PREDICTIVE MAINTENANCE AND PERFORMANCE OPTIMIZATION FOR JET ENGINES BASED ON ROLLS-ROYCE ENGINE MANUFACTURER AND SERVICES WITHIN THE AEROSPACE SECTOR

SITI SYAHIRAH BINTI MOHD YUNUS

CHAPTER 4

UNIVERSITI TEKNOLOGI MALAYSIA

Chapter 4.

4.1 Introduction

4.2 EDA

CHAPTER 4

INITIAL RESULTS

Data Visualization

Data visualization is crucial as it shows the pattern and exact illustration in graph to aid with better understanding on each testing condition.

4.2 Data Extraction, EDA & Feature engineering

Turbofan Engines Lifetime showing maximum cycles time for each unit form engine 1-100. Based on the plotted chart, it is clearly shown the performance readings are varies.

```
#Maximum time cycle of each unit
max_time_cycles=dataframe[index_names].groupby('unit_nr').max()


plt.figure(figsize=(20,50))
plt.title('Turbofan Engines LifeTime',fontweight='bold',size=30)
ax=max_time_cycles['time_cycles'].plot(kind='barh',width=0.8, stacked=True,align='center')

plt.ylabel('Time cycle',fontweight='bold',size=20)
plt.yticks(size=15)
plt.xlabel('unit',fontweight='bold',size=20)
plt.xticks(size=15)
plt.grid(False)
plt.tight_layout()

#Show the plot
plt.show()
```

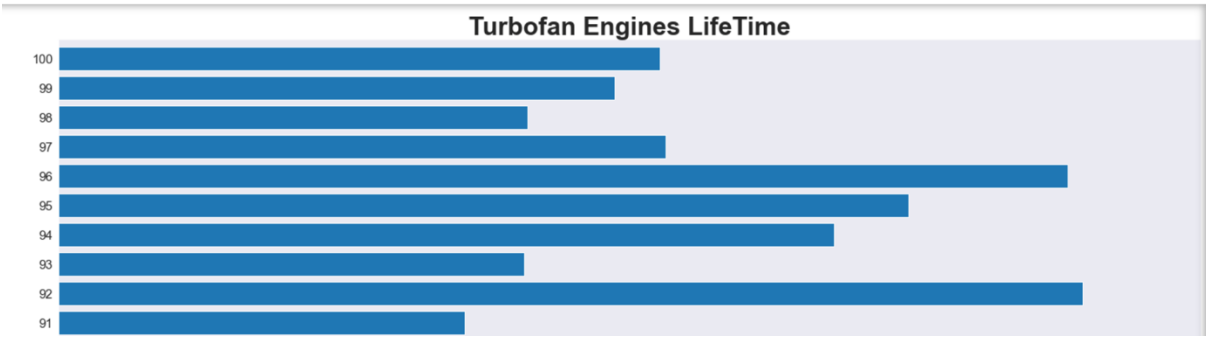


Figure 4.1 The chart showing example reading for engine 91-100.

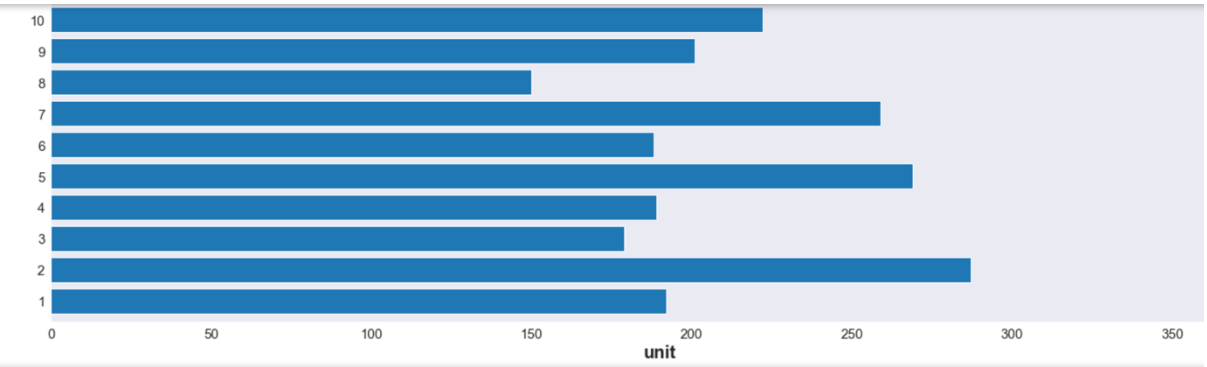


Figure 4.2 The chart showing example reading for engine 1-10.

The distribution of maximum time cycles are shown below indicate the engine can achieve the maximum cycles time is the range 190 and 210 before the HPC Failure. The RUL is skewed to the left side showing the greatest engine performance is at first 50% or up to second quartiles and the trend goes down going through the next third and fourth quartile.

```
#Distribution of maximum time cycles
sns.displot(max_time_cycles['time_cycles'],kde=True,bins=20,height=6,aspect=2)
plt.xlabel('max time cycle')
```

```
Text(0.5, 9.444444444444459, 'max time cycle')
```

