f7-c27f-4b63-bfe8-50bfed52cb02'
In [134]: x_train[0]
Out[134]: array([1.00000000e+00, 1.00000000e+00, 4.59770115e-01, 1.66666667e-01,
0.00000000e+00, 0.00000000e+00, 1.83734940e-01, 4.06801831e-01,
3.09756921e-01, 0.00000000e+00, 1.00000000e+00, 7.26247987e-01,
2.42424242e-01, 1.09755003e-01, 0.00000000e+00, 3.69047619e-01,
6.33262260e-01, 2.05882353e-01, 1.99607803e-01, 3.63986149e-01,
0.00000000e+00, 3.33333333e-01, 0.00000000e+00, 0.00000000e+00,
7.13178295e-01, 7.24661696e-01, 1.91000000e+02])
In [137]: model_log.predict([[1.00000000e+00, 1.00000000e+00, 4.59770115e-01, 1.66666667e-01,
0.00000000e+00, 0.00000000e+00, 1.83734940e-01, 4.06801831e-01,
3.09756921e-01, 0.00000000e+00, 1.00000000e+00, 7.26247987e-01,
2.42424242e-01, 1.09755003e-01, 0.00000000e+00, 3.69047619e-01,
6.33262260e-01, 2.05882353e-01, 1.99607803e-01, 3.63986149e-01,
0.00000000e+00, 3.33333333e-01, 0.00000000e+00, 0.00000000e+00,
Data
Files
Connections
Upload one file at a time. All file types accepted. 5 GB max file size.
Drag and drop files here or upload.
PM_test.txt
Insert to code
PM_train.txt
Insert to code
PM_truth.txt
Insert to code
engine_model.sav
Insert to code

