Ref No:	CABS/06/61/TRI/011
Type:	
Date:	24/05/2021
Copy No:	
No. of Pages:	54

Title: Predictive and Prescriptive Analytics Using LSTM techniques

College/University: Vellore Institute of Technology

	Name and Affiliation	Signature with Date
Student:	Mahalti Mohammed Sohail	M.Md. Sohail ,24/05/21
Guide:	Dr. Rajesh R	
Group Director:		
Head HR:		

Period of Project/ Internship: 6 months

Abstract: In the wake of extensive large-scale industries and mass production of products, with services becoming more complicated with inclusion of machinery, constant deterioration of working machinery is a major threat to continuous operation of these industries.

Predictive maintenance is the strategy of diagnosing potential equipment malfunctions in real time to prevent failures. It uses smart sensors such as machine vision to gather data from equipment, vehicles or other assets, automating the task of monitoring equipment. So far predictive maintenance is introduced for milling machines, heat exchangers and the health of robot. In this project, an attempt is made to introduce the deep learning based predictive maintenance to predict the Residual Useful Life (RUL) of turbofan degradation. Here, C-MAPSS data sets from Prognostics Centre of Excellence (PCoE) at NASA are used to train and test the model. After the prediction, the prescription for the Aircraft is calculated (i.e) Estimating which part of the engine is to be serviced first hand to increase the RUL of the aircraft engine drastically. Performance analysis are made between Long Short-Term Memory (LSTM), Bidirectional LSTM and Gated Recurrent Units with LSTM(GRU-LSTM) algorithms and GRU-LSTM out performs.

Predictive and Prescriptive Analytics Using LSTM techniques

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

In

Electronics and Communication Engineering

By

MAHALTI MOHAMMED SOHAIL

17BEC0187

Under the guidance of

Dr. VALARMATHI J

SENSE

VIT, Vellore.

April,2021

DECLARATION

I hereby declare that the thesis entitled "Predictive and Prescriptive Analytics Using LSTM techniques" submitted by me, for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering to VIT is a record of bonafide work carried out by me under the internal supervision of Dr. Valarmathi J and external supervision of Dr.R.Rajesh

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date:

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “Predictive and Prescriptive Analytics Using LSTM techniques” submitted by Mahalti Mohammed Sohail ,17BEC0187, SENSE, VIT, for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2020 to 30.04.2020, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfils the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: **Signature of the Guide**

Internal Examiner **External Examiner**

Head of the Department

Electronics and Communication Engineering

ACKNOWLEDGEMENTS

This report on “Predictive and Prescriptive Analytics Using LSTM techniques” is a result of a collective effort of a lot of precious people who have directly or indirectly supported me over the years.

First of all, I would like to express my sincere gratitude to my guide, Dr. Valarmathi J, Professor, SENSE, for guiding, motivating and helping me overcome any obstacle that came along the way. Her guidance and knowledge added with the exquisite help from DRDO scientists Dr.R.Rajesh and Shri.Dhipu T.M who kept me motivated throughout this project. Adding to this, their constant guidance made this research topic all the more enthralling and invigorating.

I would like to thank the almighty god for blessing us all with this opportunity to learn and give something back to the society. I am extremely grateful and would like to pay sincere and deepest gratitude to our Honorable Chancellor, Dr. G. Viswanathan for providing all the students of this University all the required facilities and literally a place to learn and grow beyond boundaries. We are extremely privileged to be here under his guidance. I owe my profound respect to the Dean, School of Electronics Engineering, Dr.Harish Kittur Mallikarjun for his continuous support and enlightenment. My sincere thanks also go to our Head of Department for believing in me and providing us with great opportunities to gain maximum knowledge and expand our horizons.

Last but not the least, I would also like to thank my family for being a constant pillar of support spiritually and providing me with everything I needed and my friends for keeping me going strong whenever I needed.

Mahalti Mohammed Sohail

Executive Summary

In the wake of extensive large-scale industries and mass production of products, with services becoming more complicated with inclusion of machinery, constant deterioration of working machinery is a major threat to continuous operation of these industries.

Predictive maintenance is the strategy of diagnosing potential equipment malfunctions in real time to prevent failures. It uses smart sensors such as machine vision to gather data from equipment, vehicles or other assets, automating the task of monitoring equipment. So far predictive maintenance is introduced for milling machines, heat exchangers and the health of robot. In this project, an attempt is made to introduce the deep learning based predictive maintenance to predict the Residual Useful Life (RUL) of turbofan degradation. Here, C-MAPSS data sets from Prognostics Centre of Excellence (PCoE) at NASA are used to train and test the model. After the prediction, the prescription for the Aircraft is calculated (i.e) Estimating which part of the engine is to be serviced first hand to increase the RUL of the aircraft engine drastically. Performance analysis are made between Long Short-Term Memory (LSTM), Bidirectional LSTM and Gated Recurrent Units with LSTM(GRU-LSTM) algorithms and GRU-LSTM out performs.

CONTENTS Page no.

Acknowledgement v

Executive summary vi

Table of contents vii

List of figures viii

List of tables ix

Abbreviations x

1 Introduction 1
2 Project description and goals 4
3 Technical specification 13
4 Design approach and details 14
5 Constraints, alternatives and trade-offs 19
6 Schedule, tasks and milestones 20
7 Project demonstration 21
8 Result & discussion 33
9 Summary 42
10 References 43

LIST OF FIGURES

Fig.1.1 (a) Reactive maintenance (b) Preventive maintenance (c) Predictive maintenance 2

Fig.2.1 (a): LSTM Cell structure 4

Fig.2.2 (a): Bidirectional LSTM models 6

Fig.2.3(a) GRU cell unit 7

Fig.2.6(a) Flowchart for prediction of RUL 9

Fig.2.6(b) Figure of Prognostics after prediction 10

Fig.2.7 Structure of the engine 10

Fig.8.1(a) R^2 vs epochs 34

Fig.8.1(b) loss vs epochs 35

Fig.8.1(c) MAE vs epoch 35

Fig.8.2(a) R^2 vs epochs 37

Fig.8.2(b) loss vs epochs 37

Fig.8.2(c) MAE vs epoch 38

Fig.8.3(a) R^2 vs epochs 40

Fig.8.3(b) loss vs epochs 40

Fig.8.3(c) MAE vs epoch 41

LIST OF TABLES

Table.2.7 Variables in the engine 12

Table.4.1(a) Epochs of each algorithm 18

Table.6.1 Task, schedule and milestone 21

Table.8.1(a) Parameters of the model 34

Table.8.1(b) confidence values with sensor and setting index 36

Table.8.2(a) Parameters of the model 37

Table.8.2(b) confidence values with sensor and setting index 39

Table.8.3(a) Parameters of the model 40

Table.8.3(b) confidence values with sensor and setting index 42

ABBREVATIONS:

RUL	Residual Useful Life
LSTM	Long Short-term memory
BiLSTM	Bidirectional Long Short-term memory
GRU	Gated Recurrent Unit
CMAPPS	Commercial Modular Aero Propulsion System Simulation
PCoE	Prognostics Centre of Excellence
LPC	Low power Compressor
HPT	High power Turbine
N1	Nozzle 1
N2	Nozzle 2
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing unit
MAE	Mean Absolute Error

1.INTRODUCTION

1.1 OBJECTIVE

The objective of this project is to predict Residual useful Life (RUL) of a turbofan engine using the sensor data from the past. For this task LSTM systems were taken into consideration and further analysis was done to compare them to find best suited for predicting the RUL. Following the analysis for prediction, steps to be taken to increase to increase the RUL of the engine are to be considered. Hence the sensors are given a particular priority for increasing the RUL. The steps to be taken to maximize usability for more longer time is called prescriptive analysis. For this work CMAPSS Dataset from NASA’s PCoE is taken create and predict the RUL of each kind of engine by training LSTM models for the task of prediction. After prediction, to find an order in which the sensor parts are to be serviced to maximise longevity and the whole prediction is performed in different conditions to maximise accuracy and result.

1.2 MOTIVATION

Predictive maintenance is the best approach to managing costs and losses for an industry or a business as unexpected halt on services results in losses where as if, before the stoppage proper measure were taken to increase longevity by predicting the estimate time is the way to go. If scheduled servicing can also be considered for such approach but it might result in unwanted costs when the machine might not need it in the first place. After such predictions the company is put into a dilemma in what and how to service the machinery and what has the most priority what has the least when considered individually or together in pairs. Then, prescriptive analysis gives the direction for what steps to be taken to directly affect the overall health of the machine.

1.3 BACKGROUND:

Predictive maintenance dates back to a few decades when condition monitored maintenance became prevalent and even after following such practices resulted in breakage of machines altogether because it was either too late or was unnecessary. With the growing factors within machinery to the losses faced due to the stoppage of services which play a vital role in proper functioning of industries and also improper maintenance results in health hazards and bad working conditions. Hence, to tackle such problems maintenance techniques should be give much more importance. The following are the conventional approaches to Maintenance.

Reactive maintenance: The tool or set of parts of a machine is fixed when its broken or deteriorated. This causes problems like stopping the services to a halt till it is fixed which may cost the companies in millions and also a health hazard if there is no proper monitoring of equipment.

Preventive Maintenance: The parts of machine undergo a scheduled maintenance to ensure proper functioning all the time. Here, the company has to spend extra costs in maintenance even if it's not needed for the machine at the particular time

Predictive maintenance: Here the working of the machine is constantly monitored and the potential time for its maintenance is predicted beforehand to save both time as well as resources for proper functioning of the whole system. As seen in Fig (1.1) predictive maintenance is currently the best way to approach the problem of maintenance of machines.

The following work shows the proper procedure pertaining to the task of predictive maintenance and the procedure followed afterwards of prediction.

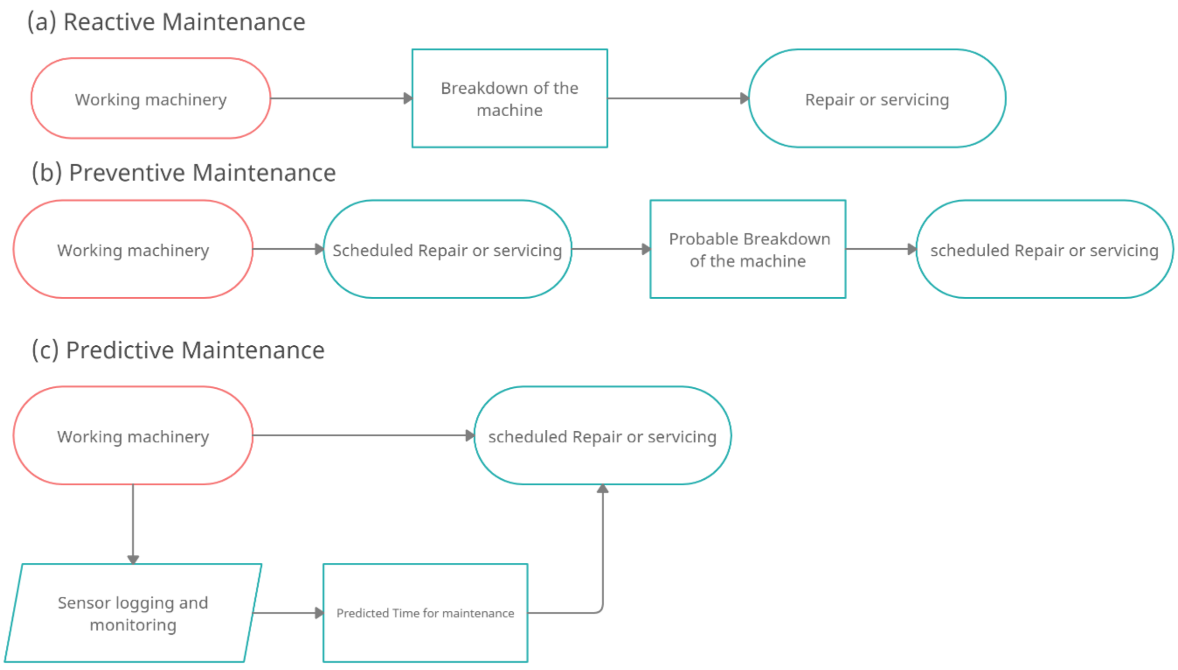

(a) Reactive Maintenance
Working machinery
Breakdown of the machine
Repair or servicing
(b) Preventive Maintenance
Working machinery
Scheduled Repair or servicing
Probable Breakdown of the machine
scheduled Repair or servicing
(c) Predictive Maintenance
Working machinery
scheduled Repair or servicing
Sensor logging and monitoring
Predicted Time for maintenance

Fig.1.1: (a) Reactive maintenance (b) Preventive maintenance (c) Predictive maintenance

For the purpose of prediction, LSTM (long short-term memory) models were opted as they avoid the problem of vanishing gradients as sensor data is mainly consistent of sequences. The following thesis consists of types of LSTM models and a comparison between them along with the prescriptions for each model after prediction for servicing the Engine. For prediction NASA's CMAPSS dataset was used which consists of training and testing logs of fleet of engines operated at different conditions. The prescriptions rank the priority order with respect to mutual information and correlation amongst the Residual useful life (RUL) of the engine.

In this report, the algorithms were consistently improved upon the prediction and analysed the data further in terms of prediction the following are the steps taken:

1) Learning and implementing types of LSTM models and comparing them
2) Implementing the models with respect to different conditions
3) Ranking the parts to be serviced in a particular order to increase the RUL of the Engine

2 PROJECT DESCRIPTION AND GOALS

2.1 Introduction to LSTM

The following is an artificial neural network with recurrent neural network architecture to minimize the vanishing gradient problem and exploding gradient. LSTMs are generally used for time series predictions.

Vanishing gradient problem:

In the artificial neural networks, which use gradient descent for weight updating with multiple iterations of updating multiple weights the size of the gradient continues to decrease resulting in very slight change in weights even after many iterations.
The value of gradient continues to decrease as each weight of the neural network has the value of gradient range between 0 and 1 and after multiple iterations and multiplication of these gradients under the chain rule for updating weights results in such problem.

Exploding gradient problem:

In artificial neural networks, which use gradient descent vice versa can also happen if the value of the gradient doesn’t lie between 0 and 1 and is higher. After each iteration it continues to increase further making it difficult for the model to learn the data properly.

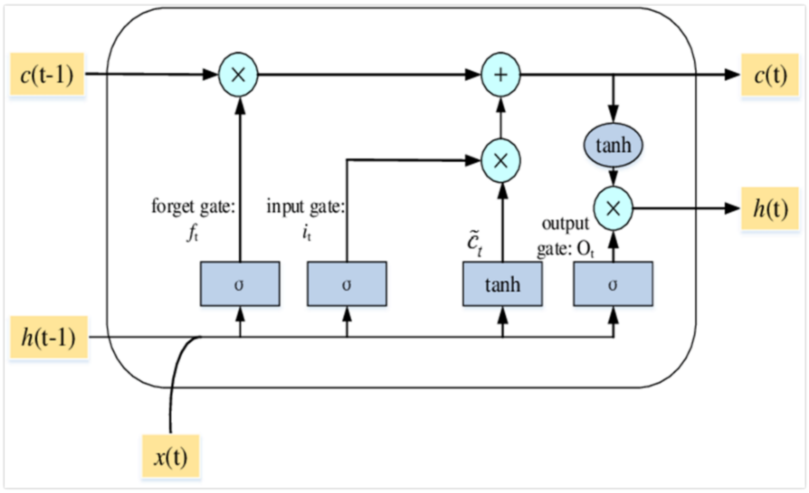


Fig.2.1 (a): LSTM Cell structure *(courtesy: wiki)*

Forget Gate:

The forget gate decides if the data is important enough to learned and remembered, the following is a sigmoid function which transforms the data in between 0 to 1. Hence if the value is closer to 1 the important the data and is the value closer to 0 the more unimportant the data and decides whether to reset the memory

Input Gate:

The following gate updates the cell state of the LSTM unit using the similar principle

Output Gate:

The following gate gives out the new weight after learning with respect to the previous cell state received

Formulae:

$$\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}$$

f_t: forget gate's activation vector

x_t: input vector to LSTM unit

i_t: input/update gate's activation vector

o_t: output gate's activation vector

h_t: hidden state vector, output vector

$\acute{c}_t$: cell input activation vector

c_t: cell state vector

W,U,b: weight and bias matrices

σ_g : Sigmoid function

σ_c : hyperbolic tan function

Here, Hadamard's operand is mentioned which means element wise multiplication. Sigmoid and hyperbolic functions are also used with reference to Fig (2.1)

2.2 Introduction to Bidirectional LSTM models:

Bidirectional LSTM models give the input twice for the model to train, once in the order which was given and in the second time the input is given in an inverted form to make it easier for the model to train

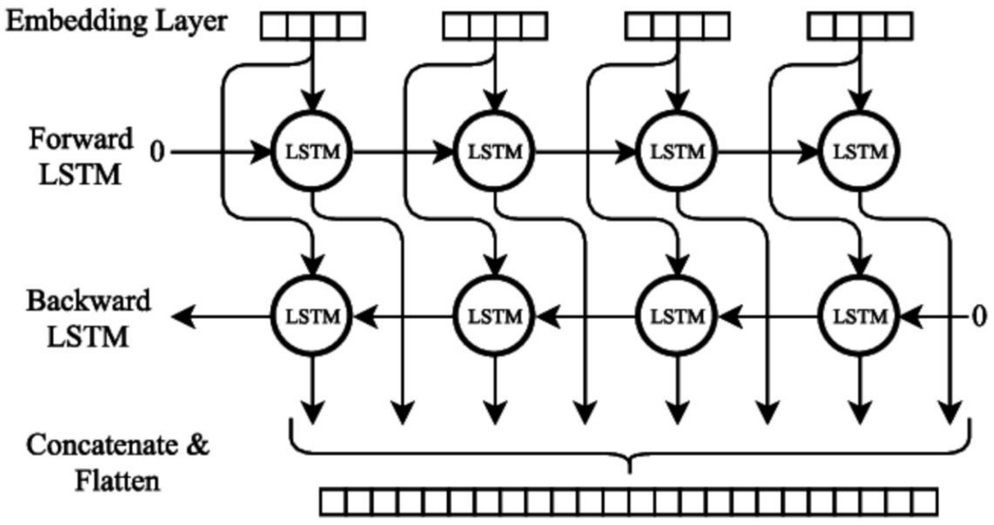


Fig.2.2 (a): Bidirectional LSTM models *(courtesy: https://paperswithcode.com/)*

Hence, from the figure we can see there are two LSTM layers each going in the opposite direction. Bidirectional LSTMs are used to increase accuracy of the LSTM predictions by helping the model train in the specified manner.

2.3 Gated Recurrent Units (GRU)s:

Gated Recurrent Units are a type of LSTM cells which simplify the LSTM cells with a forget gate by using few parameters as it lacks an output gate. GRU mainly consist of Update gate and Reset Gate

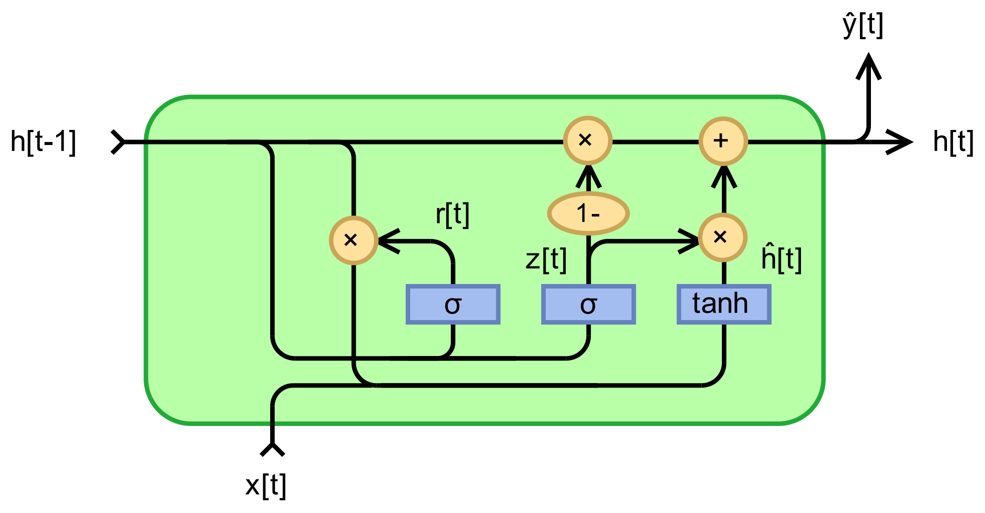


Fig.2.3(a) GRU cell unit *(courtesy: wiki)*

Reset Gate:

The input and hidden state are concatenated and computed under sigmoid function and the result value ranges between 0 and 1. The following value decides if the memory is to be reset or the following input received by the cell is important and has a high probability to occur.