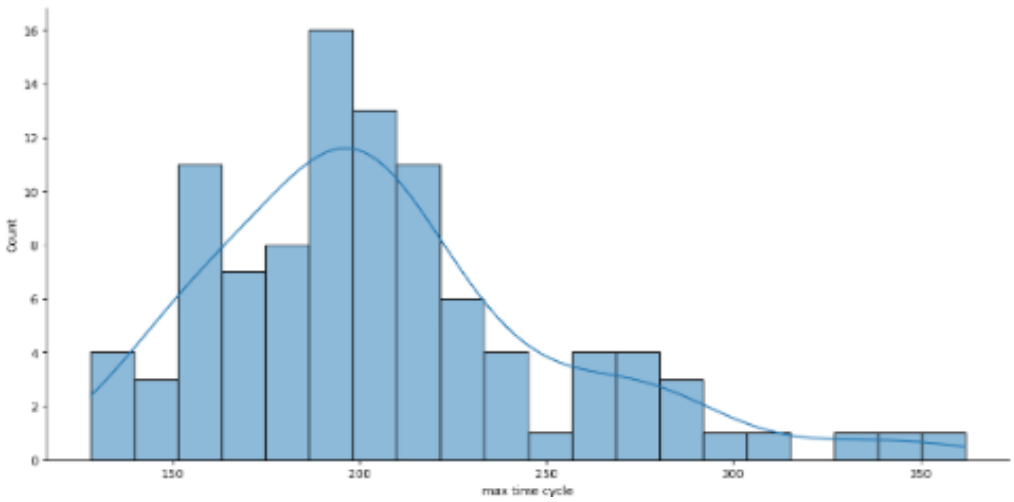


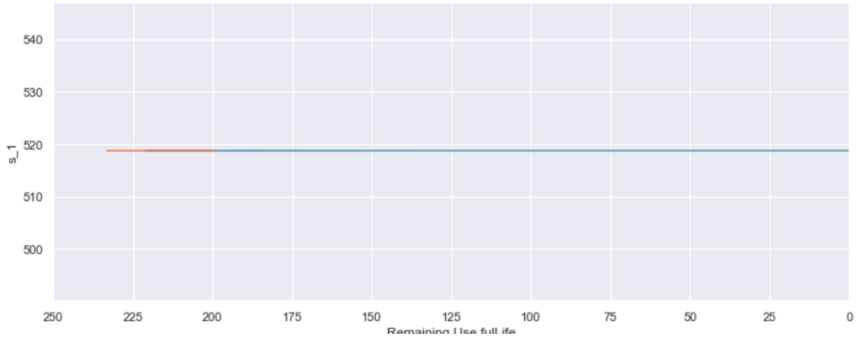
Figure 4.3 : Distribution of maximum time cycles

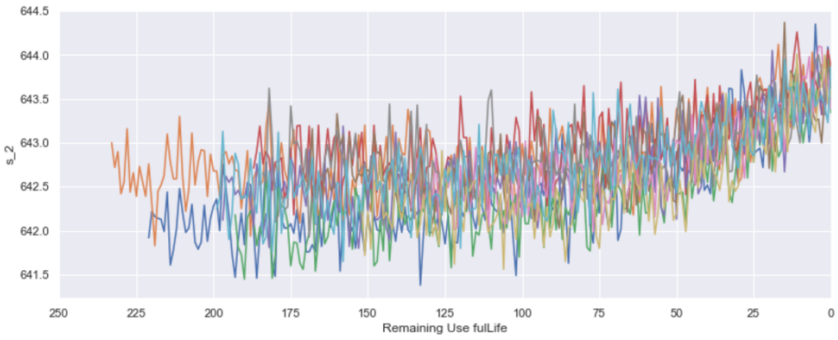
Plotting the Sensor Values vs Engine RUL. Sensor s_1 to s_21 all are having different reading to show the correlation between each sensor and the function to determine the RUL. Based on the observation, sensor 1,5,6,10,16,18 and 19 have remained constant throughout the

cycles test hence this means these sensors have no useful information and not have significant function to the RUL. Figures 4.1-4.6 Showing sensors graph plotted against engine RUL.

```
def plot_sensor(sensor_name):
    plt.figure(figsize=(13,5))
    for i in train['unit_nr'].unique():
        if (i % 10 == 0):  # only plot every 10th unit_nr
            plt.plot('RUL', sensor_name,
                     data=train[train['unit_nr']==i])
    plt.xlim(250, 0)  # reverse the x-axis so RUL counts down to zero
    plt.xticks(np.arange(0, 275, 25))
    plt.ylabel(sensor_name)
    plt.xlabel('Remaining Use fulLife')
    plt.show()

for sensor_name in sensor_names:
    plot_sensor(sensor_name)
```