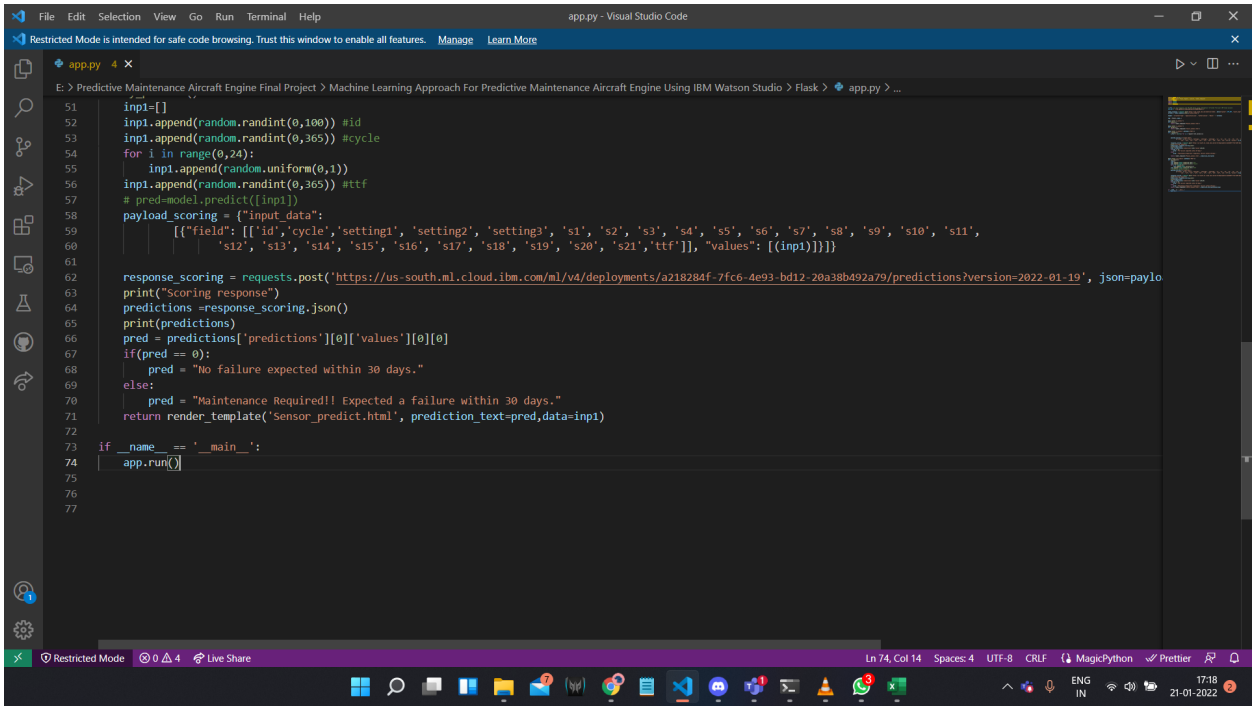




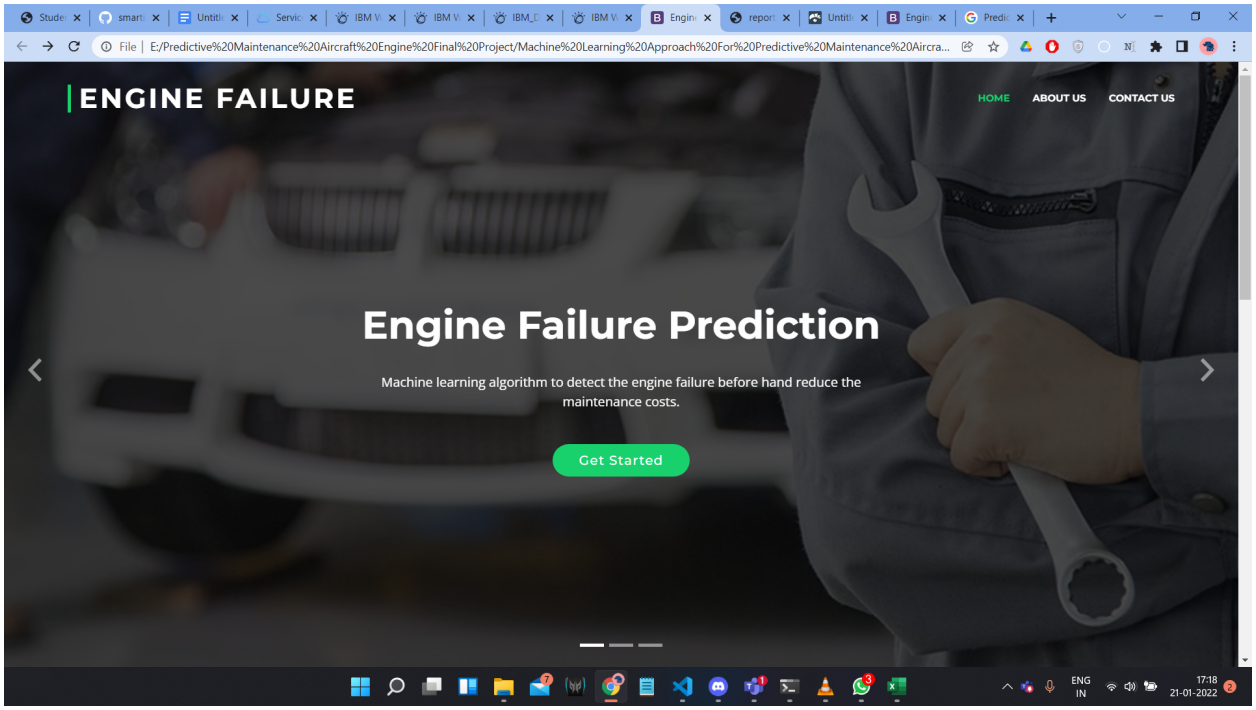
Flask python file

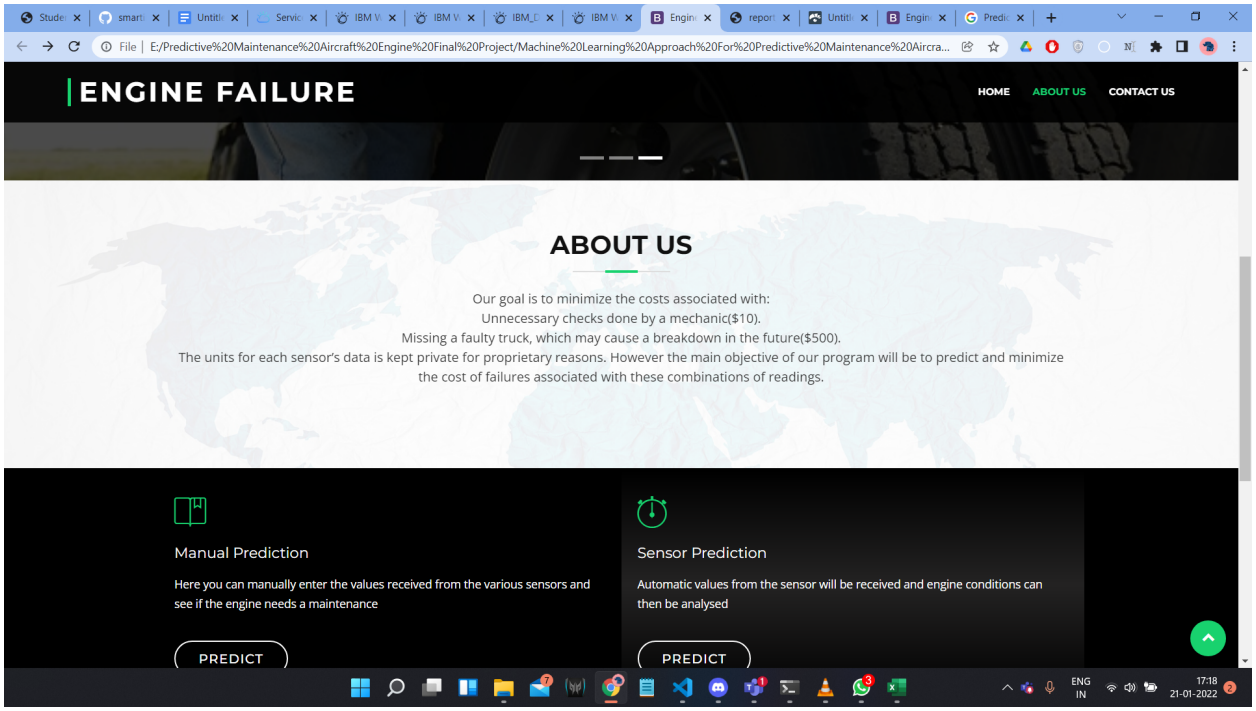

File Edit Selection View Go Run Terminal Help
app.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py 4
E: > Predictive Maintenance Aircraft Engine Final Project > Machine Learning Approach For Predictive Maintenance Aircraft Engine Using IBM Watson Studio > Flask > app.py > ...
import numpy as np
from flask import Flask,request, jsonify, render_template
import joblib
import random
import pickle
import requests
# NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account.
API_KEY = "lr0H_CgBPBvkI7tnG0zpVKqTkNYwFnTOMRMfGM6GsF_f"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey": API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
mltoken = token_response.json()["access_token"]
header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}
app = Flask(__name__)
@app.route('/m_predict')
def predict():
return render_template('Manual_predict.html')
@app.route('/s_predict')
def spredict():
return render_template('Sensor_predict.html')
@app.route('/y_predict',methods=['POST'])
def y_predict():
x_test = [[int(x) for x in request.form.values()]]
payload_scoring = {"input_data":
[{"field": [['id','cycle','setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21','ttf']], "values": (x_test)}]}
Restricted Mode 0 4 Live Share
Ln 74, Col 14 Spaces: 4 UTF-8 CRLF MagicPython Prettier
ENG IN
17:17
21-01-2022