Update Gate:

At the Update gate the hidden state of the GRU is update depending upon the input, previous hidden state and their resulting probability. The Tan Hyperbolic function helps the GRU to minimize Vanishing gradient problem by spreading the value in between -1 and 1 which is also seen in regular LSTM networks.

Formulae:

$$\begin{aligned} z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\ \hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \end{aligned}$$

Variables:

x_t: input vector

h_t : output vector

$\hat{h}_t$: candidate activation vector

z_t: update gate vector

r_t: reset gate vector

W, U and b: parameter matrices and vector

σ_g : Sigmoid function

φ_h : hyperbolic tan function

Here, Greek letter Sigma denotes Sigmoid function, Greek letter phi denotes hyperbolic function and the Hadamard operand means element wise product.

Performance metrics used:

2.4 Correlation Coefficient:

The following value denotes the relationship between a dependant and an independent variable and assists in giving a set of permutation of the sensors which are highly dependent on the Residual Useful of the Engine.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Here x and y are two variables

2.5 Mutual information score:

Mutual Information score gives an estimate of similarity between two variables and is used in the following project to assess the result obtained after prediction and obtain ranking of the sensors from the most influential to the least. The following ranges for 0 to 1 where, 1 signifies the values to be most similar and 0 to least similar.

2.6 Procedure flowchart

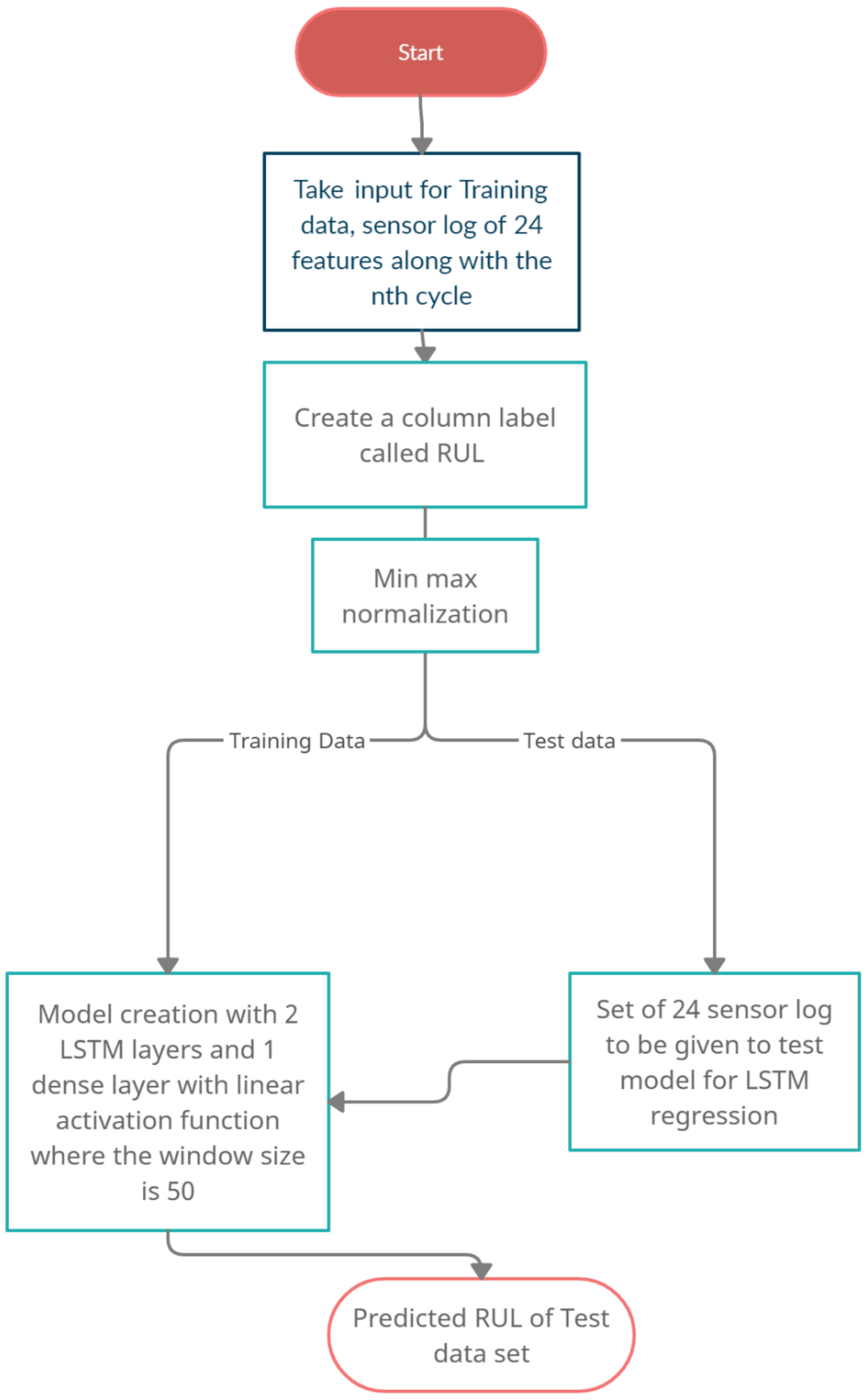

Start
Take input for Training data, sensor log of 24 features along with the nth cycle
Create a column label called RUL
Min max normalization
Training Data
Test data
Model creation with 2 LSTM layers and 1 dense layer with linear activation function where the window size is 50
Set of 24 sensor log to be given to test model for LSTM regression
Predicted RUL of Test data set

Fig.2.6(a) Flowchart for prediction of RUL

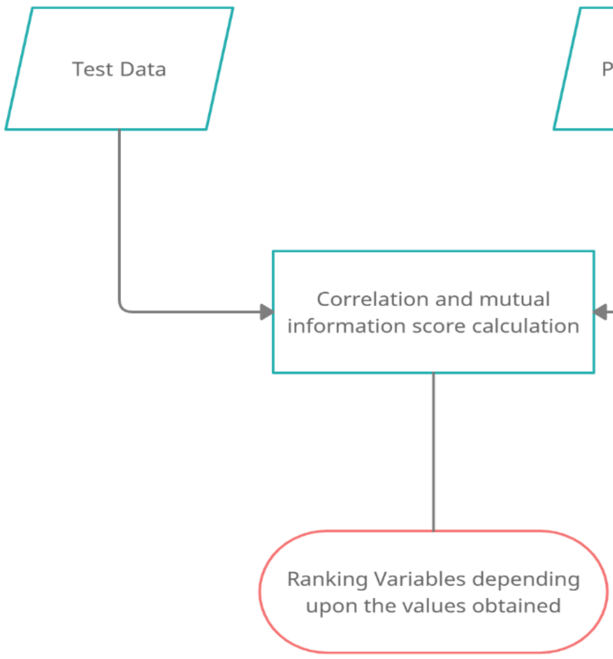

Test Data
Predicted data
Correlation and mutual information score calculation
Ranking Variables depending upon the values obtained

Fig.2.6(b) Figure of Prognostics after prediction

2.7 Dataset:

For the following project CMAPSS Dataset made by NASA's Prognostics Centre of Excellence was taken. The following was made by a simulation in MATLAB and contained Sensor logs of 21 sensors present on the engine. The dataset is segregated depending upon the conditions imposed on them.

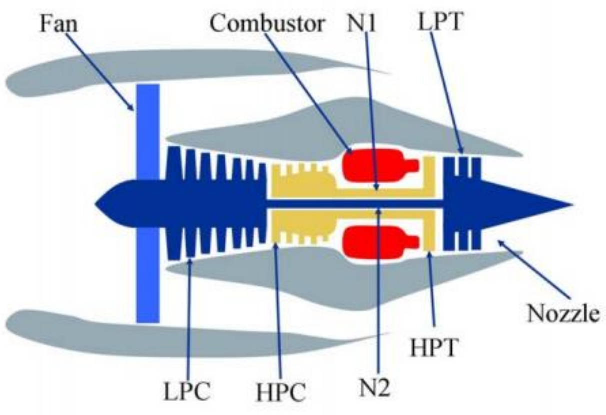


Fig.2.7 Structure of the engine

Symbol	Description	Units	Index
T2	Total temperature at fan inlet	°R	4
T24	Total temperature at LPC outlet	°R	5
T30	Total temperature at HPC outlet	°R	6
T50	Total temperature at LPT outlet	°R	7
P2	Pressure at fan inlet	psia	8
P15	Total pressure in bypass-duct	psia	9
P30	Total pressure at HPC outlet	psia	10
Nf	Physical fan speed	rpm	11
Nc	Physical core speed	rpm	12
Epr	Engine pressure ratio (P50/P2)	--	13
Ps30	Static pressure at HPC outlet	psia	14
Phi	Ratio of fuel flow to Ps30	pps/psi	15
NRf	Corrected fan speed	rpm	16
NRc	Corrected core speed	rpm	17
BPR	Bypass Ratio	--	18

farB	Burner fuel-air ratio	--	19
htBleed	Bleed Enthalpy	--	20
Nf_dmd	Demanded fan speed	rpm	21
PCNfR_dmd	Demanded corrected fan speed	rpm	22
W31	HPT coolant bleed	lbm/s	23
W32	LPT coolant bleed	lbm/s	24

Table.2.7 Variables in the engine

The table 2.7 consists of all the 21 sensors and their units

The Segregation of the Dataset is as follows:

Data Set: FD001
Train trajectories: 100
Test trajectories: 100
Conditions: ONE (Sea Level)
Fault Modes: ONE (HPC Degradation)

Data Set: FD002
Train trajectories: 260
Test trajectories: 259
Conditions: SIX
Fault Modes: ONE (HPC Degradation)

Data Set: FD003
Train trajectories: 100
Test trajectories: 100
Conditions: ONE (Sea Level)
Fault Modes: TWO (HPC Degradation, Fan Degradation)

Data Set: FD004

Train trajectories: 248

Test trajectories: 249

Conditions: SIX

Fault Modes: TWO (HPC Degradation, Fan Degradation)

Goals:

(i) Using the LSTM techniques mentioned creating linear regression models to find the relationship between the RUL of the type of engine to the 21 sensors along with operating settings mentioned in the dataset

(ii) Using the model predicting RUL of the Engine

(iii) Finding the priority order in which the sensors are to tended in order to increase the RUL of the Engine

(iv) Give the priority in the form of a score

3.TECHNICAL SPECIFICATIONS:

The following project was made by me on my laptop using Anaconda Navigator which creates python environment in which one can install libraries pertaining to their project easily and run them. Anaconda also supports CUDA which enables the model to use the GPU while training which makes it faster.

Computer specifications:

Processor: intel core i7 7700HQ

GPU: GTX 1060 6GB

RAM: 16 GB

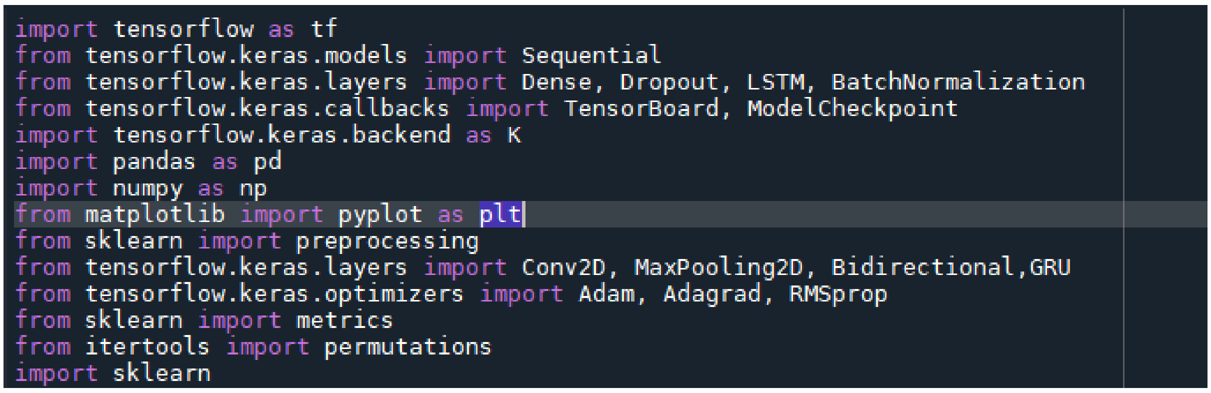
Anaconda allows absolutely anyone to develop deep learning applications using popular libraries such as PyTorch, TensorFlow, Keras, and OpenCV.The following project was made in Spyder script editor.

4 DESIGN APPROACH:

4.1 Creating the model:

Importing Libraries:

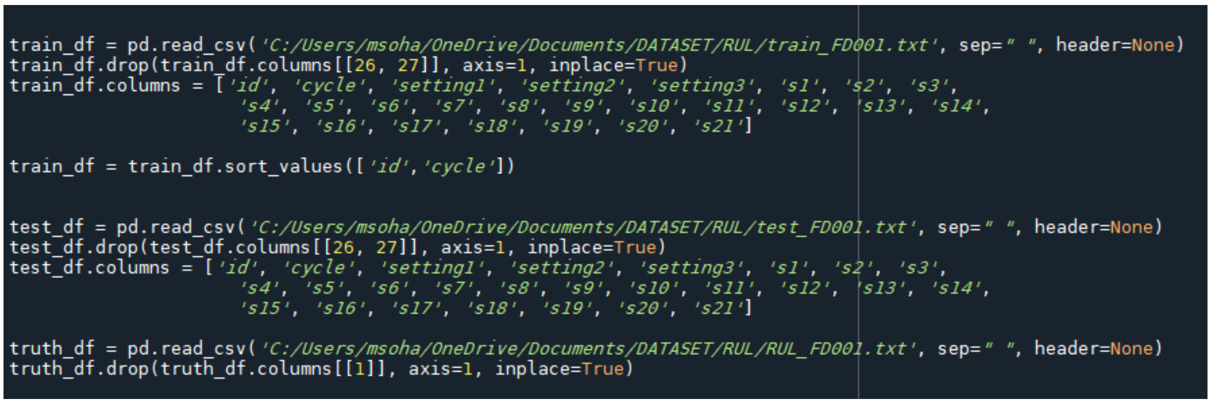
In Python script, First the following libraries are imported.Tensorflow library enables us to import layers pertaining to the model.Libraries like pandas , numpy and sklearn are used for datapreprocessing before giving the data for the model to train



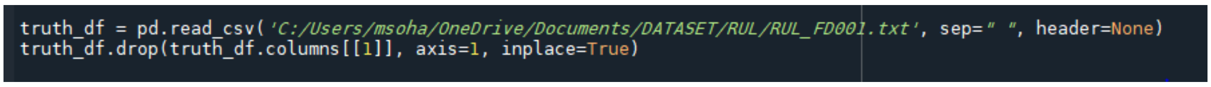
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
import tensorflow.keras.backend as K
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import preprocessing
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional,GRU
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from sklearn import metrics
from itertools import permutations
import sklearn
```

Creating Data frames:

From the dataset, training data, testing data and the truth files are used to create data frames by using the following command.
For the time being the columns are named as id, cycle, setting [from 1 to 3], sensor logs s1, s2, s3…. and so, on



```
train_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/train_FD001.txt', sep=" ", header=None)
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
train_df.columns = ['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']

train_df = train_df.sort_values(['id','cycle'])


test_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/test_FD001.txt', sep=" ", header=None)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
test_df.columns = ['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']

truth_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/RUL_FD001.txt', sep=" ", header=None)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)
```

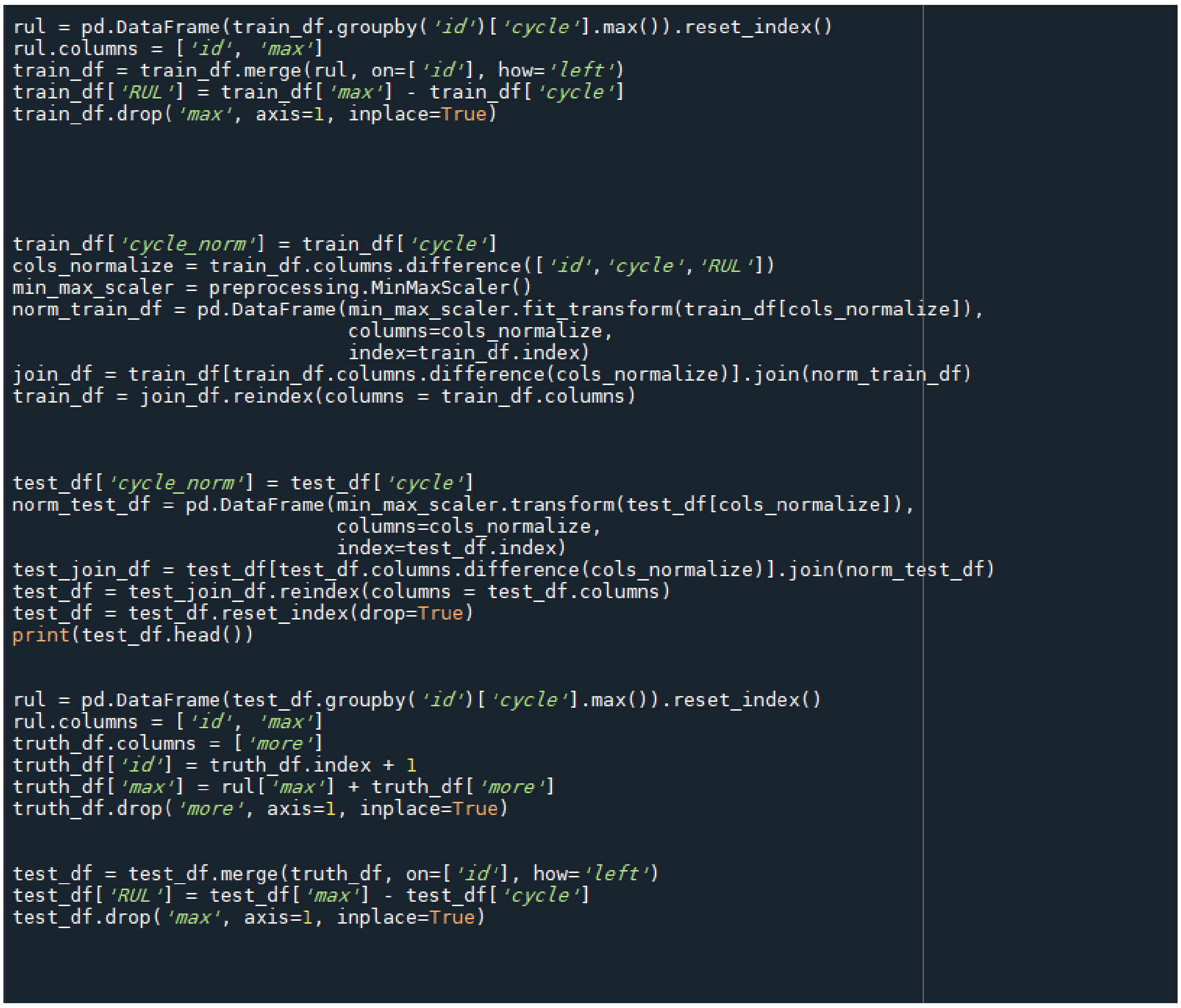