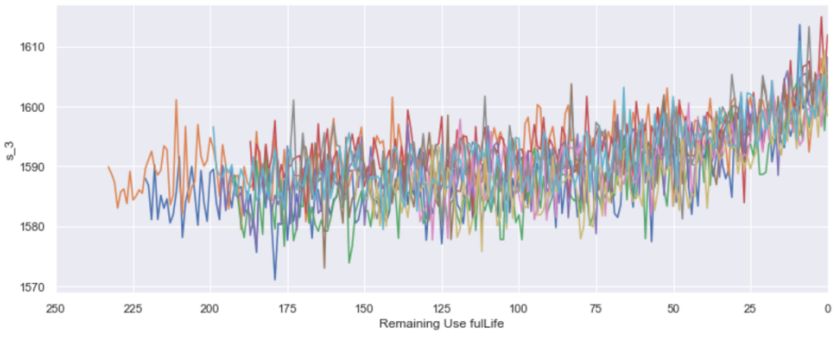


Figure 4.5: Sensor values vs RUL s_1 s_3

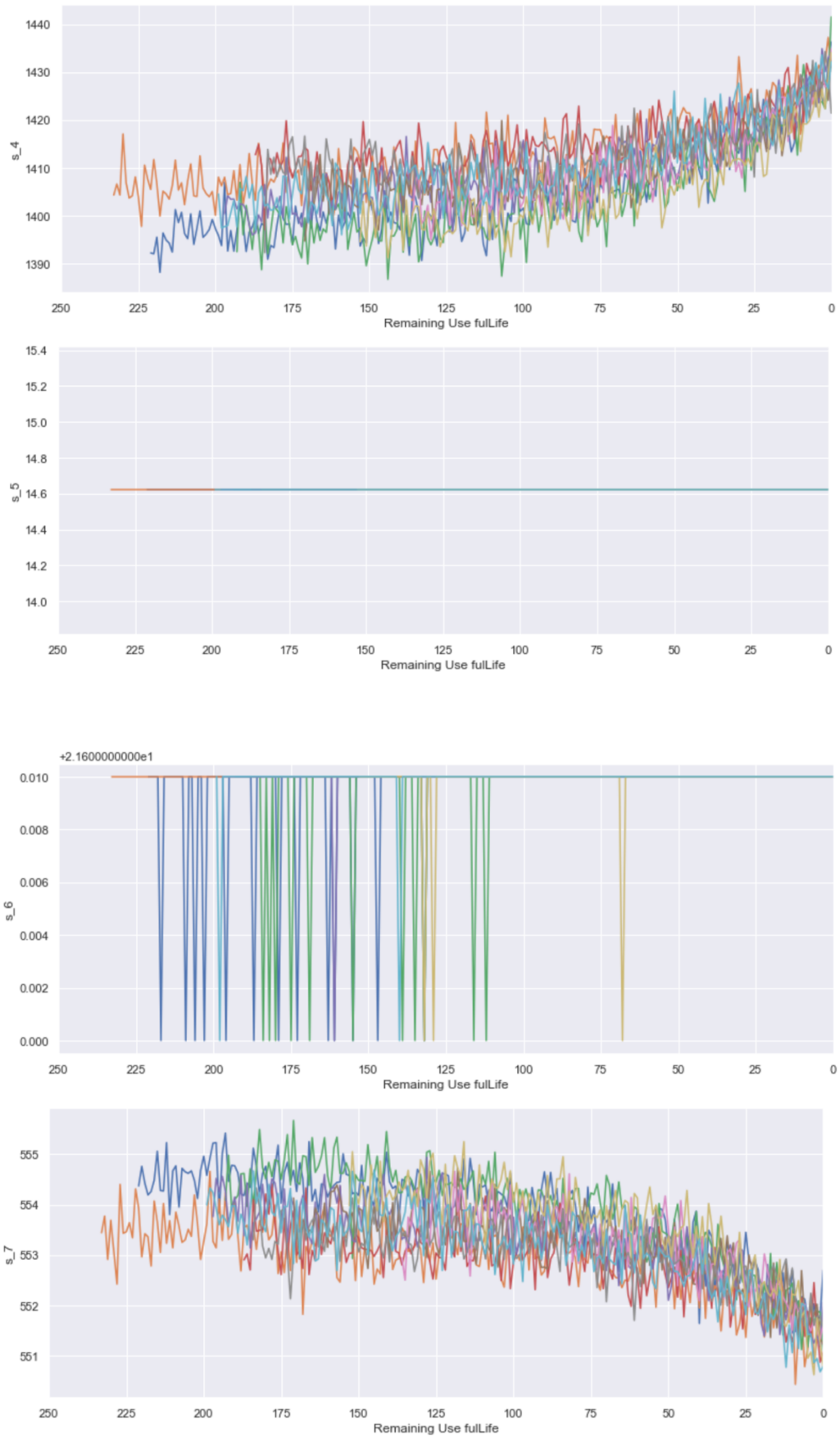


Figure 4.2: Sensor values vs RUL s_4 till s_7

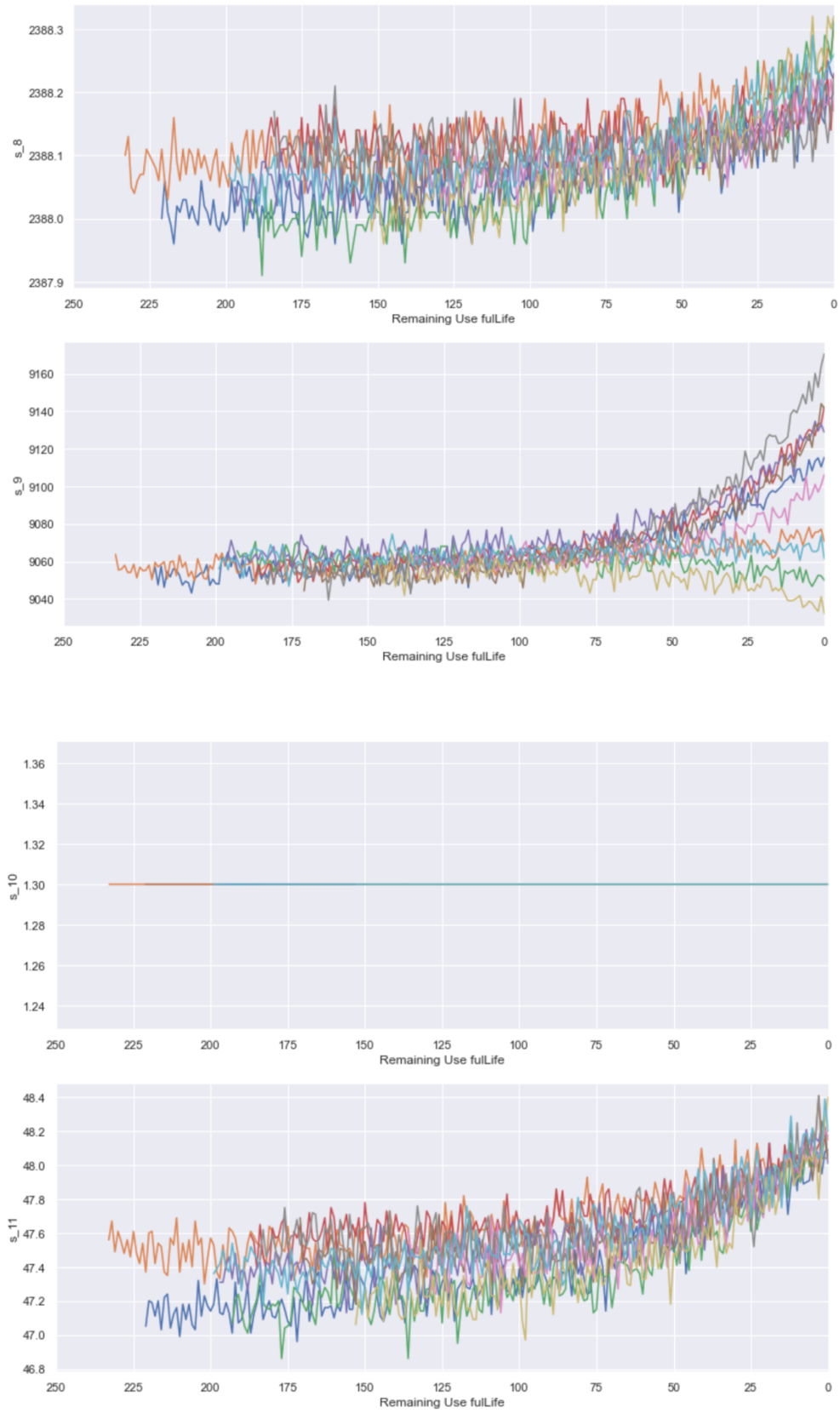


Figure 4.3: Sensor values vs RUL s_8 till s_11

Figure 4.4: Sensor values vs RUL s_12 till s_15

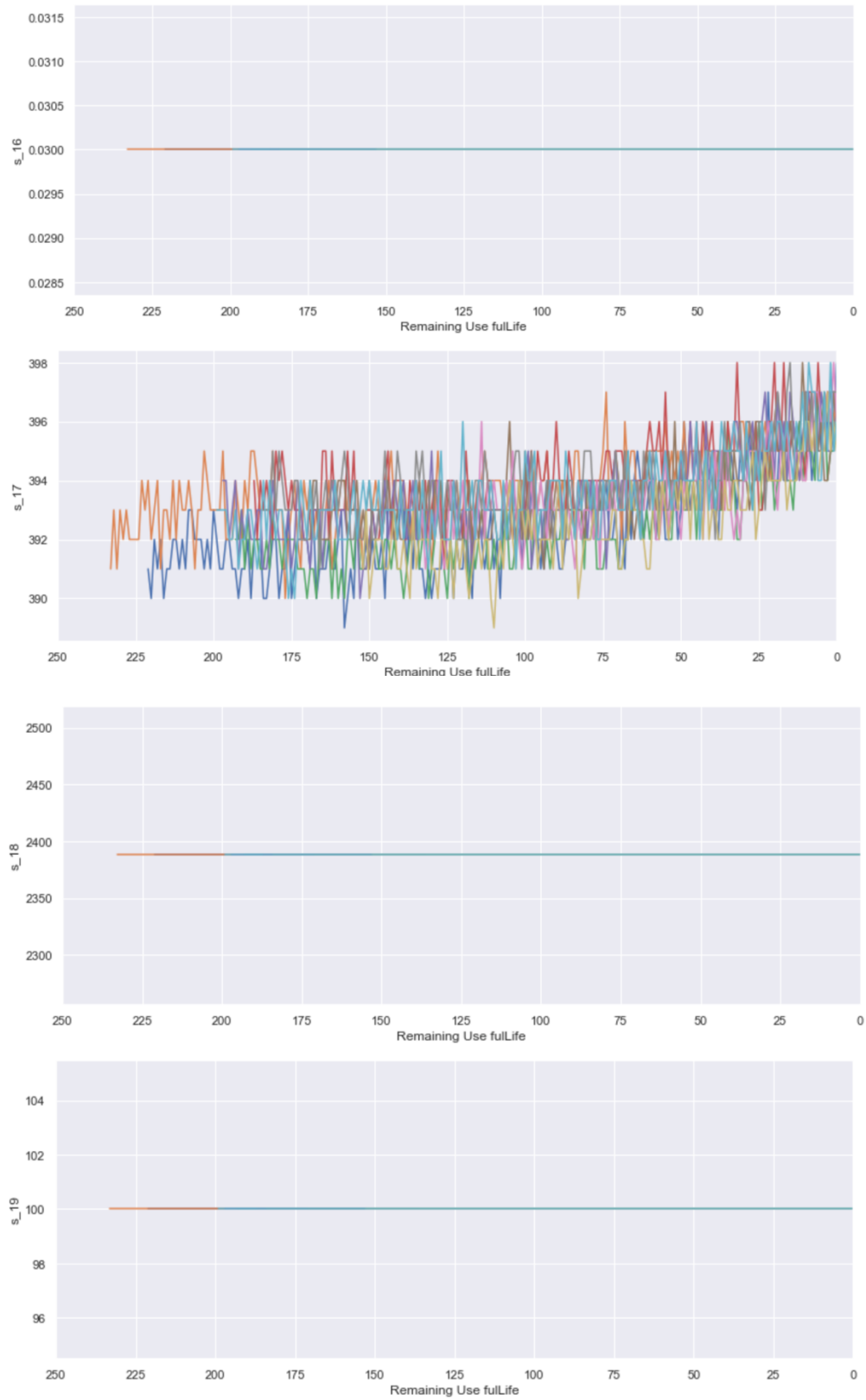


Figure 4.5: Sensor values vs RUL s_16 till s_19

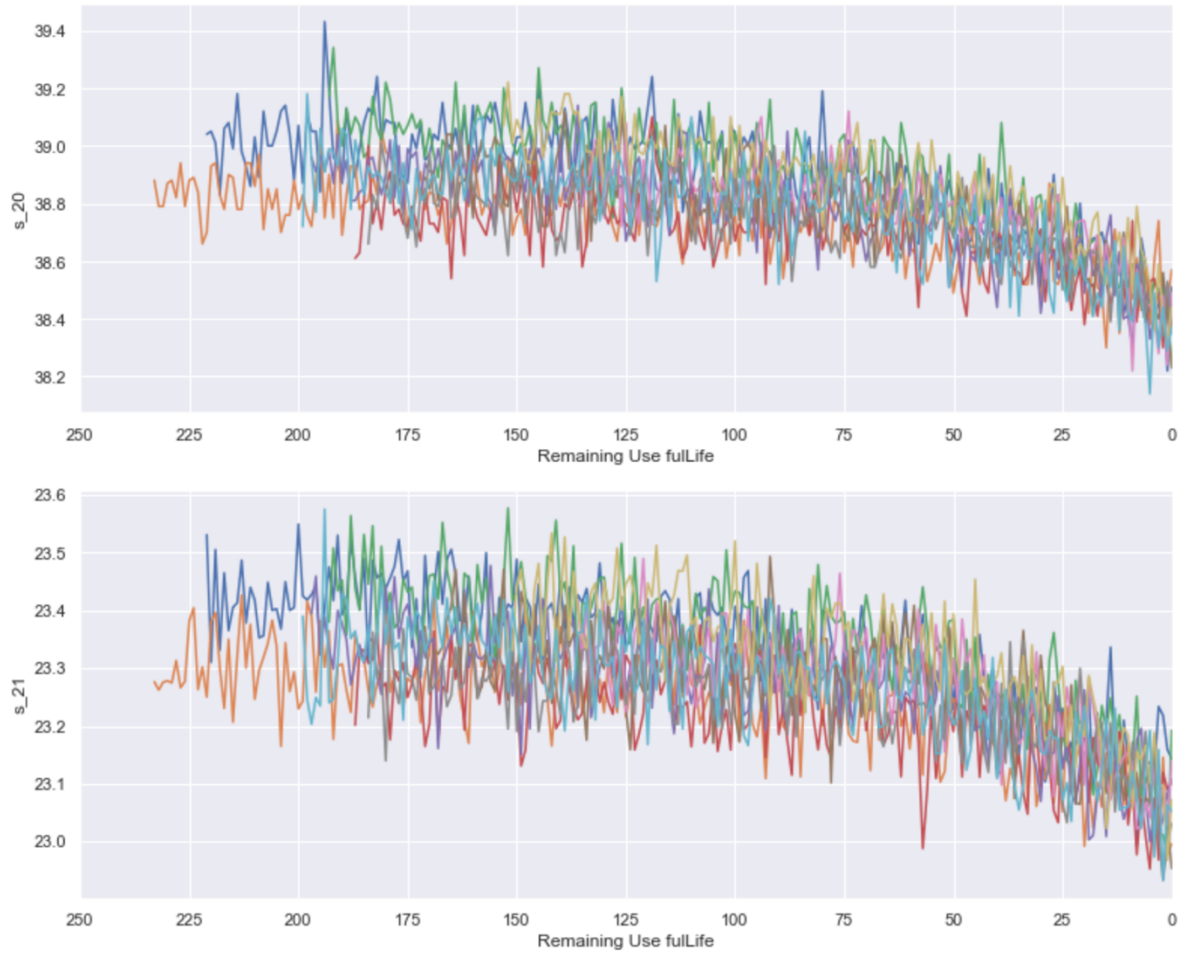


Figure 4.6: Sensor values vs RUL s_20 and s_21

The EDA sensor dictionary 1,5,6,10,16,18 and 19 has remained constant throughout the cycles test hence this means these sensors have no useful information and does not have significant function to the RUL. The figure 4.7 below match each sensor with its significant function.

```
dict_list=[ "(Fan inlet temperature) (◦R)",
"(LPC outlet temperature) (◦R)",
"(HPC outlet temperature) (◦R)",
"(LPT outlet temperature) (◦R)",
"(Fan inlet Pressure) (psia)",