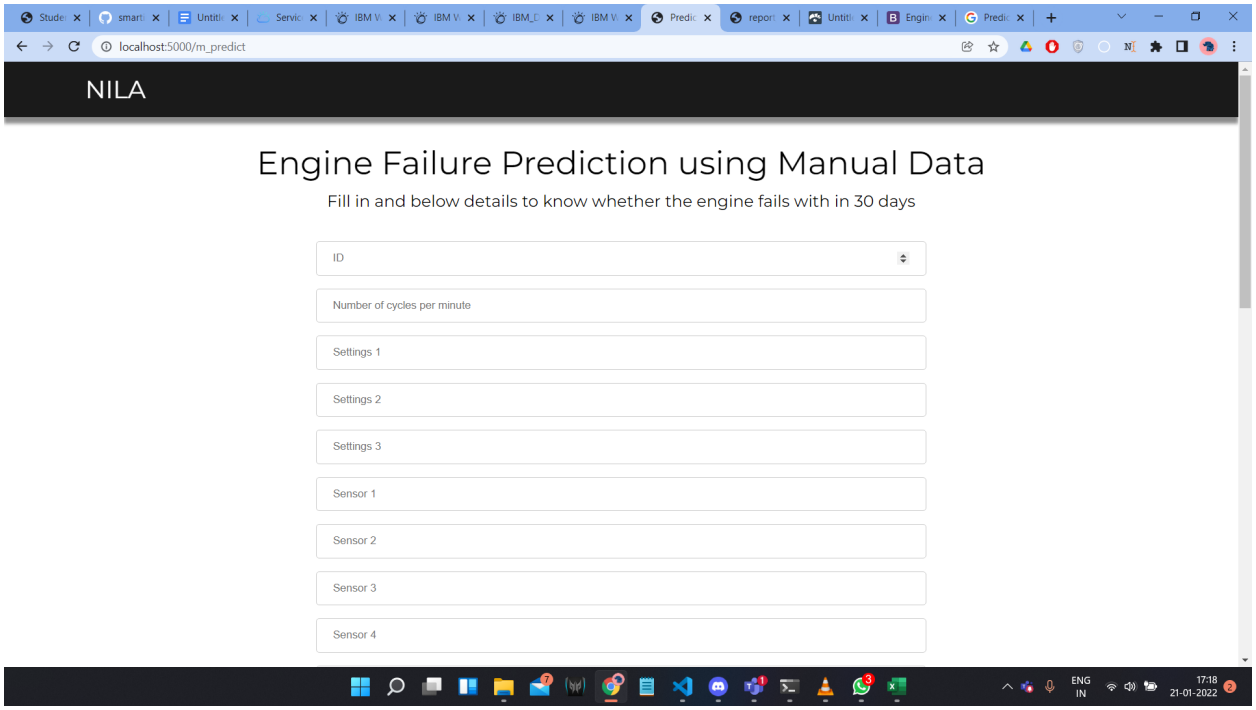

File Edit Selection View Go Run Terminal Help
app.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py 4
E: > Predictive Maintenance Aircraft Engine Final Project > Machine Learning Approach For Predictive Maintenance Aircraft Engine Using IBM Watson Studio > Flask > app.py > ...
payload_scoring = {"input_data":
[{"field": [['id','cycle','setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21','ttf']], "values": (x_test)}]}
response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/a218284f-7fc6-4e93-bd12-20a38b492a79/predictions?version=2022-01-19', json=paylo
print("Scoring response")
predictions =response_scoring.json()
print(predictions)
pred = predictions['predictions'][0]['values'][0][0]
if(pred == 0):
pred = "No failure expected within 30 days."
else:
pred = "Maintenance Required!! Expected a failure within 30 days."
return render_template('Manual_predict.html', prediction_text=pred)
@app.route('/sy_predict',methods=['POST'])
def sy_predict():
inp1=[]
inp1.append(random.randint(0,100)) #id
inp1.append(random.randint(0,365)) #cycle
for i in range(0,24):
inp1.append(random.uniform(0,1))
inp1.append(random.randint(0,365)) #ttf
# pred=model.predict([inp1])
payload_scoring = {"input_data":
[{"field": [['id','cycle','setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21','ttf']], "values": [(inp1)]}]}
response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/a218284f-7fc6-4e93-bd12-20a38b492a79/predictions?version=2022-01-19', json=paylo
print("Scoring response")
predictions =response_scoring.json()
print(predictions)
pred = predictions['predictions'][0]['values'][0][0]
Restricted Mode 0 4 Live Share
Ln 74, Col 14 Spaces: 4 UTF-8 CRLF MagicPython Prettier
ENG IN
17:18
21-01-2022