```
truth_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/RUL_FD001.txt', sep=" ", header=None)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)
```

Data pre-processing:

The following data from each data frame is sorted and normalised depending upon the engine id number. After sorting the following another column is added for training named RUL.
The column called RUL consists of maximum possible cycles observed at the particular trajectory.

The following column is added to both the data frames of training and test



```
rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)




train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)


test_df['cycle_norm'] = test_df['cycle']
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
print(test_df.head())


rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)


test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)
```

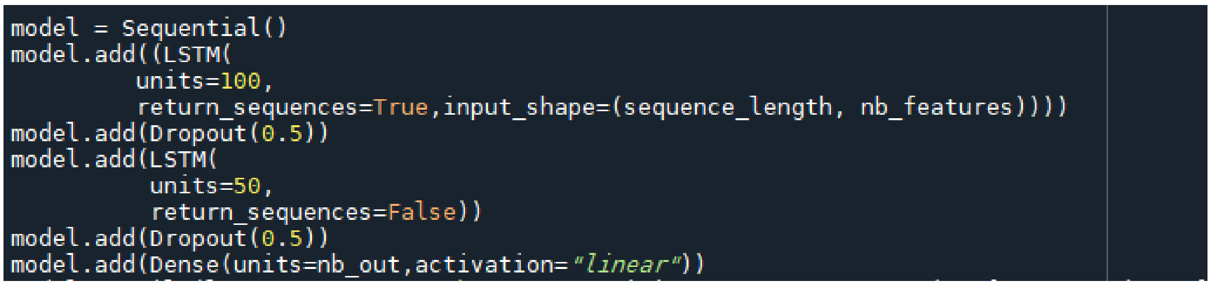
Here, ‘cycle’ signifies the nth cycle of the engine.

Coding the model:

The model consists of layers depending upon the name. Depending upon the same the parameters and the hyper parameters also vary. The following are the codes of model specific to each.

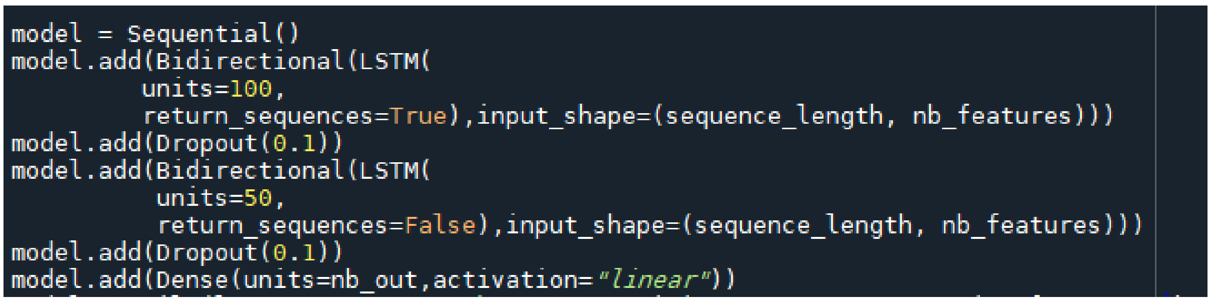
LSTM:

By multiple iterations and optimization, the hyper parameters are found to be as follows.



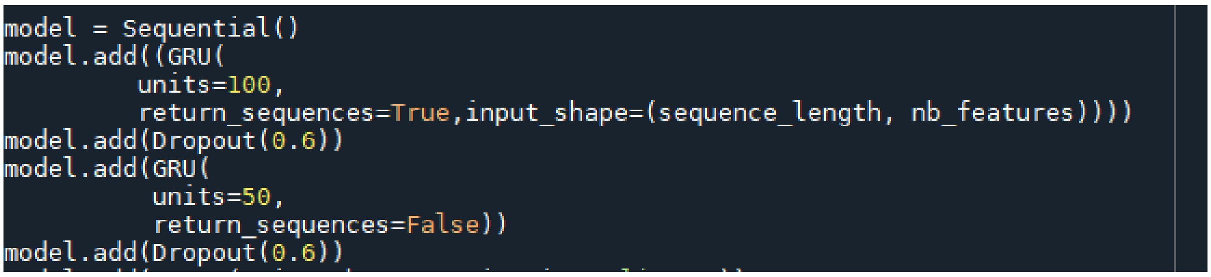
```
model = Sequential()
model.add((LSTM(
         units=100,
         return_sequences=True,input_shape=(sequence_length, nb_features))))
model.add(Dropout(0.5))
model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(units=nb_out,activation="linear"))
```

BiLSTM:



```
model = Sequential()
model.add(Bidirectional(LSTM(
         units=100,
         return_sequences=True),input_shape=(sequence_length, nb_features)))
model.add(Dropout(0.1))
model.add(Bidirectional(LSTM(
          units=50,
          return_sequences=False),input_shape=(sequence_length, nb_features)))
model.add(Dropout(0.1))
model.add(Dense(units=nb_out,activation="linear"))
```

GRU:



```
model = Sequential()
model.add((GRU(
         units=100,
         return_sequences=True,input_shape=(sequence_length, nb_features))))
model.add(Dropout(0.6))
model.add(GRU(
          units=50,
          return_sequences=False))
model.add(Dropout(0.6))
```

Dropout command specifies the rate at which the data should be dropped for making the learning easier for the model.

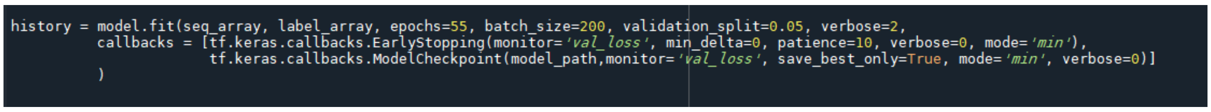
Compiling the model:

Depending upon the layers and the type of the model the epochs varies as some models may learn at lower iterations and increasing it to more iterations causes the problem of overfitting

	LSTM	BiLSTM	GRU
Epochs made to run	55	20	30

Table.4.1(a)

Command for compiling the model:



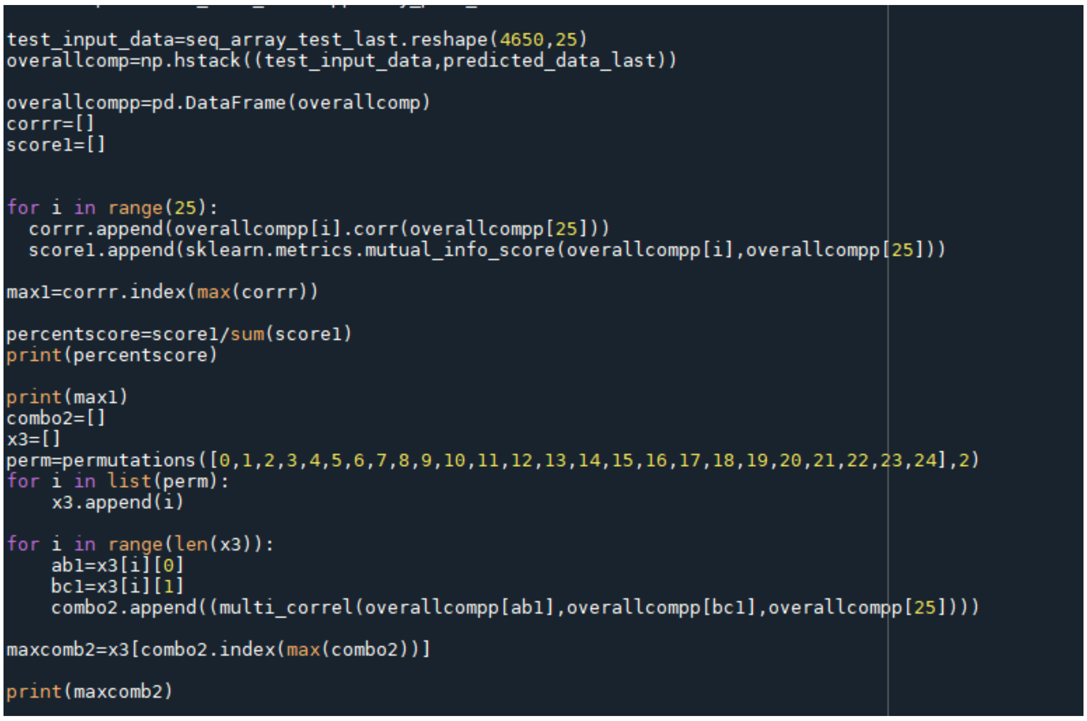
```
history = model.fit(seq_array, label_array, epochs=55, batch_size=200, validation_split=0.05, verbose=2,
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       tf.keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )
```

Here, the training data and the label array are specified along with the batch size and validation split. Validation split divided the training data automatically by the specified amount and the values like loss, mean absolute error are minimised after constant prediction during training.

After fitting and optimizing the model predictions are made using the test data. The data gathered after prediction is concatenated to find the relationship between the test data and the predicted RUL.

After getting the resultant data frame the values like coefficient of correlation and mutual information score are calculated to find the priority order in which the part of the engine is to serviced.

Code used:



```
test_input_data=seq_array_test_last.reshape(4650,25)
overallcomp=np.hstack((test_input_data,predicted_data_last))

overallcompp=pd.DataFrame(overallcomp)
corrr=[]
score1=[]


for i in range(25):
  corrr.append(overallcompp[i].corr(overallcompp[25]))
  score1.append(sklearn.metrics.mutual_info_score(overallcompp[i],overallcompp[25]))

max1=corrr.index(max(corrr))

percentscore=score1/sum(score1)
print(percentscore)

print(max1)
combo2=[]
x3=[]
perm=permutations([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],2)
for i in list(perm):
    x3.append(i)

for i in range(len(x3)):
    ab1=x3[i][0]
    bc1=x3[i][1]
    combo2.append((multi_correl(overallcompp[ab1],overallcompp[bc1],overallcompp[25])))

maxcomb2=x3[combo2.index(max(combo2))]

print(maxcomb2)
```

5. ALTERNATIVES, TRADE OFFS AND CONSTRAINTS

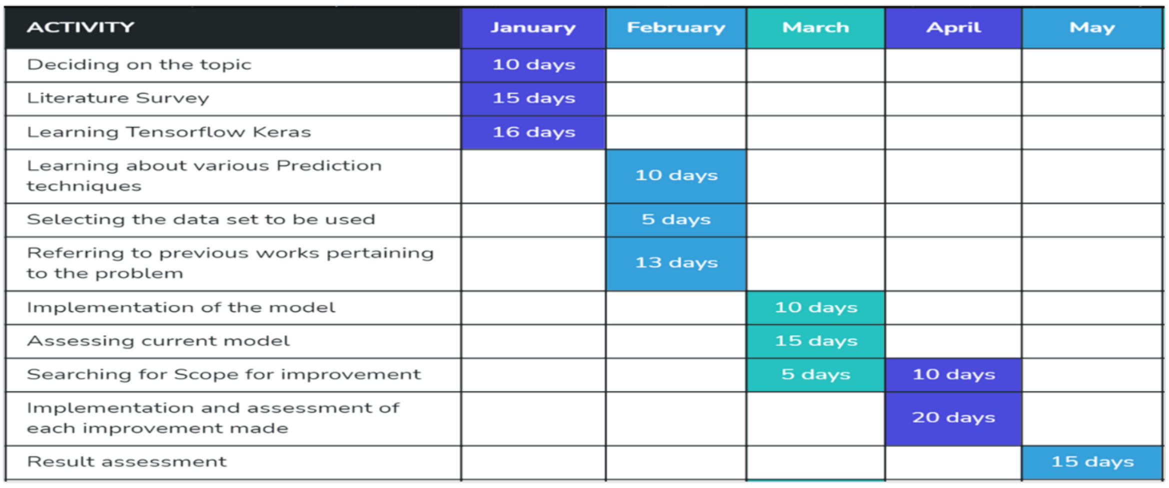
Constraints:

With respect to regression error is bound to occur with predictions and also there would be problem if the correlation coefficient is very close to 1, which signifies overfitting and being close to 0 causes under fitting. Even after optimizing the model to cater to all kinds of predictions the error is distributed among all the predictions hence, which causes them to deviate from the expected value.

Alternatives:

For the following project it has many alternatives like scheduled maintenance, reactive maintenance and within predictive maintenance, Deep belief Networks , SVMs exist as the alternatives depending upon the task. In the view of more Accuracy and less complexity LSTM systems were chosen for the project.

6.1 SCHEDULE, TASK AND MILESTONE

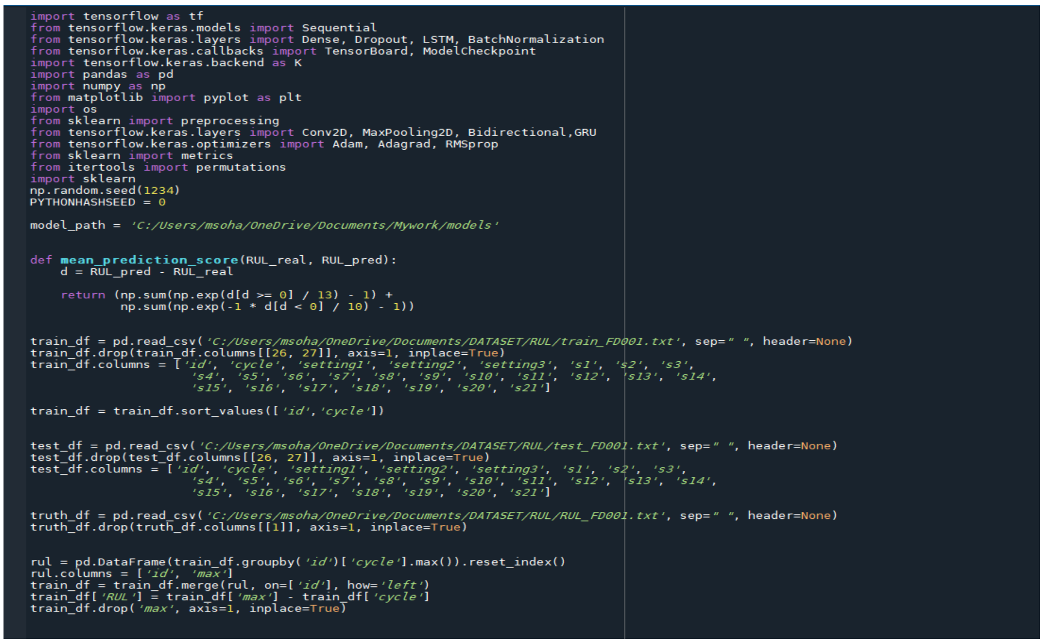


ACTIVITY	January	February	March	April	May
Deciding on the topic	10 days				
Literature Survey	15 days				
Learning Tensorflow Keras	16 days				
Learning about various Prediction techniques		10 days			
Selecting the data set to be used		5 days			
Referring to previous works pertaining to the problem		13 days			
Implementation of the model			10 days		
Assessing current model			15 days		
Searching for Scope for improvement			5 days	10 days	
Implementation and assessment of each improvement made				20 days	
Result assessment					15 days

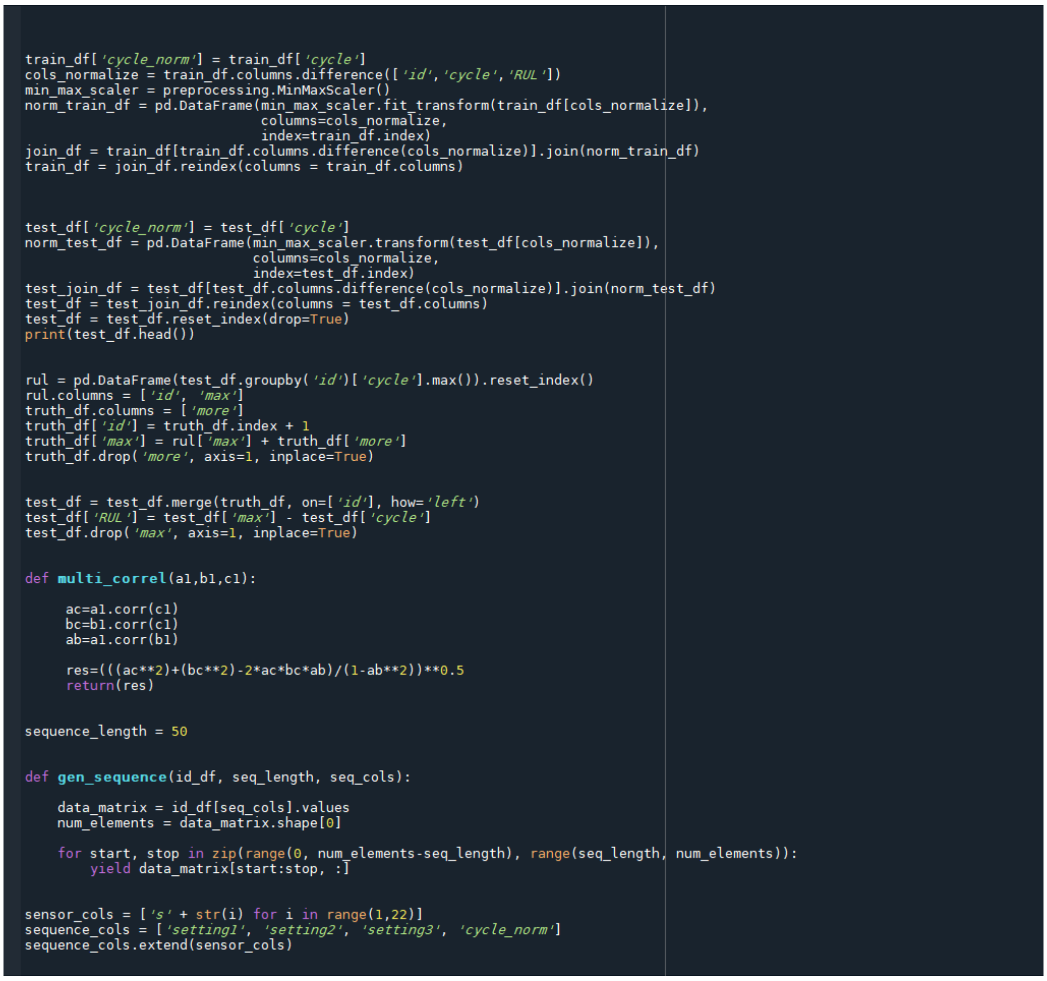
Table.6.1

7. PROJECT DEMONSTRATION:

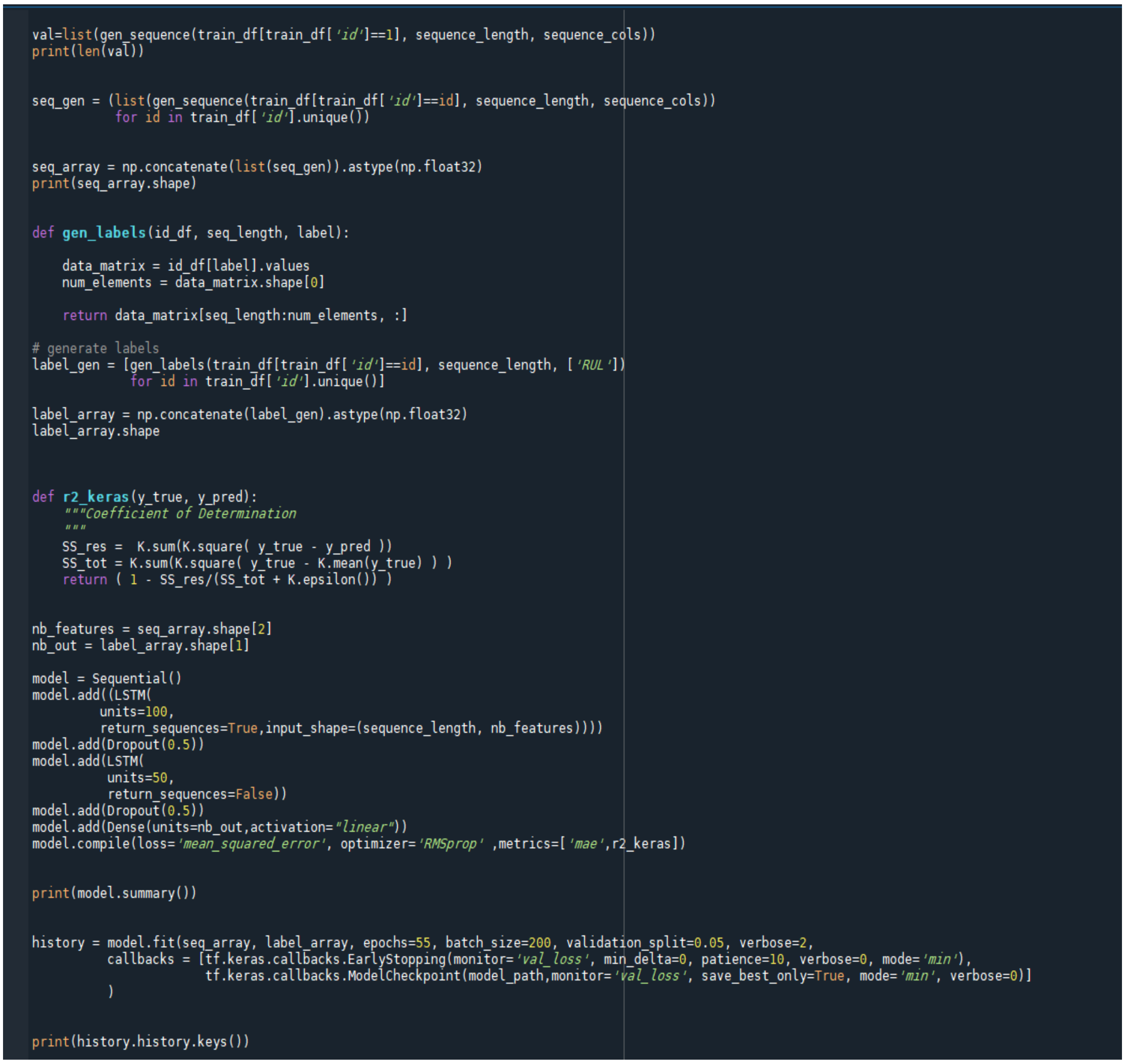
The following is the code for FD001, LSTM:



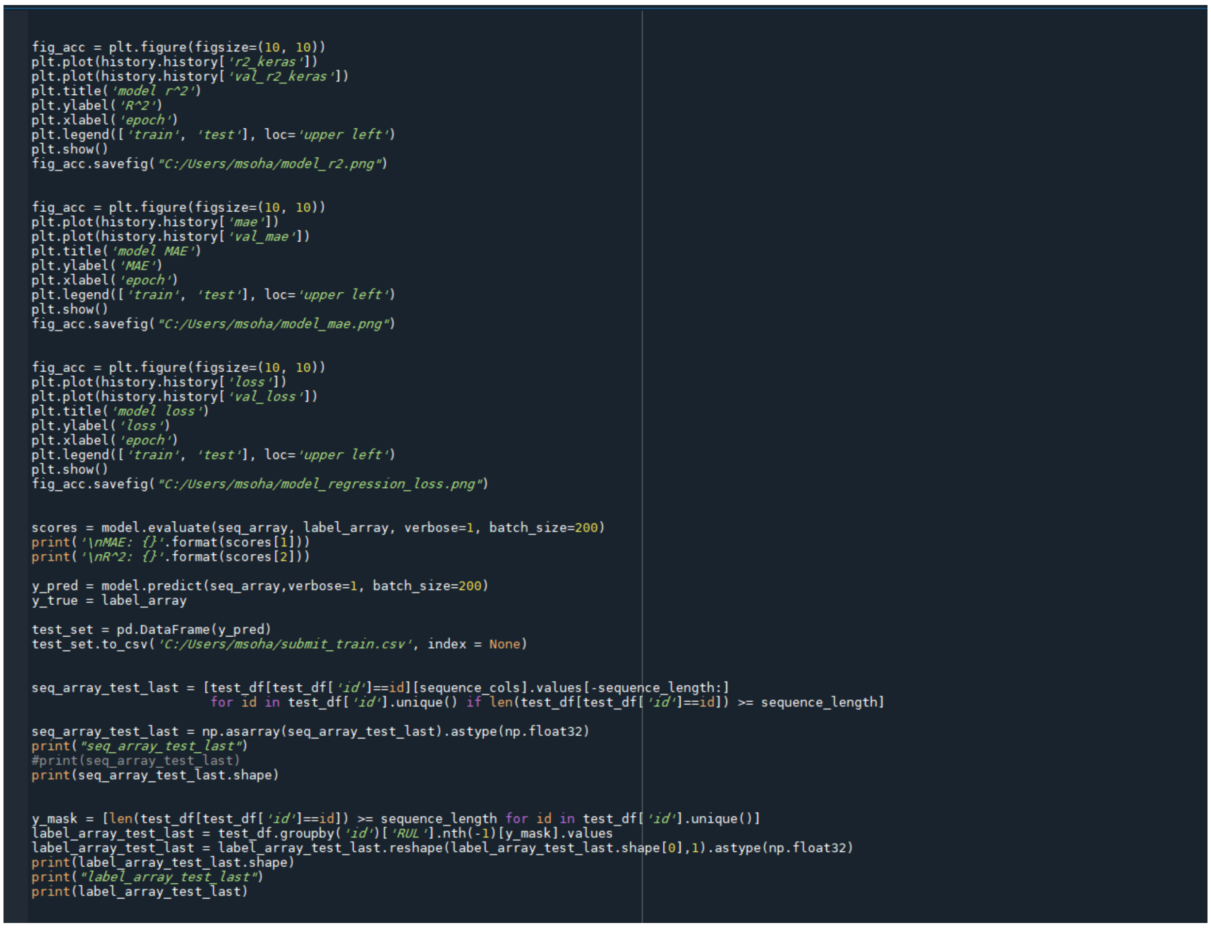
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
import tensorflow.keras.backend as K
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os
from sklearn import preprocessing
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional,GRU
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from sklearn import metrics
from itertools import permutations
import sklearn
np.random.seed(1234)
PYTHONHASHSEED = 0

model_path = 'C:/Users/msoha/OneDrive/Documents/Mywork/models'


def mean_prediction_score(RUL_real, RUL_pred):
    d = RUL_pred - RUL_real

    return (np.sum(np.exp(d[d >= 0] / 13) - 1) +
            np.sum(np.exp(-1 * d[d < 0] / 10) - 1))


train_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/train_FD001.txt', sep=" ", header=None)
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
train_df.columns = ['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']

train_df = train_df.sort_values(['id','cycle'])


test_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/test_FD001.txt', sep=" ", header=None)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
test_df.columns = ['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']

truth_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/RUL_FD001.txt', sep=" ", header=None)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)


rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)
```