"(bypass-duct pressure) (psia)",
"(HPC outlet pressure) (psia)",
"(Physical fan speed) (rpm)",
"(Physical core speed) (rpm)",
"(Engine pressure ratio(P50/P2)",
"(HPC outlet Static pressure) (psia)",
"(Ratio of fuel flow to Ps30) (pps/psia)",
"(Corrected fan speed) (rpm)",
"(Corrected core speed) (rpm)",
"(Bypass Ratio) ",
"(Burner fuel-air ratio)",
"(Bleed Enthalpy)",
"(Required fan speed)",
"(Required fan conversion speed)",
"(High-pressure turbines Cool air flow)",
"(Low-pressure turbines Cool air flow)" ]
i=1
for x in dict_list :
    Sensor_dictionary['s_'+str(i)]=x
    i+=1
Sensor_dictionary
```

```
{'s_1': '(Fan inlet temperature) (◦R)',
 's_2': '(LPC outlet temperature) (◦R)',
 's_3': '(HPC outlet temperature) (◦R)',
 's_4': '(LPT outlet temperature) (◦R)',
 's_5': '(Fan inlet Pressure) (psia)',
 's_6': '(bypass-duct pressure) (psia)',
 's_7': '(HPC outlet pressure) (psia)',
 's_8': '(Physical fan speed) (rpm)',
 's_9': '(Physical core speed) (rpm)',
 's_10': '(Engine pressure ratio(P50/P2)',
 's_11': '(HPC outlet Static pressure) (psia)',
 's_12': '(Ratio of fuel flow to Ps30) (pps/psia)',
 's_13': '(Corrected fan speed) (rpm)',
 's_14': '(Corrected core speed) (rpm)',
 's_15': '(Bypass Ratio) ',
 's_16': '(Burner fuel-air ratio)',
 's_17': '(Bleed Enthalpy)',
 's_18': '(Required fan speed)',
 's_19': '(Required fan conversion speed)',
 's_20': '(High-pressure turbines Cool air flow)',
 's_21': '(Low-pressure turbines Cool air flow)'}
```

Figure 4.7: Sensor dictionaries

In the context of a jet engine, a **heatmap** is used to visualize the sensor training vs RUL where it showing the correlation between these two or even its components. Heatmap act as a visual way to show the relationship between sensor data (such as vibration, pressure, temperature etc.) with the engine performance degradation over period of time. This method is being used to highlight which variables are more significant affecting the RUL.