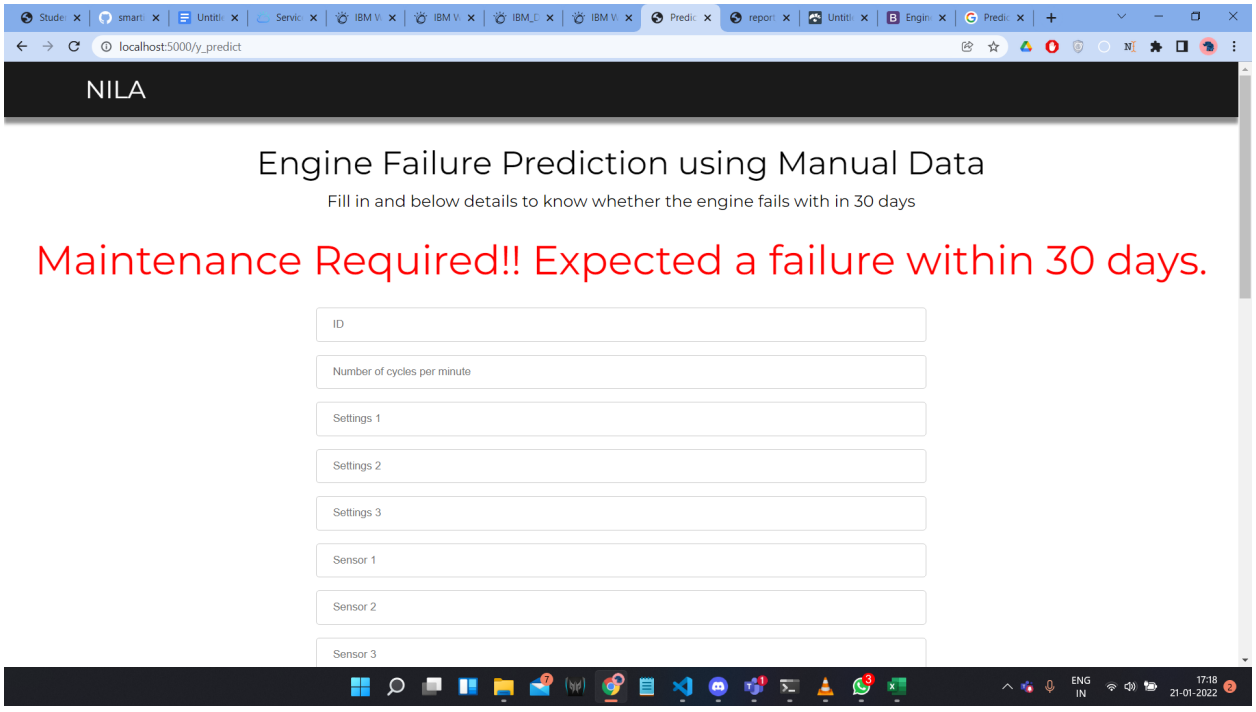

File Edit Selection View Go Run Terminal Help
app.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py 4
E: > Predictive Maintenance Aircraft Engine Final Project > Machine Learning Approach For Predictive Maintenance Aircraft Engine Using IBM Watson Studio > Flask > app.py > ...
inp1=[]
inp1.append(random.randint(0,100)) #id
inp1.append(random.randint(0,365)) #cycle
for i in range(0,24):
inp1.append(random.uniform(0,1))
inp1.append(random.randint(0,365)) #ttf
# pred=model.predict([inp1])
payload_scoring = {"input_data":
[{"field": [['id','cycle','setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21','ttf']], "values": [(inp1)]}]}
response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/a218284f-7fc6-4e93-bd12-20a38b492a79/predictions?version=2022-01-19', json=paylo
print("Scoring response")
predictions =response_scoring.json()
print(predictions)
pred = predictions['predictions'][0]['values'][0][0]
if(pred == 0):
pred = "No failure expected within 30 days."
else:
pred = "Maintenance Required!! Expected a failure within 30 days."
return render_template('Sensor_predict.html', prediction_text=pred,data=inp1)
if __name__ == '__main__':
app.run()
Restricted Mode
Live Share
Ln 74, Col 14
Spaces: 4
UTF-8
CRLF
MagicPython
Prettier
ENG
IN
17:18
21-01-2022