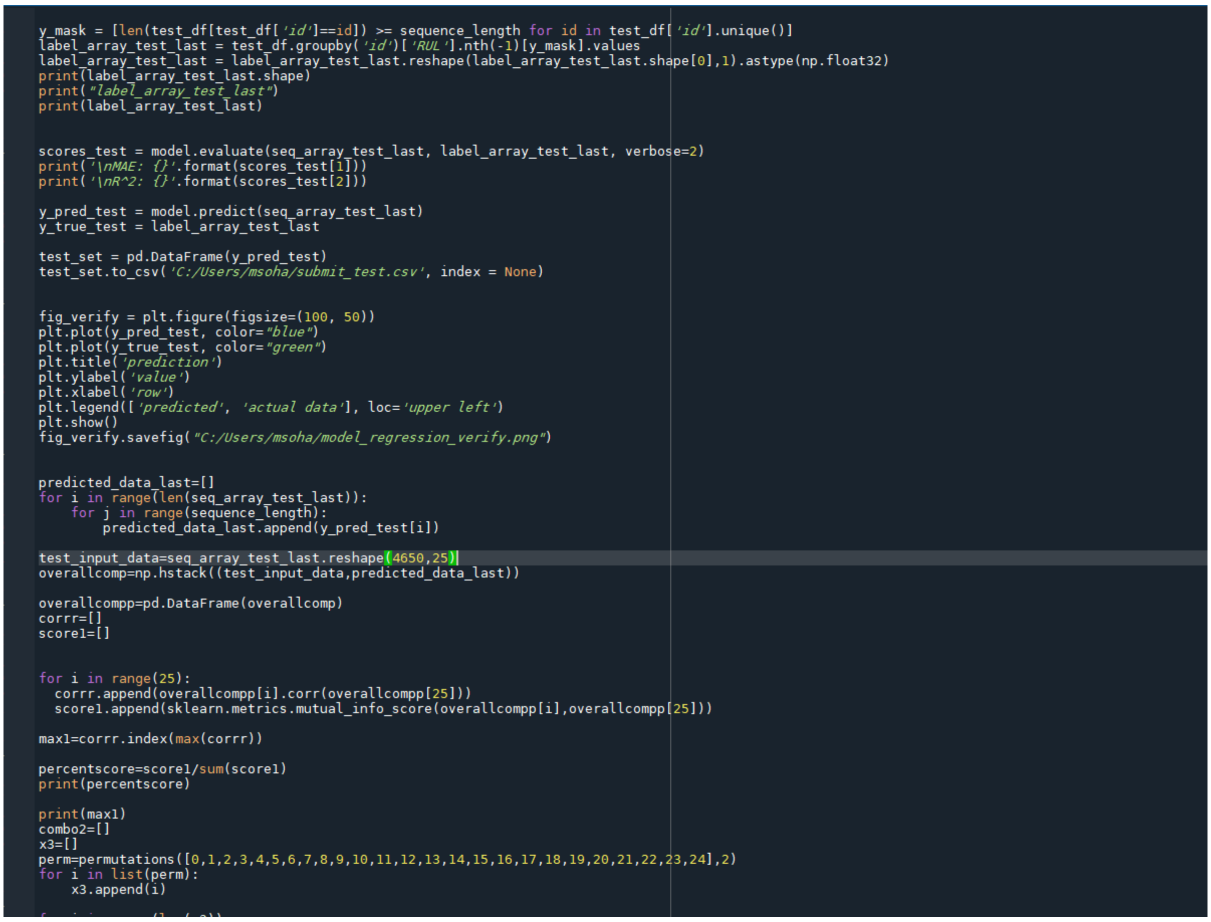
```python
train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)


test_df['cycle_norm'] = test_df['cycle']
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
print(test_df.head())


rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)


test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)


def multi_correl(a1,b1,c1):

     ac=a1.corr(c1)
     bc=b1.corr(c1)
     ab=a1.corr(b1)

     res=(((ac**2)+(bc**2)-2*ac*bc*ab)/(1-ab**2))**0.5
     return(res)


sequence_length = 50


def gen_sequence(id_df, seq_length, seq_cols):

    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]

    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield data_matrix[start:stop, :]


sensor_cols = ['s' + str(i) for i in range(1,22)]
sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols.extend(sensor_cols)
```



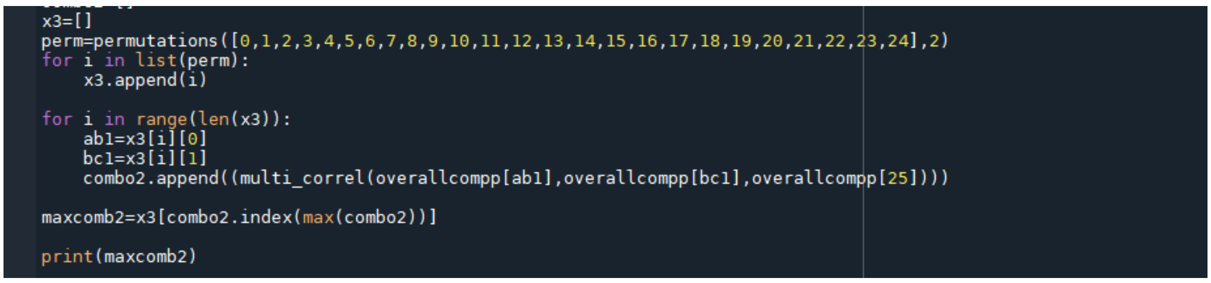
```
val=list(gen_sequence(train_df[train_df['id']==1], sequence_length, sequence_cols))
print(len(val))


seq_gen = (list(gen_sequence(train_df[train_df['id']==id], sequence_length, sequence_cols))
           for id in train_df['id'].unique())


seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)


def gen_labels(id_df, seq_length, label):

    data_matrix = id_df[label].values
    num_elements = data_matrix.shape[0]

    return data_matrix[seq_length:num_elements, :]

# generate labels
label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['RUL'])
             for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
label_array.shape



def r2_keras(y_true, y_pred):
    """Coefficient of Determination
    """
    SS_res =  K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )


nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

model = Sequential()
model.add((LSTM(
         units=100,
         return_sequences=True,input_shape=(sequence_length, nb_features))))
model.add(Dropout(0.5))
model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(units=nb_out,activation="linear"))
model.compile(loss='mean_squared_error', optimizer='RMSprop' ,metrics=['mae',r2_keras])


print(model.summary())


history = model.fit(seq_array, label_array, epochs=55, batch_size=200, validation_split=0.05, verbose=2,
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       tf.keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )


print(history.history.keys())
```



```
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r^2')
plt.ylabel('R^2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_r2.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model MAE')
plt.ylabel('MAE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_mae.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_regression_loss.png")


scores = model.evaluate(seq_array, label_array, verbose=1, batch_size=200)
print('\nMAE: {}'.format(scores[1]))
print('\nR^2: {}'.format(scores[2]))

y_pred = model.predict(seq_array,verbose=1, batch_size=200)
y_true = label_array

test_set = pd.DataFrame(y_pred)
test_set.to_csv('C:/Users/msoha/submit_train.csv', index = None)


seq_array_test_last = [test_df[test_df['id']==id][sequence_cols].values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df[test_df['id']==id]) >= sequence_length]

seq_array_test_last = np.asarray(seq_array_test_last).astype(np.float32)
print("seq_array_test_last")
#print(seq_array_test_last)
print(seq_array_test_last.shape)


y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
label_array_test_last = test_df.groupby('id')['RUL'].nth(-1)[y_mask].values
label_array_test_last = label_array_test_last.reshape(label_array_test_last.shape[0],1).astype(np.float32)
print(label_array_test_last.shape)
print("label_array_test_last")
print(label_array_test_last)
```

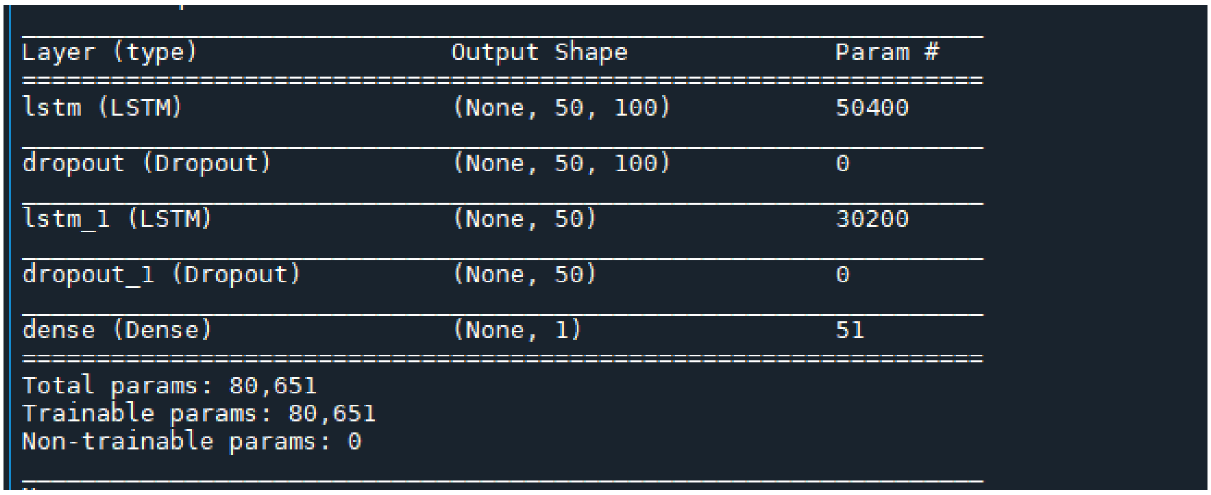


```
y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
label_array_test_last = test_df.groupby('id')['RUL'].nth(-1)[y_mask].values
label_array_test_last = label_array_test_last.reshape(label_array_test_last.shape[0],1).astype(np.float32)
print(label_array_test_last.shape)
print("label_array_test_last")
print(label_array_test_last)


scores_test = model.evaluate(seq_array_test_last, label_array_test_last, verbose=2)
print('\nMAE: {}'.format(scores_test[1]))
print('\nR^2: {}'.format(scores_test[2]))

y_pred_test = model.predict(seq_array_test_last)
y_true_test = label_array_test_last

test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('C:/Users/msoha/submit_test.csv', index = None)


fig_verify = plt.figure(figsize=(100, 50))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
fig_verify.savefig("C:/Users/msoha/model_regression_verify.png")


predicted_data_last=[]
for i in range(len(seq_array_test_last)):
    for j in range(sequence_length):
        predicted_data_last.append(y_pred_test[i])

test_input_data=seq_array_test_last.reshape(4650,25)
overallcomp=np.hstack((test_input_data,predicted_data_last))

overallcompp=pd.DataFrame(overallcomp)
corrr=[]
score1=[]


for i in range(25):
  corrr.append(overallcompp[i].corr(overallcompp[25]))
  score1.append(sklearn.metrics.mutual_info_score(overallcompp[i],overallcompp[25]))

max1=corrr.index(max(corrr))

percentscore=score1/sum(score1)
print(percentscore)

print(max1)
combo2=[]
x3=[]
perm=permutations([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],2)
for i in list(perm):
    x3.append(i)
```

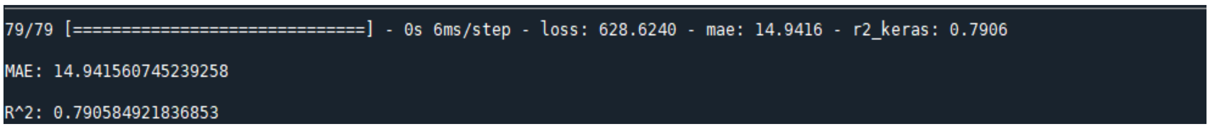


```
x3=[]
perm=permutations([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],2)
for i in list(perm):
    x3.append(i)

for i in range(len(x3)):
    ab1=x3[i][0]
    bc1=x3[i][1]
    combo2.append((multi_correl(overallcompp[ab1],overallcompp[bc1],overallcompp[25])))

maxcomb2=x3[combo2.index(max(combo2))]

print(maxcomb2)
```