Brighter colours showing high intensity regions indicate strong correlation between sensor readings and RUL as for example, the higher temperatures strongly indicate faster degradation hence the RUL being reduced.

Darker colours act as indicator of low intensity regions hence the correlation between sensors reading and RUL is weaker, meaning the sensors are less significant in RUL prediction.

The figure showing colour patterns and the pattern can indicate critical points as for example when engine is getting to the end of its life, the sensors that shows vibration will give higher reading and on heatmap it will show brighter colour.

```
sns.heatmap(dftrain.corr(),annot=True,cmap='RdYlGn')
fig=plt.gcf()
fig.set_size_inches(20,20)
plt.show()
```

Figure 4.8 Train the sensor data to display Heatmap

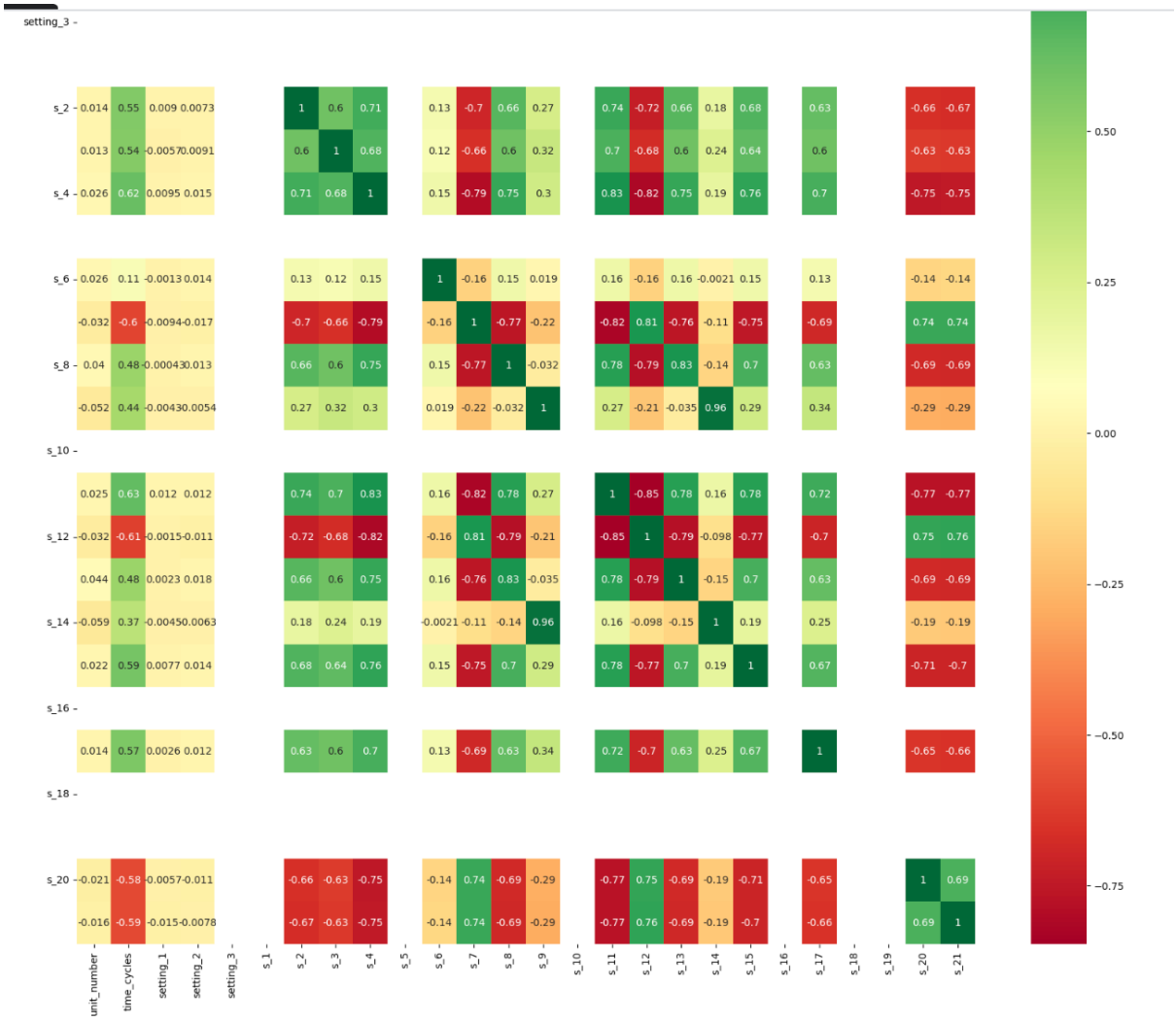


Figure 4.9 Heatmap indicate the correlation between sensors & RUL

Based on the input from heatmap, it is beneficial to identify which sensors that indicate which variables need to be focus onto and which that need to be eliminate. This is call as dimensionality reduction to reduce unnecessary features to reduce the overfitting.

By selecting highly correlated features, this can helps to reduce overfitting and avoid the wastage for the engine performance. Training relevant features only are so important to ensure the credibility of RUL. The details shown as per Figure 4.10.

```
cor=train.corr()
#cor_target = abs(cor["RUL"])
#Selecting highly correlated features
train_relevant_features = cor[abs(cor['RUL'])>=0.5]

train_relevant_features['RUL']
```

```
time_cycles   -0.736241
s_2           -0.606484
s_3           -0.584520
s_4           -0.678948
s_7            0.657223
s_8           -0.563968
s_11          -0.696228
s_12           0.671983
s_13          -0.562569
s_15          -0.642667
s_17          -0.606154
s_20           0.629428
s_21           0.635662
RUL            1.000000
Name: RUL, dtype: float64
```

```
list_relevant_features=train_relevant_features.index
list_relevant_features=list_relevant_features[1:]
list_relevant_features
```

```
Index(['s_2', 's_3', 's_4', 's_7', 's_8', 's_11', 's_12', 's_13', 's_15',
       's_17', 's_20', 's_21', 'RUL'],
      dtype='object')
```