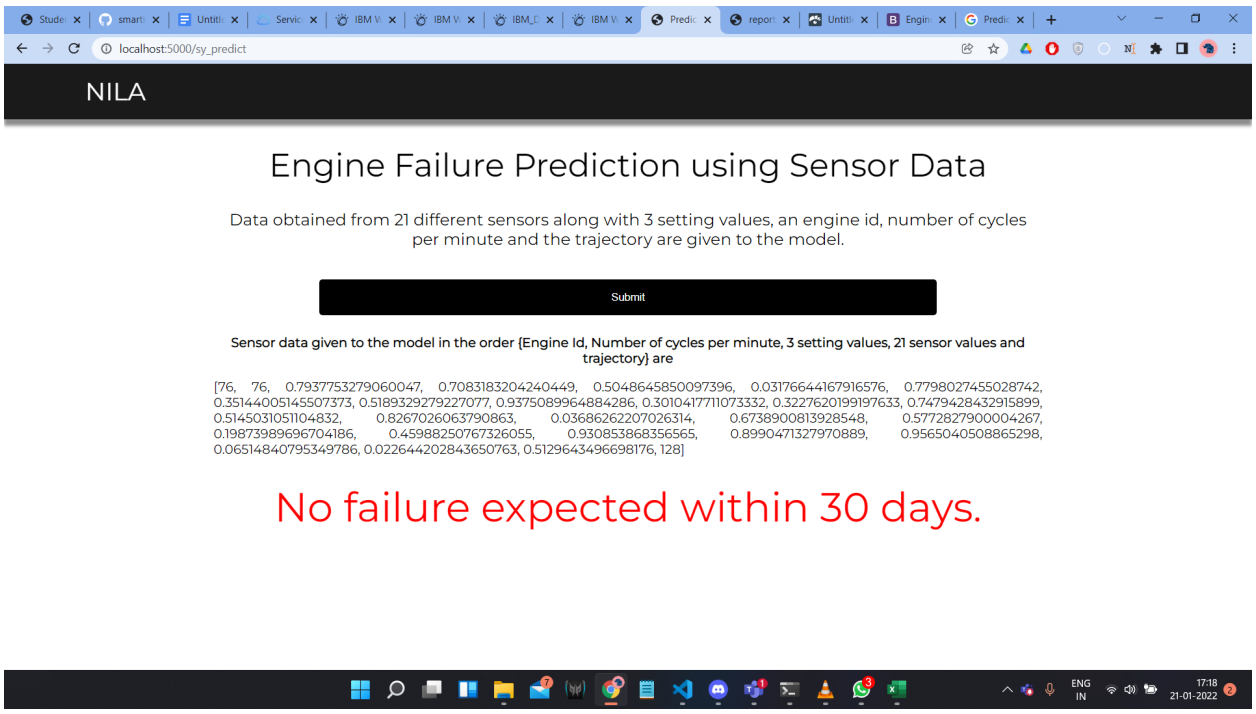
Output Page:


ENGINE FAILURE
HOME
ABOUT US
CONTACT US
Engine Failure Prediction
Machine learning algorithm to detect the engine failure before hand reduce the maintenance costs.
Get Started


ENGINE FAILURE
HOME
ABOUT US
CONTACT US
ABOUT US
Our goal is to minimize the costs associated with:
Unnecessary checks done by a mechanic($10).
Missing a faulty truck, which may cause a breakdown in the future($500).
The units for each sensor's data is kept private for proprietary reasons. However the main objective of our program will be to predict and minimize the cost of failures associated with these combinations of readings.
Manual Prediction
Here you can manually enter the values received from the various sensors and see if the engine needs a maintenance
PREDICT
Sensor Prediction
Automatic values from the sensor will be received and engine conditions can then be analysed
PREDICT


localhost:5000/m_predict
NILA
Engine Failure Prediction using Manual Data
Fill in and below details to know whether the engine fails with in 30 days
ID
Number of cycles per minute
Settings 1
Settings 2
Settings 3
Sensor 1
Sensor 2
Sensor 3
Sensor 4
ENG
IN
17:18
21-01-2022


localhost:5000/y_predict
NILA
Engine Failure Prediction using Manual Data
Fill in and below details to know whether the engine fails with in 30 days
Maintenance Required!! Expected a failure within 30 days.
ID
Number of cycles per minute
Settings 1
Settings 2
Settings 3
Sensor 1
Sensor 2
Sensor 3


localhost:5000/sy_predict
NILA
Engine Failure Prediction using Sensor Data
Data obtained from 21 different sensors along with 3 setting values, an engine id, number of cycles per minute and the trajectory are given to the model.
Submit
Sensor data given to the model in the order {Engine Id, Number of cycles per minute, 3 setting values, 21 sensor values and trajectory} are
[76, 76, 0.7937753279060047, 0.7083183204240449, 0.5048645850097396, 0.03176644167916576, 0.7798027455028742, 0.35144005145507373, 0.5189329279227077, 0.9375089964884286, 0.30104177111073332, 0.3227620199197633, 0.7479428432915899, 0.5145031051104832, 0.8267026063790863, 0.03686262207026314, 0.6738900813928548, 0.5772827900004267, 0.19873989696704186, 0.45988250767326055, 0.930853868356565, 0.8990471327970889, 0.9565040508865298, 0.06514840795349786, 0.022644202843650763, 0.5129643496698176, 128]
No failure expected within 30 days.