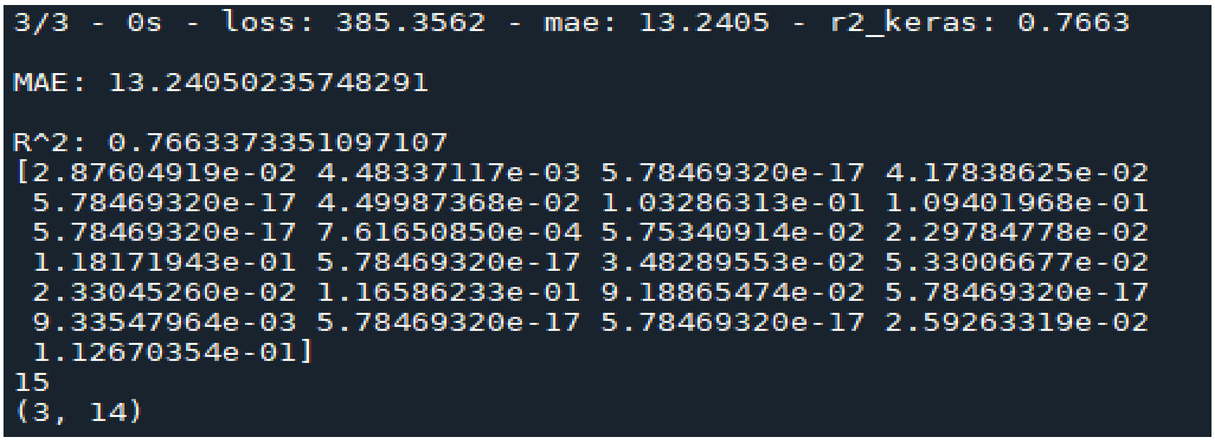
OUTPUT:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50, 100)           50400
_________________________________________________________________
dropout (Dropout)            (None, 50, 100)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                30200
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 80,651
Trainable params: 80,651
Non-trainable params: 0
_________________________________________________________________
```

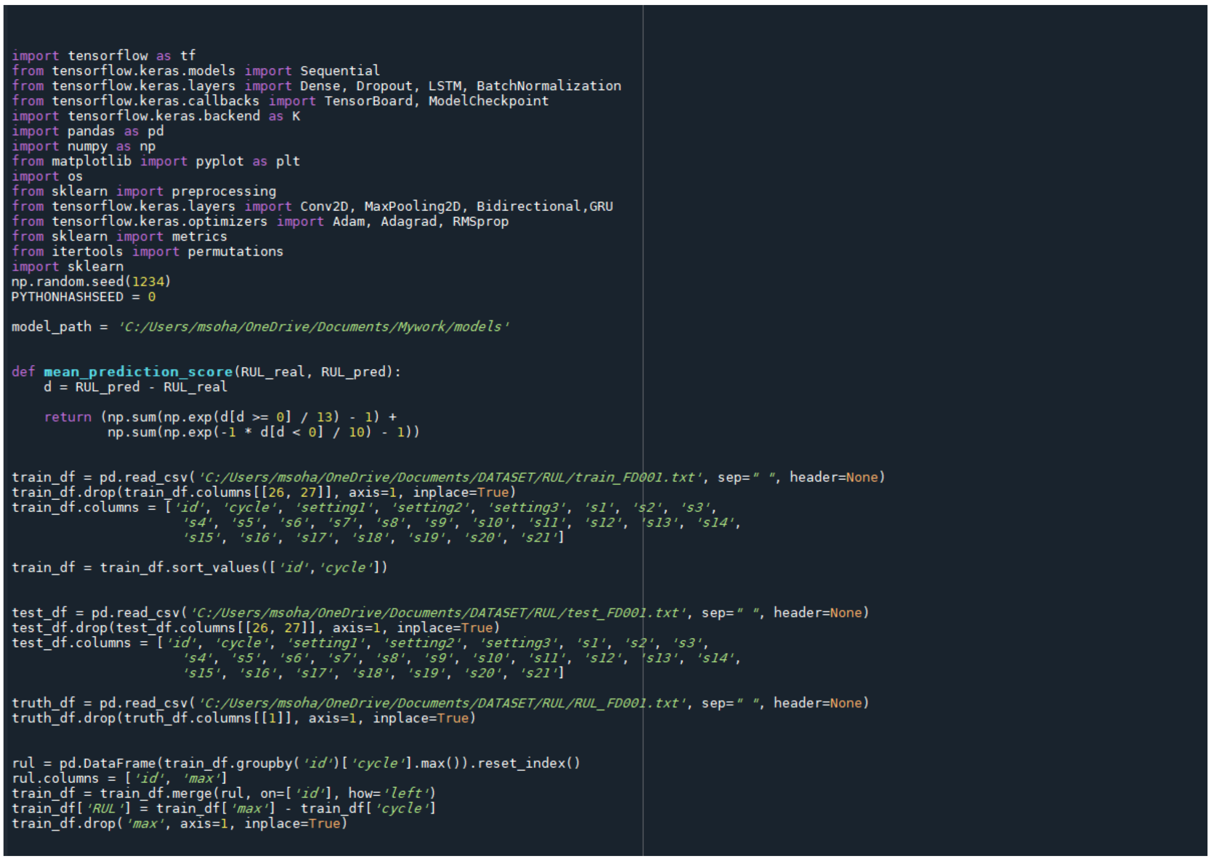


```
79/79 [==============================] - 0s 6ms/step - loss: 628.6240 - mae: 14.9416 - r2_keras: 0.7906

MAE: 14.941560745239258

R^2: 0.790584921836853
```

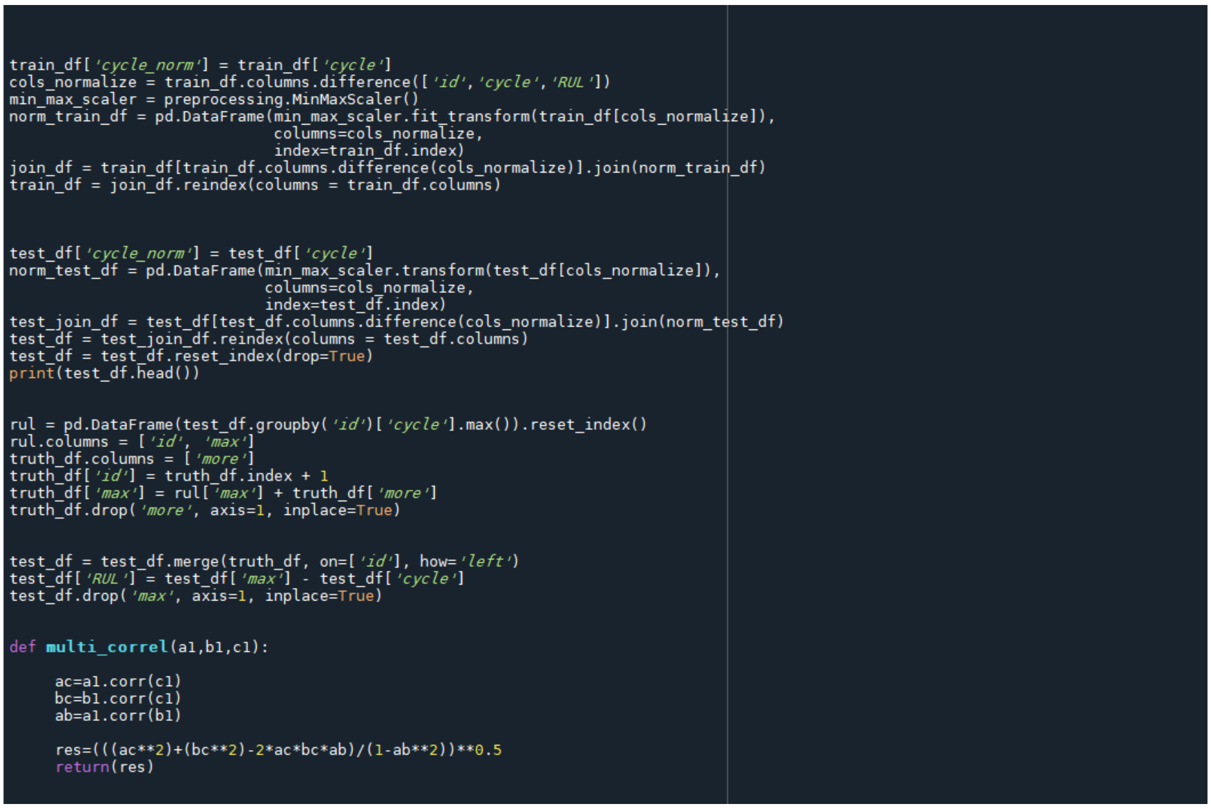


```
3/3 - 0s - loss: 385.3562 - mae: 13.2405 - r2_keras: 0.7663

MAE: 13.240502357482910

R^2: 0.7663373351097107
[2.87604919e-02 4.48337117e-03 5.78469320e-17 4.17838625e-02
 5.78469320e-17 4.49987368e-02 1.03286313e-01 1.09401968e-01
 5.78469320e-17 7.61650850e-04 5.75340914e-02 2.29784778e-02
 1.18171943e-01 5.78469320e-17 3.48289553e-02 5.33006677e-02
 2.33045260e-02 1.16586233e-01 9.18865474e-02 5.78469320e-17
 9.33547964e-03 5.78469320e-17 5.78469320e-17 2.59263319e-02
 1.12670354e-01]
15
(3, 14)
```

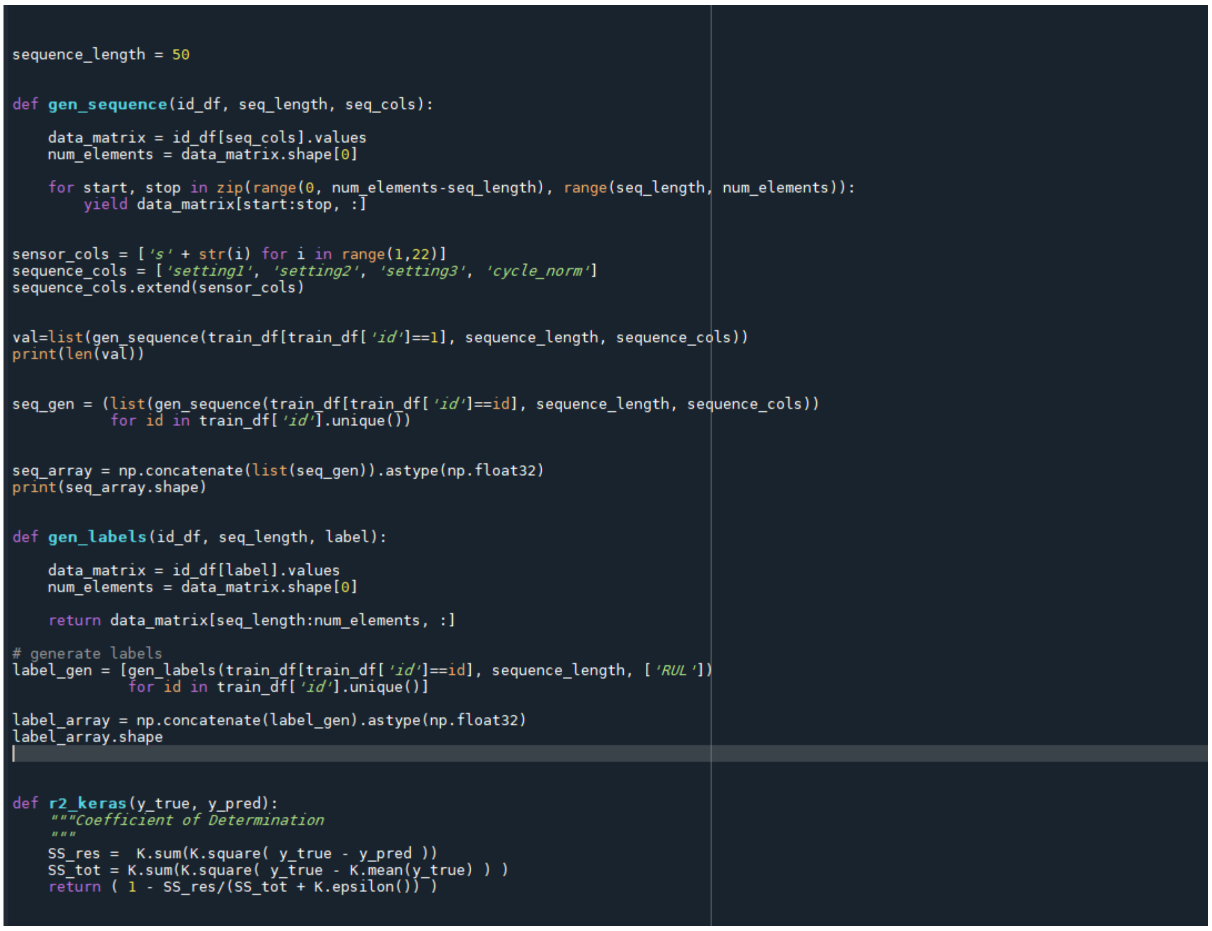
Following is the code for FD001, BiLSTM:



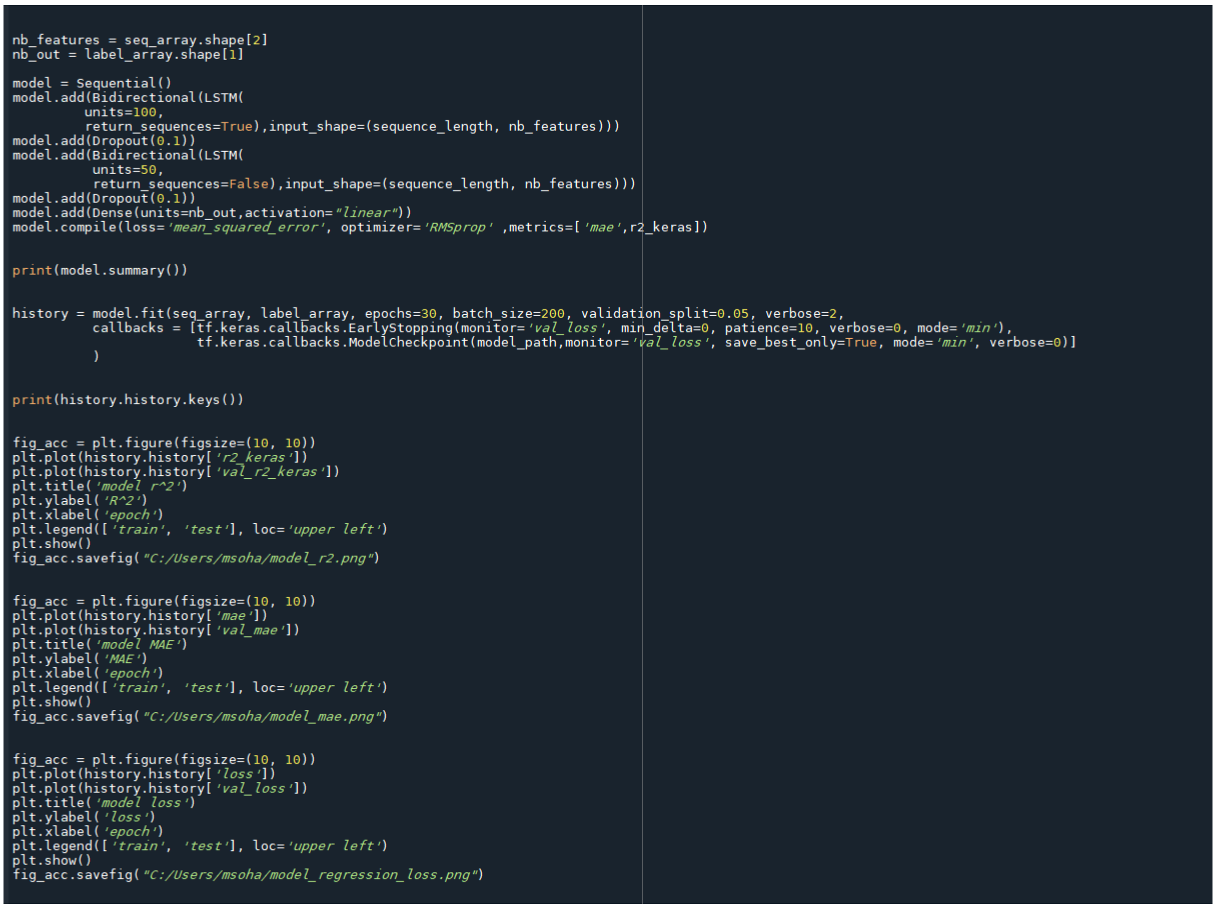
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
import tensorflow.keras.backend as K
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os
from sklearn import preprocessing
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional,GRU
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from sklearn import metrics
from itertools import permutations
import sklearn
np.random.seed(1234)
PYTHONHASHSEED = 0

model_path = 'C:/Users/msoha/OneDrive/Documents/Mywork/models'


def mean_prediction_score(RUL_real, RUL_pred):
    d = RUL_pred - RUL_real

    return (np.sum(np.exp(d[d >= 0] / 13) - 1) +
            np.sum(np.exp(-1 * d[d < 0] / 10) - 1))


train_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/train_FD001.txt', sep=" ", header=None)
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
train_df.columns = ['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']

train_df = train_df.sort_values(['id','cycle'])


test_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/test_FD001.txt', sep=" ", header=None)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
test_df.columns = ['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']

truth_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/RUL_FD001.txt', sep=" ", header=None)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)


rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)
```



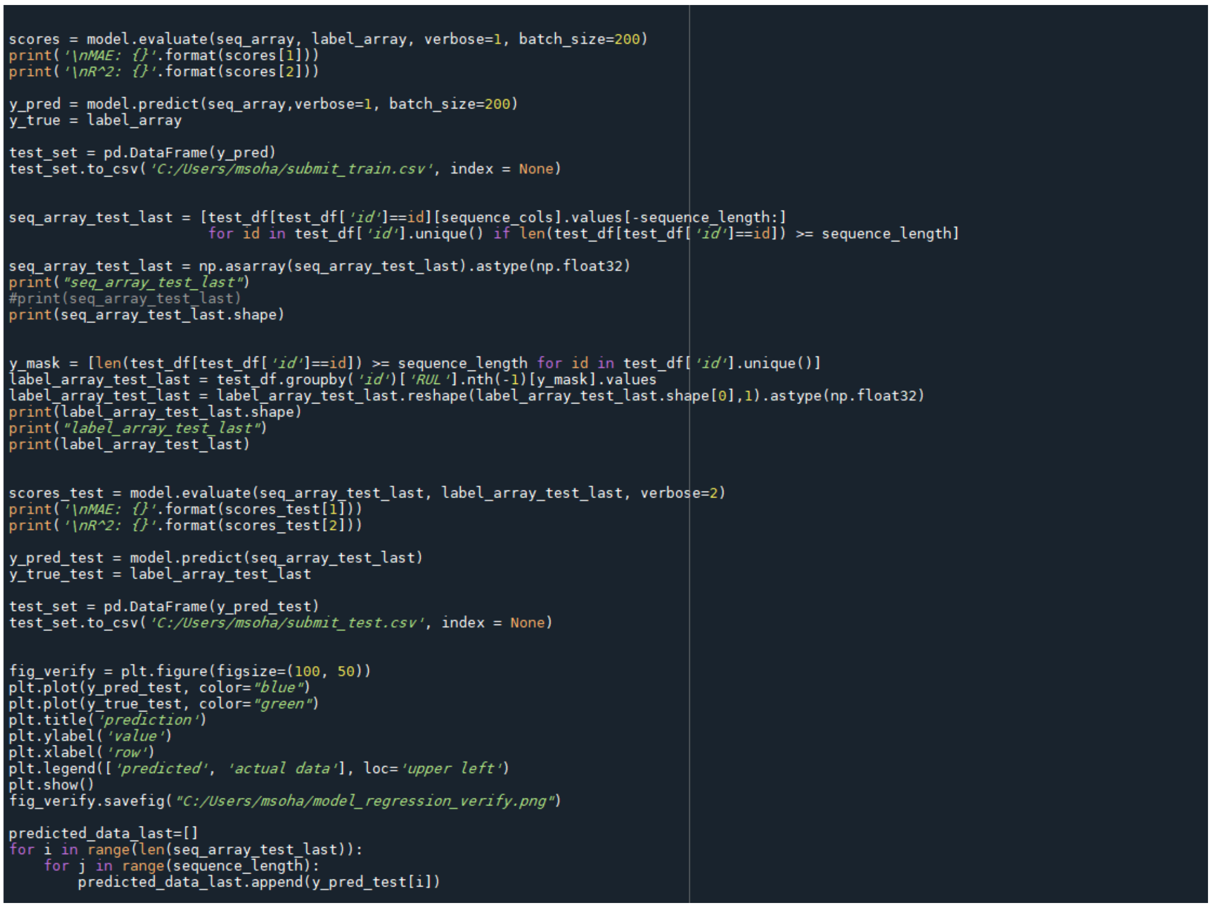
```python
train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)



test_df['cycle_norm'] = test_df['cycle']
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
print(test_df.head())


rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)


test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)


def multi_correl(a1,b1,c1):

     ac=a1.corr(c1)
     bc=b1.corr(c1)
     ab=a1.corr(b1)

     res=(((ac**2)+(bc**2)-2*ac*bc*ab)/(1-ab**2))**0.5
     return(res)
```