Figure 4.10 Select and train relevant sensors features

Out if 21 sensors, only 12 sensors contributing relevant aspects to the RUL hence only 12 sensors are train to see the correlation with RUL

```
# Above list contains important features have correlation of magnitude greater and equal to 0.5 with our target variable RUL.
```

```
# Now we will keep only these important features in both train & test dataset.
train=train[list_relevant_features]
```

```
train.head(5)
```

	s_2	s_3	s_4	s_7	s_8	s_11	s_12	s_13	s_15	s_17	s_20	s_21	RUL
0	641.82	1589.70	1400.60	554.36	2388.06	47.47	521.66	2388.02	8.4195	392	39.06	23.4190	191
1	642.15	1591.82	1403.14	553.75	2388.04	47.49	522.28	2388.07	8.4318	392	39.00	23.4236	190
2	642.35	1587.99	1404.20	554.26	2388.08	47.27	522.42	2388.03	8.4178	390	38.95	23.3442	189
3	642.35	1582.79	1401.87	554.45	2388.11	47.13	522.86	2388.08	8.3682	392	38.88	23.3739	188
4	642.37	1582.85	1406.22	554.00	2388.06	47.28	522.19	2388.04	8.4294	393	38.90	23.4044	187

Figure 4.11 Train the relevant sensors features

Among all sensors, selected features are as below dictionaries. Only these variables are being used to test and train for modelling purpose.

```
[['cycle',
  '(LPC outlet temperature) (◦R)',
  '(HPC outlet temperature) (◦R)',
  '(LPT outlet temperature) (◦R)',
  '(bypass-duct pressure) (psia)',
  '(HPC outlet pressure) (psia)',
  '(Physical fan speed) (rpm)',
  '(Physical core speed) (rpm)',
  '(HPC outlet Static pressure) (psia)',
  '(Ratio of fuel flow to Ps30) (pps/psia)',
  '(Corrected fan speed) (rpm)',
  '(Bypass Ratio) ',
  '(Bleed Enthalpy)',
  '(High-pressure turbines Cool air flow)',
  '(Low-pressure turbines Cool air flow)']]
```

Figure 4.12 Sensors variables used in modelling

4.3 Initial Insights

In this chapter, only 2 machine learning model are being used to do initial finding for turbofan engine RUL. The model chosen are the simplest machine learning model which are Linear regression and Support Vector Machine (SVM). **Table 4.1** explain the comparison aspects.

Aspect	Linear Regression	Support Vector Regression (SVR)
Data Relationship	Assumes a **linear relationship** between sensor features and RUL.	Captures **non-linear relationships** in sensor data using kernels.
Robustness to Outliers	**Sensitive** to outliers; large deviations can distort predictions.	**Robust** to outliers; uses a margin of tolerance (ε) to ignore noise.
Feature Complexity	Requires explicit **feature engineering** for non-linear trends (e.g., adding polynomial terms).	Automatically handles **complex patterns** through kernel transformations.
Interpretability	**Highly interpretable**; easy to explain relationships between features and RUL.	**Less interpretable**; kernel transformations obscure feature contributions.
Scalability	**Efficient**; suitable for large datasets with many features.	Computationally **expensive** for large datasets, especially with RBF kernels.
Performance on Non-linear Data	Performs poorly if the sensor data has **non-linear degradation patterns**.	Performs well on **non-linear degradation trends** in sensor data.
Ease of Implementation	Simple and fast to implement as a baseline model.	More complex; requires tuning of parameters (C, ε, kernel).
Training Time	**Very fast**, even for large datasets.	**Slower**, especially for high-dimensional data or large datasets.
Example Use Case	Effective when RUL decreases **linearly** with operating cycles or sensor degradation.	Effective for **complex, non-linear degradation** patterns over time.

Table 4.1 Linear regression and SVR comparison aspects

Model 1: Linear Regression

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train1 = sc.fit_transform(X_train)
X_test1 = sc.transform(X_test)
```

```
# create and fit model
lm = LinearRegression()
lm.fit(X_train1, y_train)

# predict and evaluate
y_hat_train1 = lm.predict(X_train1)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train1,'train')

y_hat_test1 = lm.predict(X_test1)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test1,'test')
```

```
train set RMSE:22.734164950962253, R2:0.7023848970100307
test set RMSE:22.914265328584634, R2:0.6959448729951723
```

```
# Make Dataframe which will contain results of all applied Model
Results=pd.DataFrame({'Model':['LR'],'RMSE-Train':[RMSE_Train],'R2-Train':[R2_Train],'RMSE-Test':[RMSE_Test],
Results
```

	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	22.734165	0.702385	22.914265	0.695945

Figure 4.13 Linear Regression

After training, actual versus predicted graph are plotted as a comparison to see if Linear regression model are capable to show the predicted trend (red graph) related to the actual RUL (blue graph)

```
# Plot Actual Vs Predicted RUL for Train Data
#c = [i for i in range(1,81,1)]
fig = plt.figure();
plt.figure(figsize=[20,12])
plt.plot(y_train,color="blue", linewidth=2.5, linestyle="-",label="Actual")
plt.plot(y_hat_train1,color="red",  linewidth=2.5, linestyle="-",label="Predicted")
fig.suptitle('Actual and Predicted', fontsize=20)            # Plot heading
#plt.xlabel('Index', fontsize=18)                            # X-label
plt.ylabel('RUL', fontsize=16)                               # Y-label
plt.legend()
plt.title("Actual RUL Vs Predicted RUL for Train Data")
```