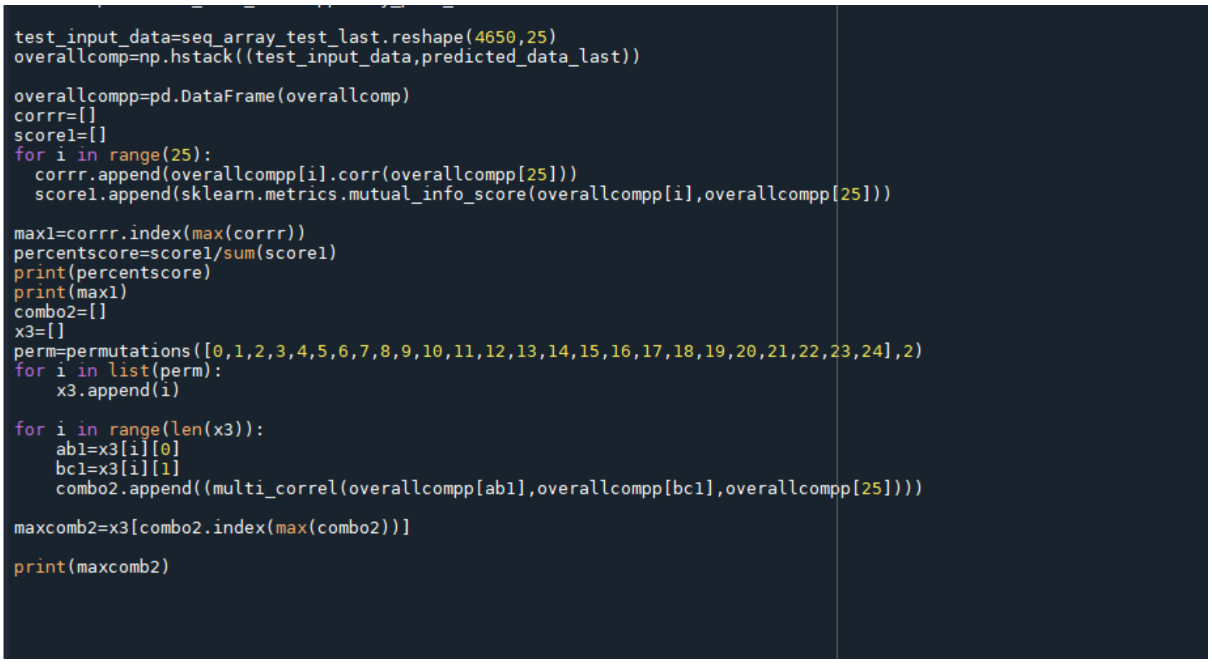
```
sequence_length = 50


def gen_sequence(id_df, seq_length, seq_cols):

    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]

    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield data_matrix[start:stop, :]


sensor_cols = ['s' + str(i) for i in range(1,22)]
sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols.extend(sensor_cols)


val=list(gen_sequence(train_df[train_df['id']==1], sequence_length, sequence_cols))
print(len(val))


seq_gen = (list(gen_sequence(train_df[train_df['id']==id], sequence_length, sequence_cols))
           for id in train_df['id'].unique())


seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)


def gen_labels(id_df, seq_length, label):

    data_matrix = id_df[label].values
    num_elements = data_matrix.shape[0]

    return data_matrix[seq_length:num_elements, :]

# generate labels
label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['RUL'])
             for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
label_array.shape



def r2_keras(y_true, y_pred):
    """Coefficient of Determination
    """
    SS_res =  K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```



```
nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

model = Sequential()
model.add(Bidirectional(LSTM(
         units=100,
         return_sequences=True),input_shape=(sequence_length, nb_features)))
model.add(Dropout(0.1))
model.add(Bidirectional(LSTM(
          units=50,
          return_sequences=False),input_shape=(sequence_length, nb_features)))
model.add(Dropout(0.1))
model.add(Dense(units=nb_out,activation="linear"))
model.compile(loss='mean_squared_error', optimizer='RMSprop' ,metrics=['mae',r2_keras])


print(model.summary())


history = model.fit(seq_array, label_array, epochs=30, batch_size=200, validation_split=0.05, verbose=2,
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       tf.keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )


print(history.history.keys())


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r^2')
plt.ylabel('R^2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_r2.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model MAE')
plt.ylabel('MAE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_mae.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_regression_loss.png")
```

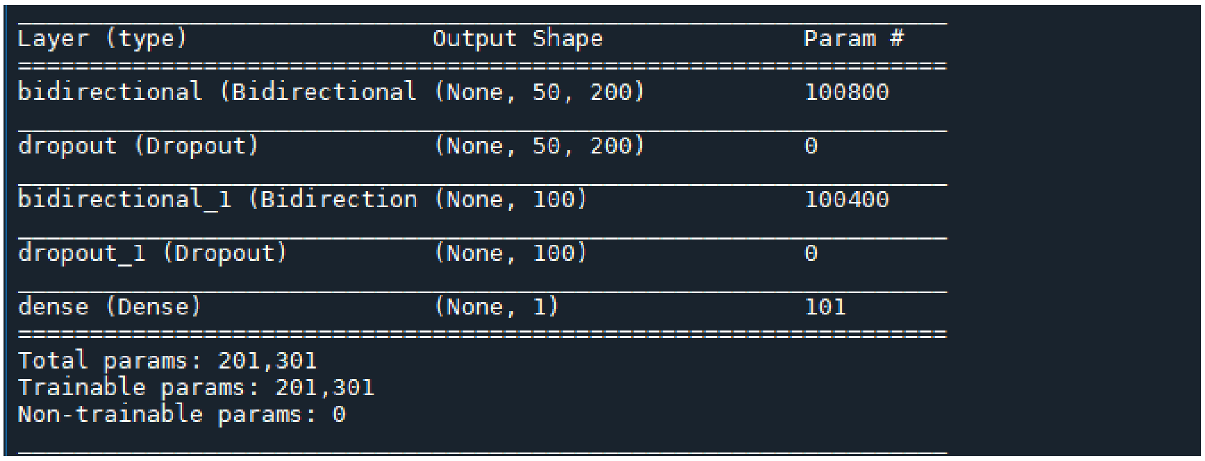


```
scores = model.evaluate(seq_array, label_array, verbose=1, batch_size=200)
print('\nMAE: {}'.format(scores[1]))
print('\nR^2: {}'.format(scores[2]))

y_pred = model.predict(seq_array,verbose=1, batch_size=200)
y_true = label_array

test_set = pd.DataFrame(y_pred)
test_set.to_csv('C:/Users/msoha/submit_train.csv', index = None)


seq_array_test_last = [test_df[test_df['id']==id][sequence_cols].values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df[test_df['id']==id]) >= sequence_length]

seq_array_test_last = np.asarray(seq_array_test_last).astype(np.float32)
print("seq_array_test_last")
#print(seq_array_test_last)
print(seq_array_test_last.shape)


y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
label_array_test_last = test_df.groupby('id')['RUL'].nth(-1)[y_mask].values
label_array_test_last = label_array_test_last.reshape(label_array_test_last.shape[0],1).astype(np.float32)
print(label_array_test_last.shape)
print("label_array_test_last")
print(label_array_test_last)


scores_test = model.evaluate(seq_array_test_last, label_array_test_last, verbose=2)
print('\nMAE: {}'.format(scores_test[1]))
print('\nR^2: {}'.format(scores_test[2]))

y_pred_test = model.predict(seq_array_test_last)
y_true_test = label_array_test_last

test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('C:/Users/msoha/submit_test.csv', index = None)


fig_verify = plt.figure(figsize=(100, 50))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
fig_verify.savefig("C:/Users/msoha/model_regression_verify.png")

predicted_data_last=[]
for i in range(len(seq_array_test_last)):
    for j in range(sequence_length):
        predicted_data_last.append(y_pred_test[i])
```

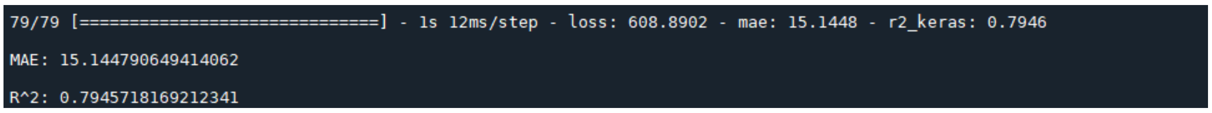


```
test_input_data=seq_array_test_last.reshape(4650,25)
overallcomp=np.hstack((test_input_data,predicted_data_last))

overallcompp=pd.DataFrame(overallcomp)
corrr=[]
score1=[]
for i in range(25):
  corrr.append(overallcompp[i].corr(overallcompp[25]))
  score1.append(sklearn.metrics.mutual_info_score(overallcompp[i],overallcompp[25]))

max1=corrr.index(max(corrr))
percentscore=score1/sum(score1)
print(percentscore)
print(max1)
combo2=[]
x3=[]
perm=permutations([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],2)
for i in list(perm):
    x3.append(i)

for i in range(len(x3)):
    ab1=x3[i][0]
    bc1=x3[i][1]
    combo2.append((multi_correl(overallcompp[ab1],overallcompp[bc1],overallcompp[25])))

maxcomb2=x3[combo2.index(max(combo2))]

print(maxcomb2)
```

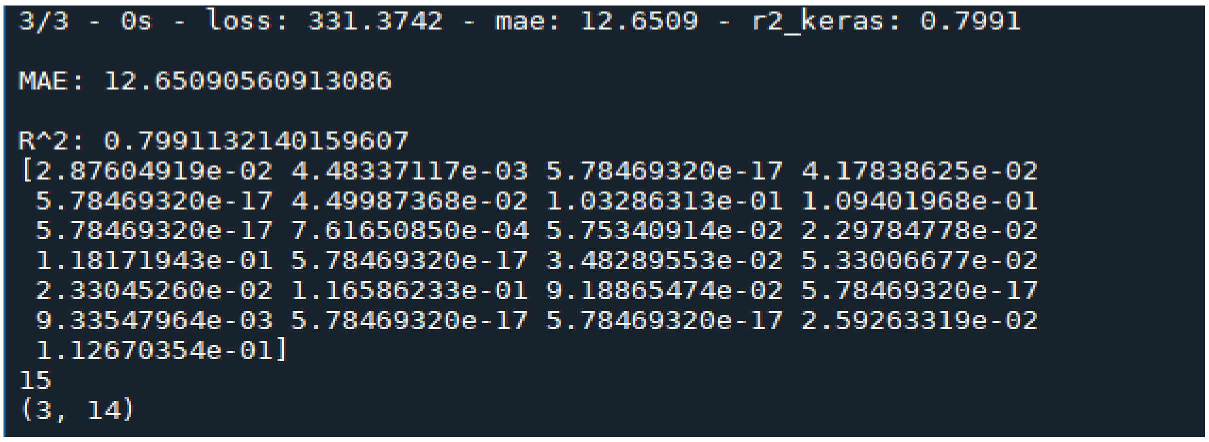
OUTPUT:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 50, 200)           100800
_________________________________________________________________
dropout (Dropout)            (None, 50, 200)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 100)               100400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 201,301
Trainable params: 201,301
Non-trainable params: 0
_________________________________________________________________
```

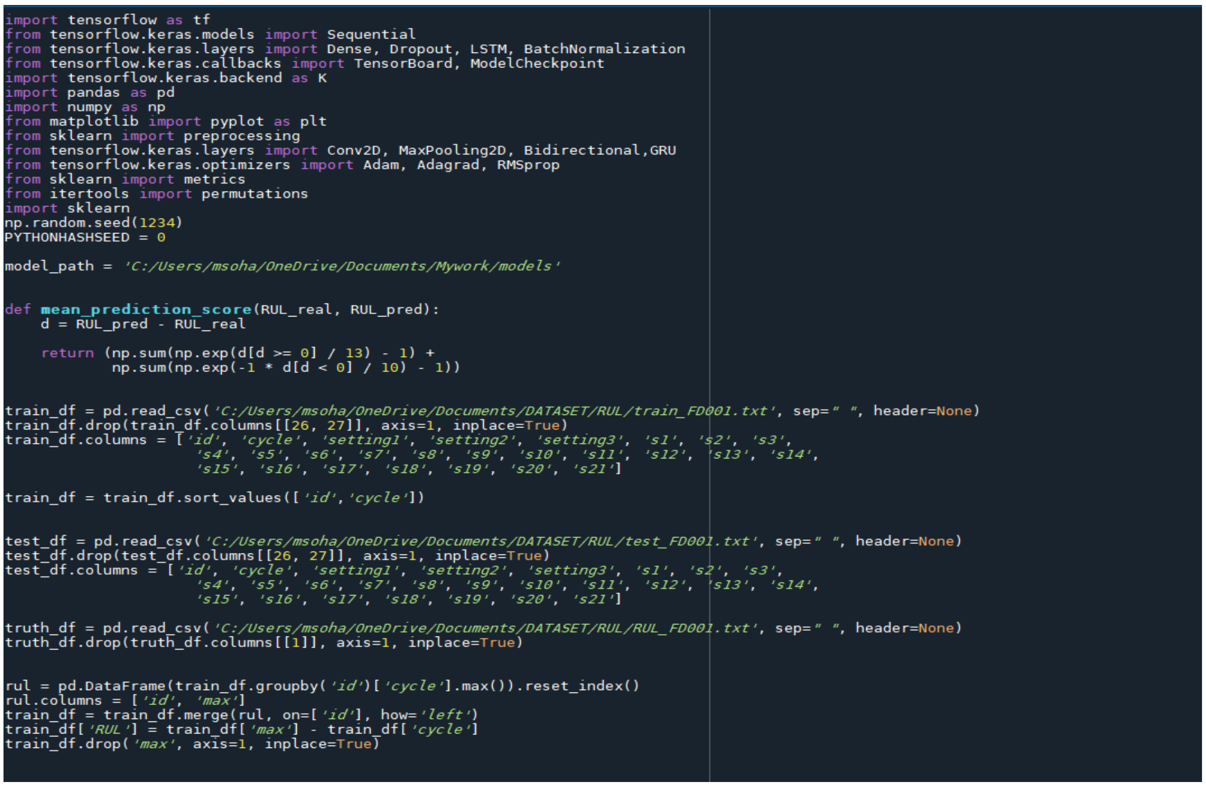


```
79/79 [==============================] - 1s 12ms/step - loss: 608.8902 - mae: 15.1448 - r2_keras: 0.7946

MAE: 15.144790649414062

R^2: 0.7945718169212341
```

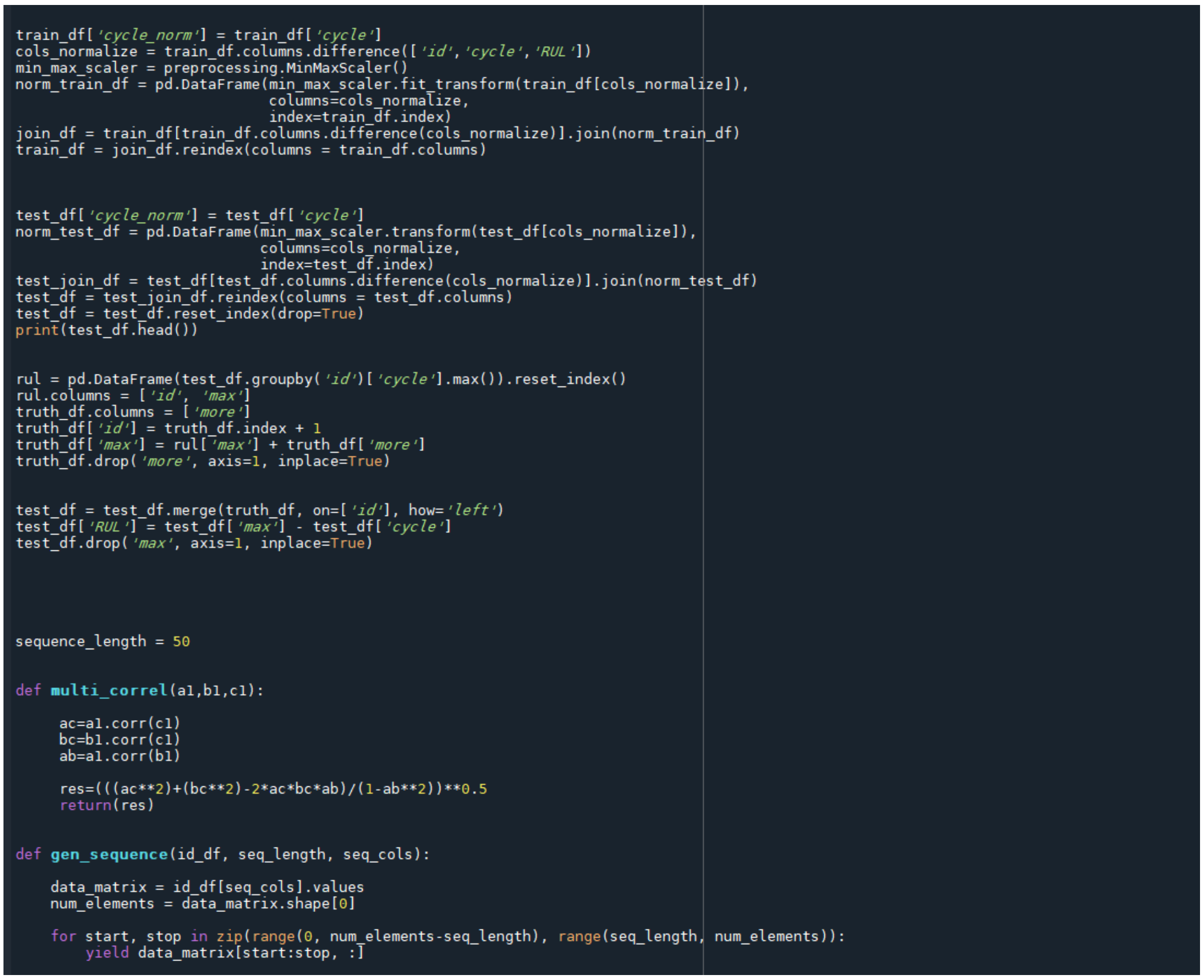


```
3/3 - 0s - loss: 331.3742 - mae: 12.6509 - r2_keras: 0.7991

MAE: 12.65090560913086

R^2: 0.7991132140159607
[2.87604919e-02 4.48337117e-03 5.78469320e-17 4.17838625e-02
 5.78469320e-17 4.49987368e-02 1.03286313e-01 1.09401968e-01
 5.78469320e-17 7.61650850e-04 5.75340914e-02 2.29784778e-02
 1.18171943e-01 5.78469320e-17 3.48289553e-02 5.33006677e-02
 2.33045260e-02 1.16586233e-01 9.18865474e-02 5.78469320e-17
 9.33547964e-03 5.78469320e-17 5.78469320e-17 2.59263319e-02
 1.12670354e-01]
15
(3, 14)
```

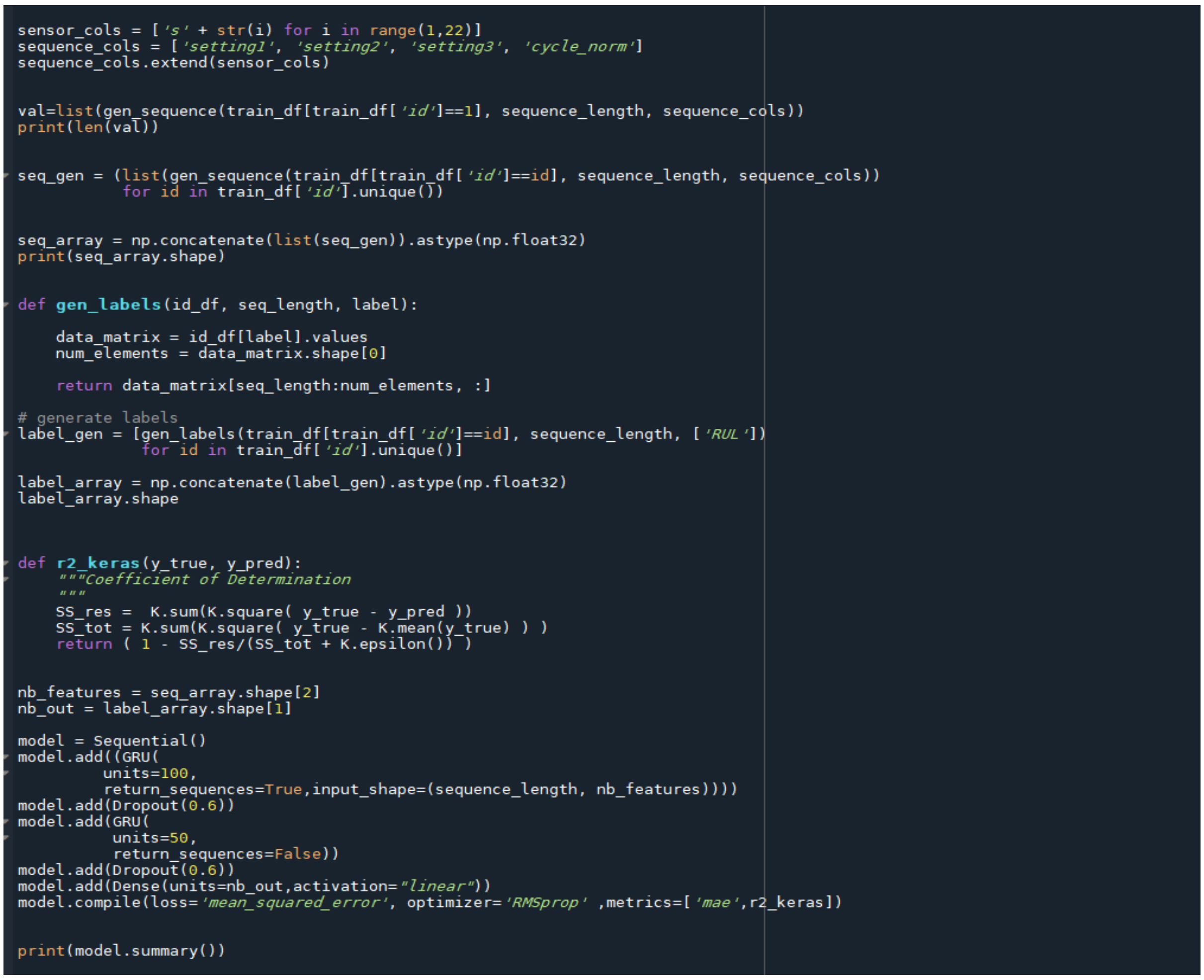
The following is the code for FD001, GRU:



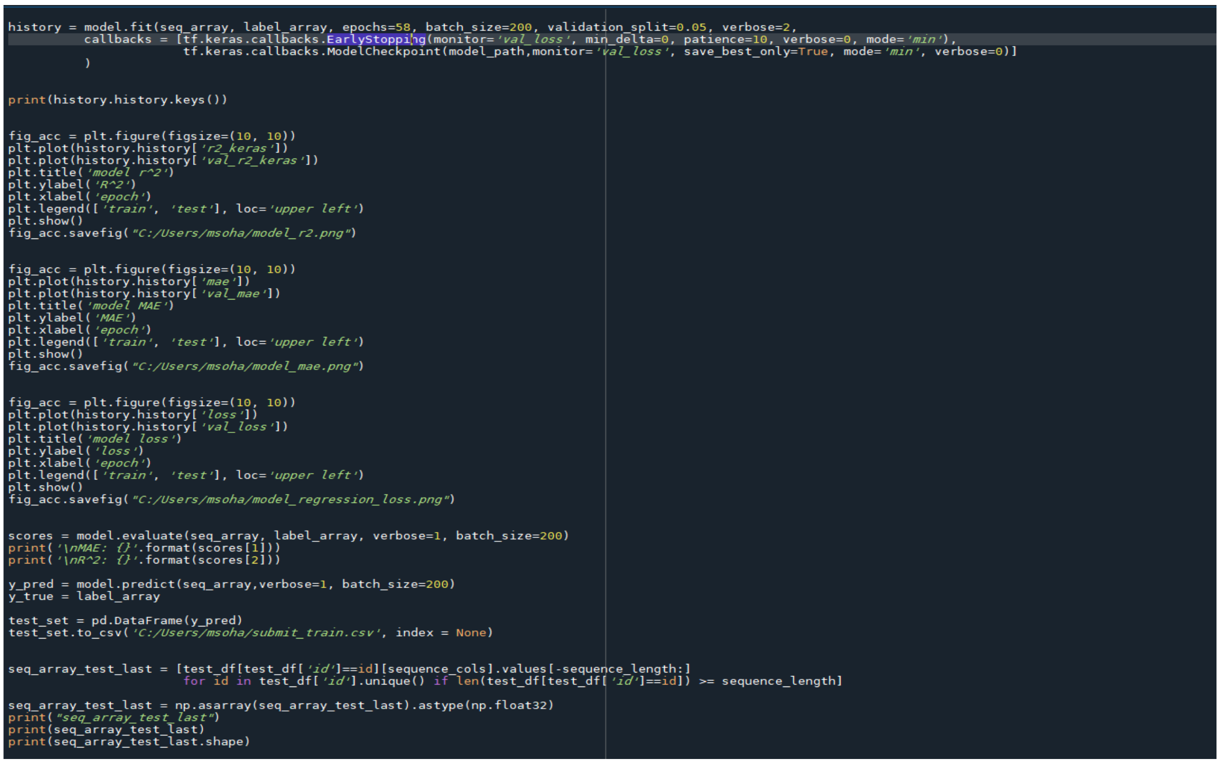
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
import tensorflow.keras.backend as K
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import preprocessing
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional,GRU
from tensorflow.keras.optimizers import Adam, Adagrad, RMSprop
from sklearn import metrics
from itertools import permutations
import sklearn
np.random.seed(1234)
PYTHONHASHSEED = 0

model_path = 'C:/Users/msoha/OneDrive/Documents/Mywork/models'


def mean_prediction_score(RUL_real, RUL_pred):
    d = RUL_pred - RUL_real

    return (np.sum(np.exp(d[d >= 0] / 13) - 1) +
            np.sum(np.exp(-1 * d[d < 0] / 10) - 1))


train_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/train_FD001.txt', sep=" ", header=None)
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
train_df.columns = ['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']

train_df = train_df.sort_values(['id','cycle'])


test_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/test_FD001.txt', sep=" ", header=None)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
test_df.columns = ['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']

truth_df = pd.read_csv('C:/Users/msoha/OneDrive/Documents/DATASET/RUL/RUL_FD001.txt', sep=" ", header=None)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)


rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)
```



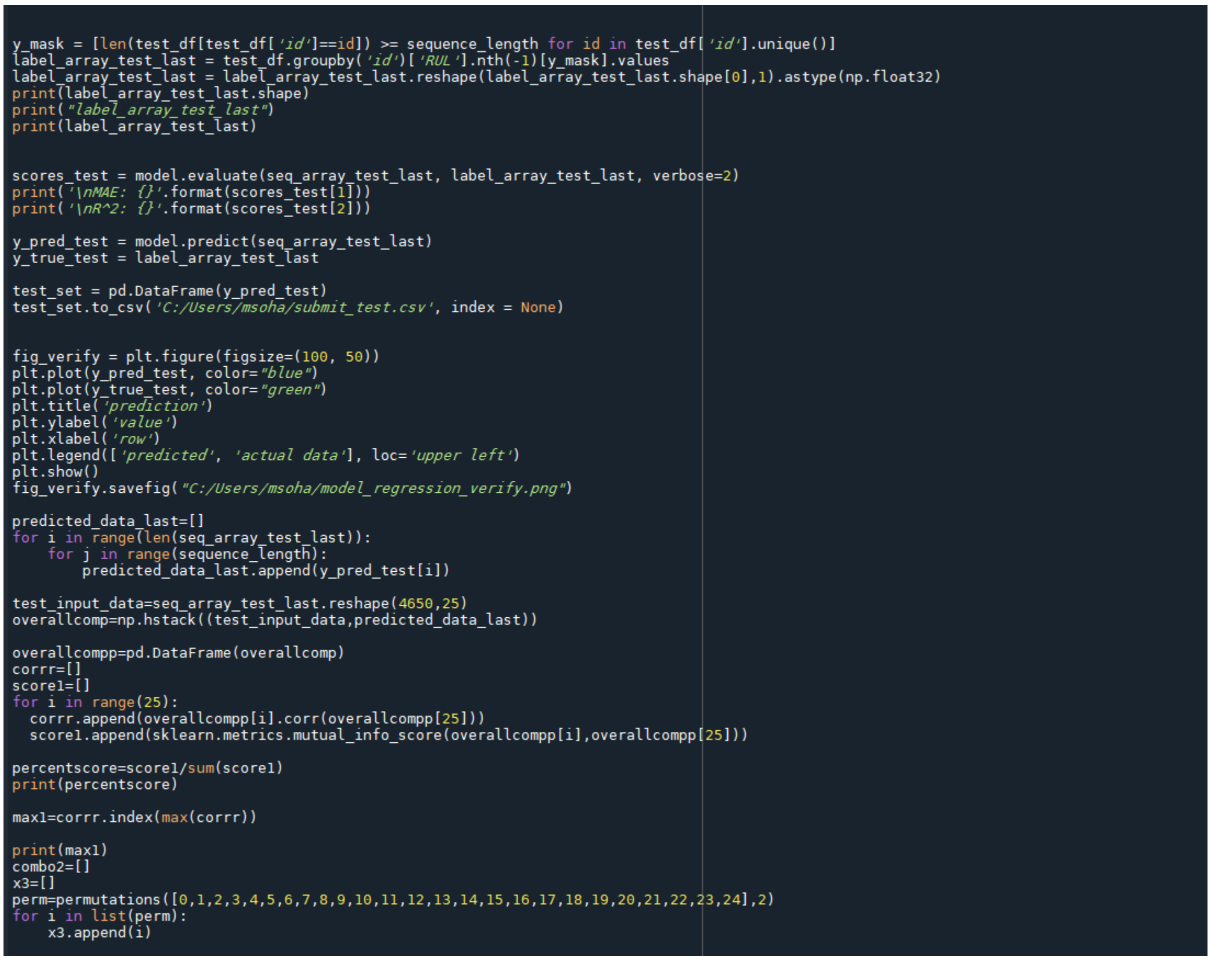
```python
train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id','cycle','RUL'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)



test_df['cycle_norm'] = test_df['cycle']
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
print(test_df.head())


rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)


test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)




sequence_length = 50


def multi_correl(a1,b1,c1):

     ac=a1.corr(c1)
     bc=b1.corr(c1)
     ab=a1.corr(b1)

     res=(((ac**2)+(bc**2)-2*ac*bc*ab)/(1-ab**2))**0.5
     return(res)


def gen_sequence(id_df, seq_length, seq_cols):

    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]

    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield data_matrix[start:stop, :]
```



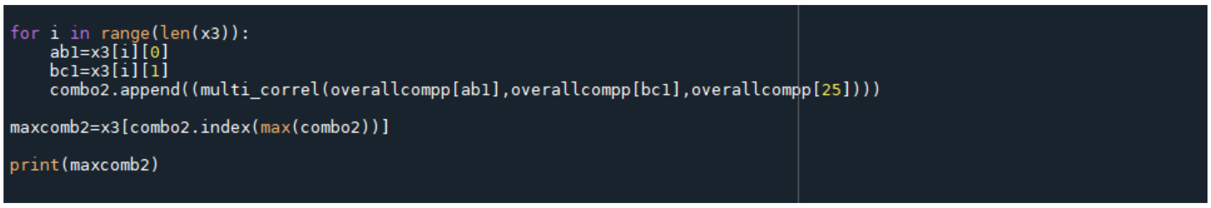
```python
sensor_cols = ['s' + str(i) for i in range(1,22)]
sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols.extend(sensor_cols)


val=list(gen_sequence(train_df[train_df['id']==1], sequence_length, sequence_cols))
print(len(val))


seq_gen = (list(gen_sequence(train_df[train_df['id']==id], sequence_length, sequence_cols))
             for id in train_df['id'].unique())


seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)


def gen_labels(id_df, seq_length, label):

    data_matrix = id_df[label].values
    num_elements = data_matrix.shape[0]

    return data_matrix[seq_length:num_elements, :]

# generate labels
label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['RUL'])
                for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
label_array.shape


def r2_keras(y_true, y_pred):
    """Coefficient of Determination
    """
    SS_res =  K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )


nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

model = Sequential()
model.add((GRU(
         units=100,
         return_sequences=True,input_shape=(sequence_length, nb_features))))
model.add(Dropout(0.6))
model.add(GRU(
          units=50,
          return_sequences=False))
model.add(Dropout(0.6))
model.add(Dense(units=nb_out,activation="linear"))
model.compile(loss='mean_squared_error', optimizer='RMSprop' ,metrics=['mae',r2_keras])


print(model.summary())
```



```python
history = model.fit(seq_array, label_array, epochs=58, batch_size=200, validation_split=0.05, verbose=2,
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       tf.keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )


print(history.history.keys())


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r^2')
plt.ylabel('R^2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_r2.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model MAE')
plt.ylabel('MAE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_mae.png")


fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("C:/Users/msoha/model_regression_loss.png")


scores = model.evaluate(seq_array, label_array, verbose=1, batch_size=200)
print('\nMAE: {}'.format(scores[1]))
print('\nR^2: {}'.format(scores[2]))

y_pred = model.predict(seq_array,verbose=1, batch_size=200)
y_true = label_array

test_set = pd.DataFrame(y_pred)
test_set.to_csv('C:/Users/msoha/submit_train.csv', index = None)


seq_array_test_last = [test_df[test_df['id']==id][sequence_cols].values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df[test_df['id']==id]) >= sequence_length]

seq_array_test_last = np.asarray(seq_array_test_last).astype(np.float32)
print("seq_array_test_last")
print(seq_array_test_last)
print(seq_array_test_last.shape)
```

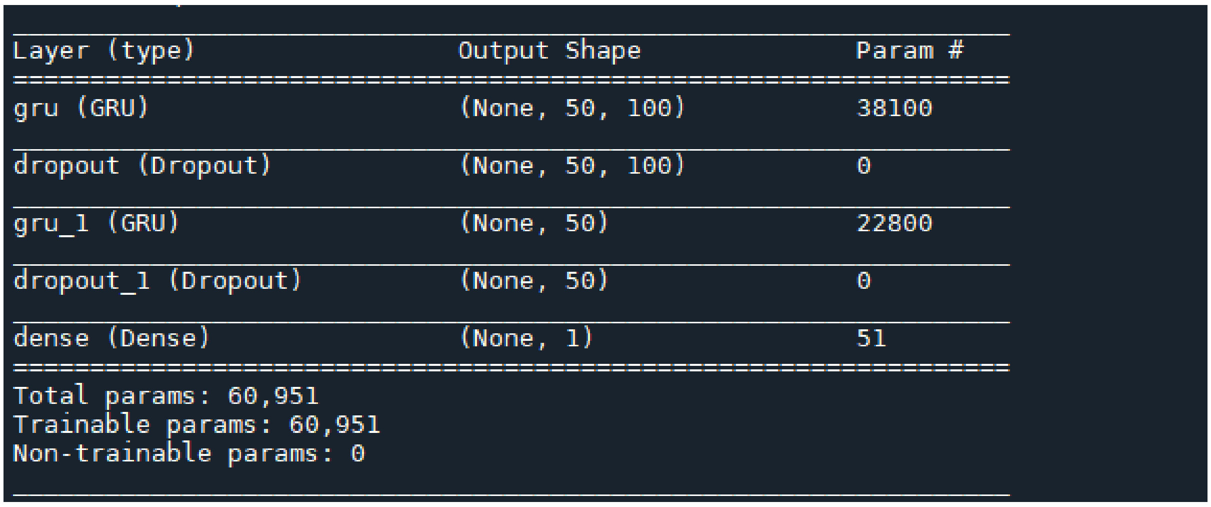


```python
y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
label_array_test_last = test_df.groupby('id')['RUL'].nth(-1)[y_mask].values
label_array_test_last = label_array_test_last.reshape(label_array_test_last.shape[0],1).astype(np.float32)
print(label_array_test_last.shape)
print("label_array_test_last")
print(label_array_test_last)


scores_test = model.evaluate(seq_array_test_last, label_array_test_last, verbose=2)
print('\nMAE: {}'.format(scores_test[1]))
print('\nR^2: {}'.format(scores_test[2]))

y_pred_test = model.predict(seq_array_test_last)
y_true_test = label_array_test_last

test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('C:/Users/msoha/submit_test.csv', index = None)


fig_verify = plt.figure(figsize=(100, 50))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
fig_verify.savefig("C:/Users/msoha/model_regression_verify.png")

predicted_data_last=[]
for i in range(len(seq_array_test_last)):
    for j in range(sequence_length):
        predicted_data_last.append(y_pred_test[i])

test_input_data=seq_array_test_last.reshape(4650,25)
overallcomp=np.hstack((test_input_data,predicted_data_last))

overallcompp=pd.DataFrame(overallcomp)
corrr=[]
score1=[]
for i in range(25):
  corrr.append(overallcompp[i].corr(overallcompp[25]))
  score1.append(sklearn.metrics.mutual_info_score(overallcompp[i],overallcompp[25]))

percentscore=score1/sum(score1)
print(percentscore)

max1=corrr.index(max(corrr))

print(max1)
combo2=[]
x3=[]
perm=permutations([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],2)
for i in list(perm):
    x3.append(i)
```

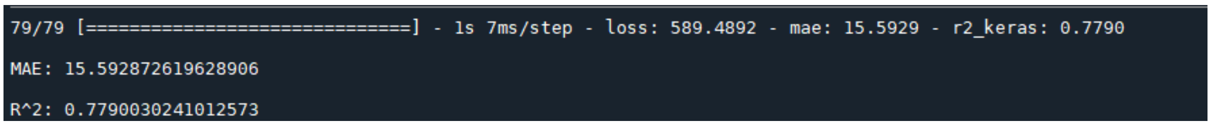