```
Text(0.5, 1.0, 'Actual RUL Vs Predicted RUL for Train Data')
<Figure size 432x288 with 0 Axes>
```

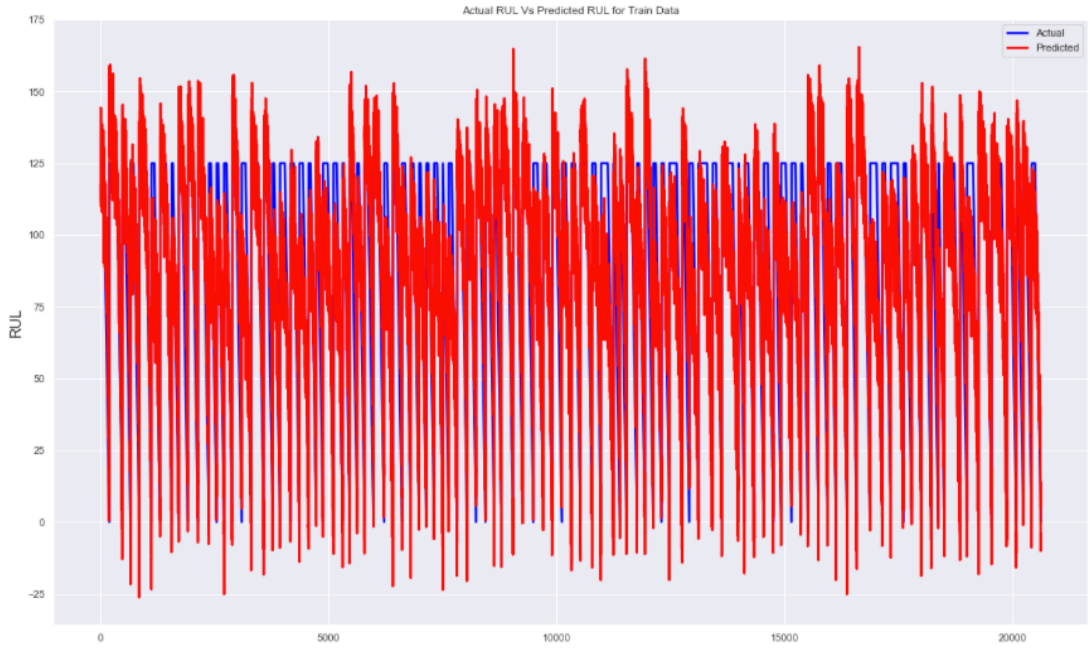


Figure 4.14 Actual vs predicted RUL for train data

The test data showing RMSE = 22.9 while R2 score = 0.695. Other machine learning model need to be run, compare and evaluate to see which machine learning model have a better RUL prediction.

```
# Making predictions using test set
y_hat_test1 = lm.predict(X_test1)
evaluate(y_test, y_hat_test1)
```

```
test set RMSE:22.914265328584634, R2:0.6959448729951723
(22.914265328584634, 0.6959448729951723)
```

```
# Plot Actual Vs Predicted RUL for Test Data
#c = [i for i in range(1,81,1)]
fig = plt.figure();
plt.figure(figsize=[20,12])
plt.plot(y_test,color="blue", linewidth=2.5, linestyle="-",label="Actual")
plt.plot(y_hat_test1,color="red",  linewidth=2.5, linestyle="-",label="Predicted")
fig.suptitle('Actual and Predicted', fontsize=20)              # Plot heading
#plt.xlabel('Index', fontsize=18)                               # X-label
plt.ylabel('RUL', fontsize=16)                                 # Y-label
plt.legend()
plt.title("Actual RUL Vs Predicted RUL for Test Data")
```

```
Text(0.5, 1.0, 'Actual RUL Vs Predicted RUL for Test Data')
<Figure size 432x288 with 0 Axes>
```

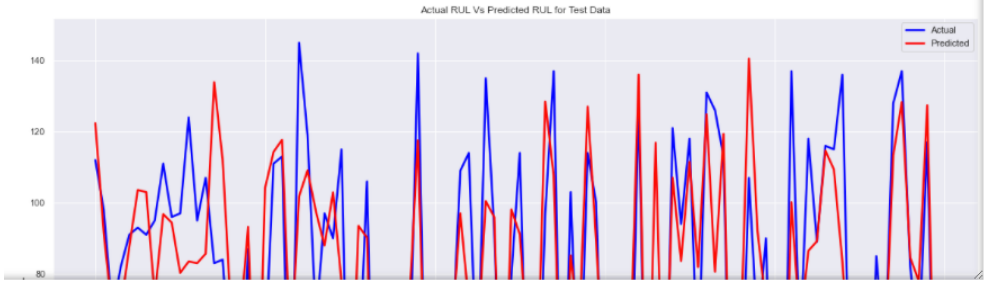


Figure 4.15 Actual vs predicted RUL for test data

Model 2 : Applying Support Vector Regression (SVR)

The RMSE Result for test and train model has been increased after applying SVR model.

```
from sklearn.svr import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train1, y_train)

# predict and evaluate
y_hat_train1 = regressor.predict(X_train1)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train1)

y_hat_test1 = regressor.predict(X_test1)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test1)
```

```
test set RMSE:21.42864070146049, R2:0.7355849328257191
test set RMSE:21.828222300276334, R2:0.7240838052912153
```

```
# Make Dataframe which will contain results of all applied Model
Results=Results.append(pd.DataFrame({'Model':['SVM'],'RMSE-Train':[RMSE_Train],'R2-Train':[R2_Train],'RMSE-T
Results
```

	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	22.734165	0.702385	22.914265	0.695945
1	SVM	21.428641	0.735585	21.828222	0.724084

Figure 4.16 Applying SVR to the dataset

All 21 sensors has been through data cleaning, training and test before producing the heatmap. The heatmap indicate the sensor correlation with the turbofan jet engine RUL. Then only sensor with the strong correlation with RUL are being used to test and train under 2 machine learning models.

Since only 2 models are used for feature engineering which are Linear Regression and Support Vector Regression and run only once, further repetition is advisable to do. Although the result is not far from the actual RUL recorded at the beginning of this chapter, there are still 4 models that will be run in near future so that better comparison can be made.