```python
for i in range(len(x3)):
    ab1=x3[i][0]
    bc1=x3[i][1]
    combo2.append((multi_correl(overallcompp[ab1],overallcompp[bc1],overallcompp[25])))

maxcomb2=x3[combo2.index(max(combo2))]

print(maxcomb2)
```

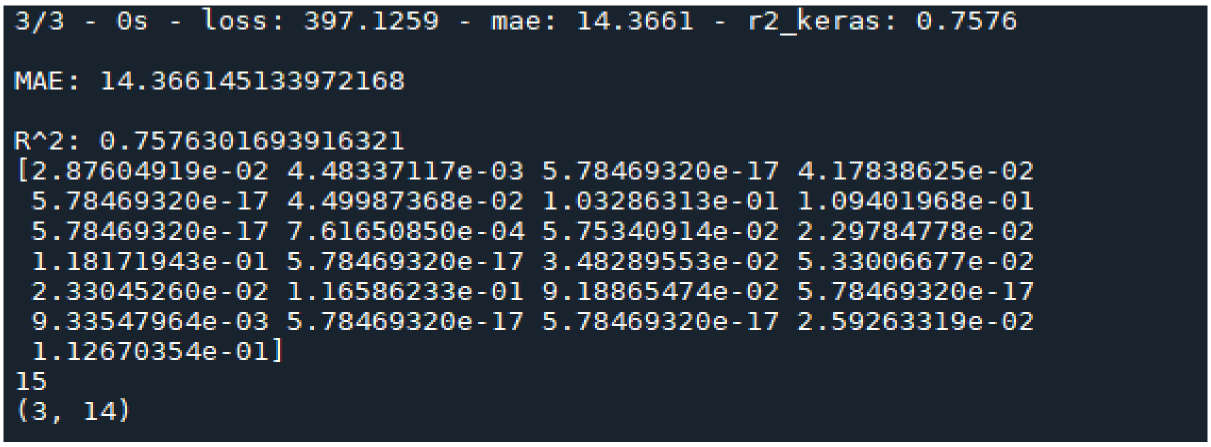
OUTPUT:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 50, 100)           38100
_________________________________________________________________
dropout (Dropout)            (None, 50, 100)           0
_________________________________________________________________
gru_1 (GRU)                  (None, 50)                22800
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 60,951
Trainable params: 60,951
Non-trainable params: 0
_________________________________________________________________
```



```
79/79 [==============================] - 1s 7ms/step - loss: 589.4892 - mae: 15.5929 - r2_keras: 0.7790

MAE: 15.592872619628906

R^2: 0.7790030241012573
```



```
3/3 - 0s - loss: 397.1259 - mae: 14.3661 - r2_keras: 0.7576

MAE: 14.366145133972168

R^2: 0.7576301693916321
[2.87604919e-02 4.48337117e-03 5.78469320e-17 4.17838625e-02
 5.78469320e-17 4.49987368e-02 1.03286313e-01 1.09401968e-01
 5.78469320e-17 7.61650850e-04 5.75340914e-02 2.29784778e-02
 1.18171943e-01 5.78469320e-17 3.48289553e-02 5.33006677e-02
 2.33045260e-02 1.16586233e-01 9.18865474e-02 5.78469320e-17
 9.33547964e-03 5.78469320e-17 5.78469320e-17 2.59263319e-02
 1.12670354e-01]
15
(3, 14)
```

8. RESULTS AND DISCUSSION:

For the ease of mentioning sensors only the indexes of the sensors and settings are mentioned.

The LSTM Algorithm are made to run for 4 Subsets of the Data set which have different conditions imposed on them as mentioned. After compiling the results were recorded as follows

8.1 LSTM FD001:

Parameters of the model:

Parameter	Value
Loss	672.18
MAE	15.935
R^2	0.7563

Table.8.1(a)

The following are the iteration graphs or the change observed per epoch:

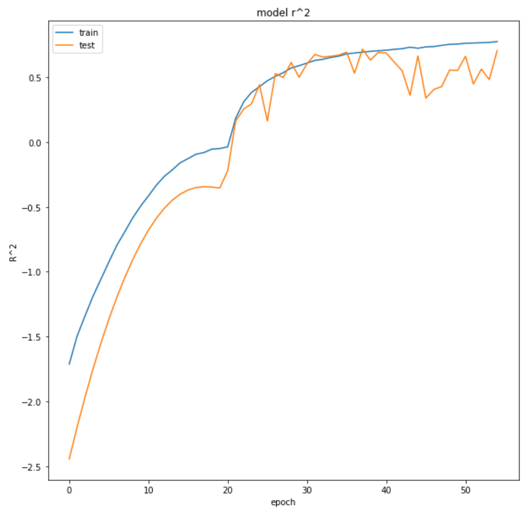


Fig.8.1(a) R^2 vs epochs

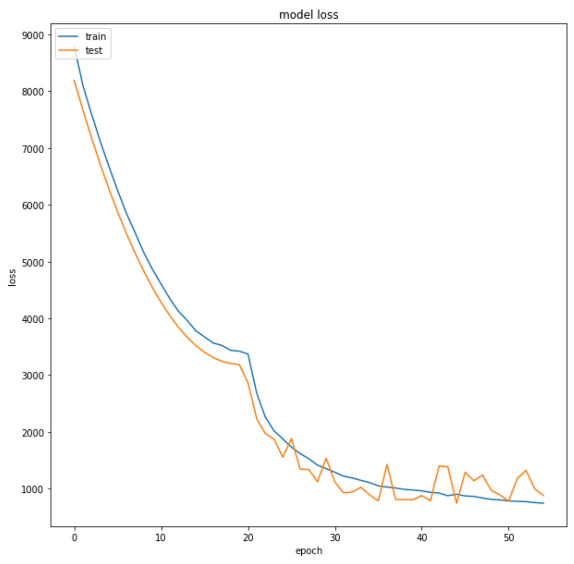


Fig.8.1(b) loss vs epochs

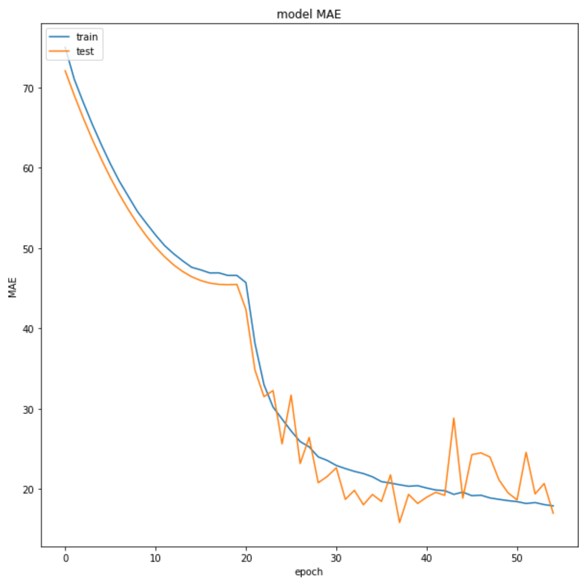


Fig.8.1(c) MAE vs epoch

The priority order or ranking with respect correlation was as follows:

[21 17 20 10 6 16 24 25 15 14 13 12 1 2]

Remaining sensors don't show correlation to the predicted RUL individually hence, some sensor indexes have been omitted as they show no correlation individually whatsoever

If given a chance by for the servicing of two variables, 4 and 15 should be increased to increase RUL as prescribed by the correlation parameter.

Mutual Information Score:

Individual score of each variable

1	Id	0.0287605
2	Setting1	0.00448337
3	Setting2	5.78469e-17
4	Setting3	0.0417839
5	T24	5.78469e-17
6	T30	0.0449987
7	T50	0.103286
8	P2	0.109402
9	P15	0.000761651
10	P30	0.0575341
11	Nf	0.0229785
12	Nc	0.118172
13	Epr	5.78469e-17
14	Ps30	0.034829
15	Phi	0.0533007
16	NRf	0.0233045
17	NRc	0.116586
18	BPR	0.0918865
19	farB	5.78469e-17
20	htBleed	0.00933548
21	Nf_dmd	5.78469e-17
22	PCNfR_dmd	5.78469e-17
23	W31	0.0259263
24	W32	0.11267
25	T24	0.112670

Table.8.3(b) confidence values with sensor and setting index

8.2 BILSTM FD001:

Parameters of the model:

Parameter	Value
Loss	928.1616
MAE	29.6583
R^2	0.6679

Table.8.2(a)

The following are the iteration graphs or the change observed per epoch:

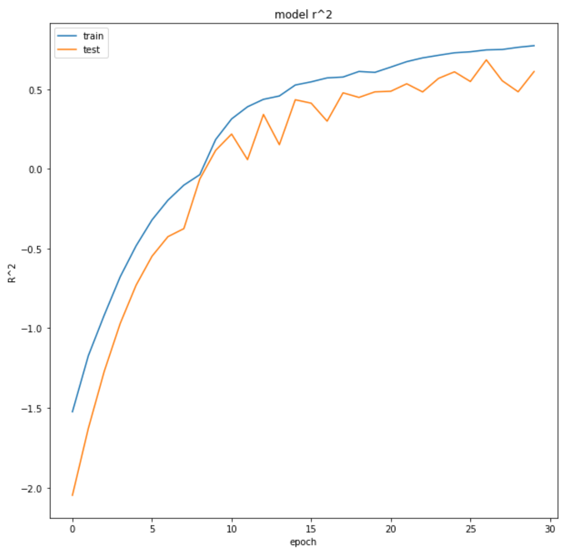


Fig.8.2(a) R^2 vs epochs

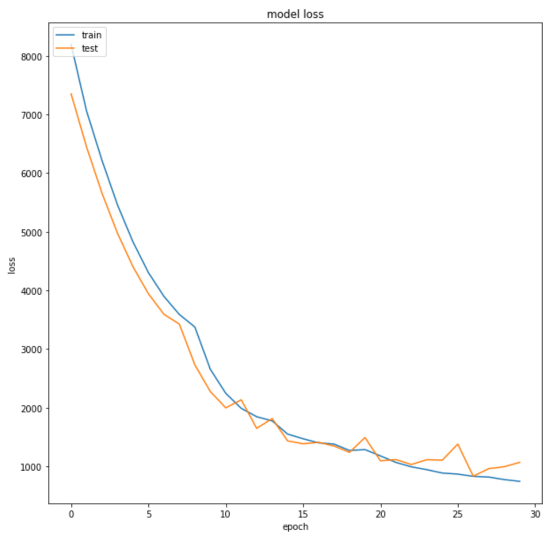


Fig.8.2(b) loss vs epochs

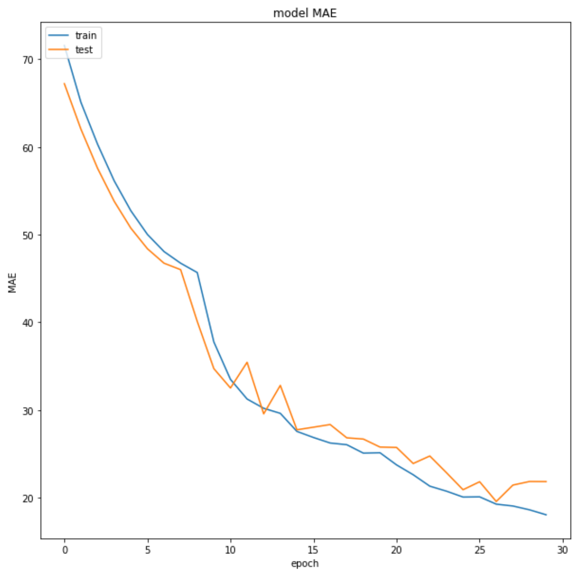


Fig.8.2(c) MAE vs epoch

The priority order or ranking with respect correlation was as follows:

[2 15 16 8 11 19 25 24 12 21 18 10 2 1]

Remaining sensors don't show correlation to the predicted RUL individually hence, some sensor indexes have been omitted as they show no correlation individually whatsoever

If given a chance by for the servicing of two variables, 4 and 15 should be increased to increase RUL as prescribed by the correlation parameter.

Mutual Information Score:

Individual score of each variable

1	Id	0.0287605
2	Setting1	0.00448337
3	Setting2	5.78469e-17
4	Setting3	0.0417839
5	T24	5.78469e-17
6	T30	0.0449987
7	T50	0.103286
8	P2	0.109402
9	P15	5.78469e-17
10	P30	0.000761651
11	Nf	0.0575341
12	Nc	0.0229785
13	Epr	0.118172
14	Ps30	5.78469e-17
15	Phi	0.034829
16	NRf	0.0533007
17	NRc	0.0233045
18	BPR	0.116586
19	farB	0.0918865
20	htBleed	5.78469e-17
21	Nf_dmd	0.00933548
22	PCNfR_dmd	5.78469e-17
23	W31	5.78469e-17
24	W32	0.0259263
25	T24	0.112670

Table.8.2(b) confidence values with sensor and setting index

8.3 GRU FD001:

Parameters of the model:

Parameter	Value
Loss	795.1201
MAE	17.0231
R^2	0.7605

Fig 8.3(a)

The following are the iteration graphs or the change observed per epoch:

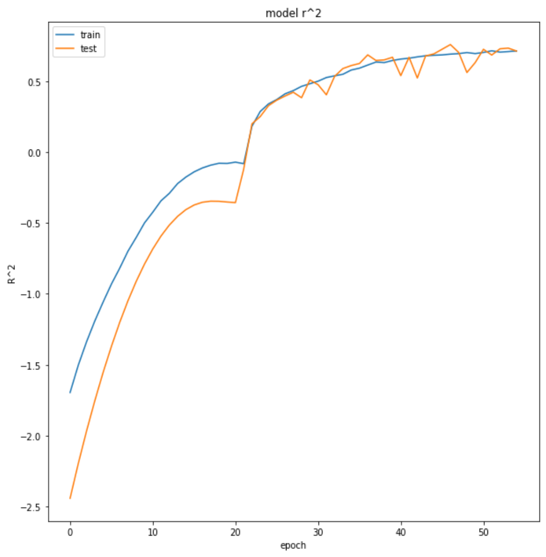


Fig.8.3(a) R^2 vs epochs

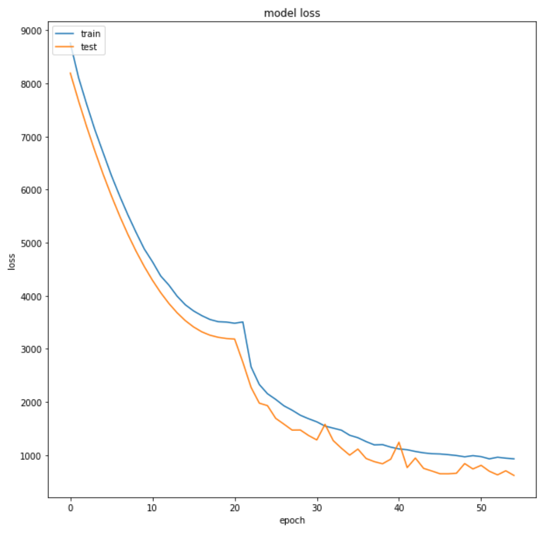


Fig.8.3(b) loss vs epochs

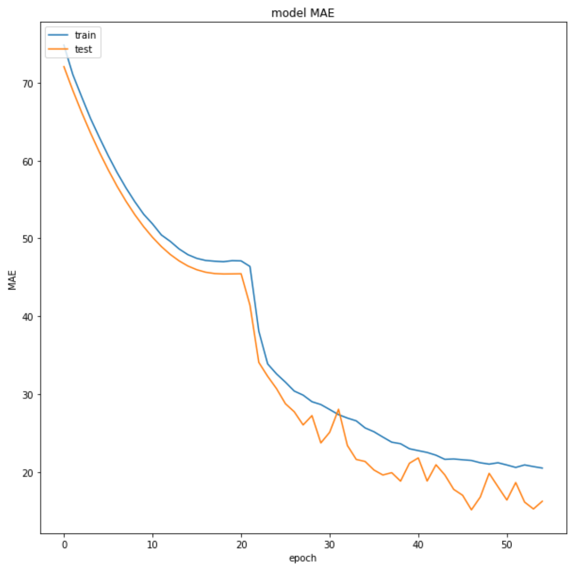


Fig.8.3(c) MAE vs epochs

The priority order or ranking with respect correlation was as follows:

[4 15 16 11 25 24 8 19 6 21 12 17 7 13 18 10 1]

Remaining sensors don't show correlation to the predicted RUL individually hence, some sensor indexes have been omitted as they show no correlation individually whatsoever

If given a chance by for the servicing of two variables, 4 and 15 should be increased to increase RUL as prescribed by the correlation parameter.

Mutual Information Score:

Individual score of each variable

1	Id	0.0287605
2	Setting1	0.00448337
3	Setting2	5.78469e-17
4	Setting3	0.0417839
5	T24	5.78469e-17
6	T30	0.0449987
7	T50	0.103286
8	P2	0.109402
9	P15	5.78469e-17

10	P30	0.000761651
11	Nf	0.0575341
12	Nc	0.0229785
13	Epr	0.118172
14	Ps30	5.78469e-17
15	Phi	0.034829
16	NRf	0.0533007
17	NRc	0.0233045
18	BPR	0.116586
19	farB	0.0918865
20	htBleed	5.78469e-17
21	Nf_dmd	0.00933548
22	PCNfR_dmd	5.78469e-17
23	W31	5.78469e-17
24	W32	0.0259263
25	T24	0.112670

Table.8.3(b) confidence values with sensor and setting index

9. SUMMARY

After running three different models and predicting the RUL of the engine, it was found that Gated recurrent unit model was more accurate and had very less as compared to other 2 models used for the objective of the project.

Hence, after predicting the RUL, the relation between the Predicted RUL and the test data input was calculated trying to find the relation between the two. If the correlation were positive, the RUL will increase with the increase in the value and it the correlation was negative it would decrease with increase in the value and vice versa. First only individual corelation was calculated to first aid the part if given a chance of tending to a single component. Later different permutations of the sensors were calculated if the management is given a chance to tend to only 2 parts.

Later using the mutual information scorer which penalises each value for less mutual information and rewards for more, a confidence value was calculated for each sensor to rank and tend to according to it as well.

Hence, the management will be given a choice depending upon the algorithm chosen to solve the particular issue, but issue will be fixed nonetheless. The following concludes my project.

10.REFERENCES:

A Directed Acyclic Graph Network Combined with CNN and LSTM for Remaining Useful Life Prediction by Jialin li, Xueyi li and David he

Remaining Useful Life Estimation of Engineered Systems using vanilla LSTM Neural Networks Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, Yingqi Liu

Deep learning models for predictive maintenance: a survey, comparison, challenges and prospect by Oscar Serradilla, Ekhi Zugasti, Urko Zurutuza

A Survey of Predictive Maintenance: Systems, Purposes and Approaches by Yongyi Ran, Xin Zhou, Pengfeng Lin, Yonggang Wen, Ruilong Deng

Advancing from Predictive Maintenance to Intelligent Maintenance with AI and IIoT by Haining Zheng, Antonio R. Paiva, Chris S. Gurciullo

Current-based mechanical fault detection for direct-drive wind turbines via synchronous sampling and impulse detection by X. Gong and W. Qiao

The analytic hierarchy process applied to maintenance strategy selection by M. Bevilacqua and M. Braglia

"Multi-level predictive maintenance for multi-component systems," Reliability engineering & system safety by K.-A. Nguyen, P. Do, and A. Grall

A new paradigm of cloud based predictive maintenance for intelligent manufacturing," Journal of Intelligent Manufacturing by J. Wang, L. Zhang, L. Duan, and R. X. Gao

Charles M. Able, Alan H. Baydush, Callistus Nguyen, Jacob Gersh, Alois Ndlovu, Igor Rebo, Jeremy Booth, Mario Perez, Benjamin Sintay, and Michael T. Munley. 2016. A model for preemptive maintenance of medical linear accelerators-predictive maintenance

Khalid F Al-Raheem and Waleed Abdul-Karem. 2011. Rolling bearing fault diagnostics using artificial neural networks based on Laplace wavelet analysis. International Journal of Engineering, Science and Technology 2, 6