Engine Prognosis Modelling Simulation

Data Science Master's
KSCHOOL Final Project 2021-2022

Lorenzo Castro Ventas

Index

1. Topic Introduction

1.1. Aerospace Maintenance

Aerospace maintenance represents one of the most important procedures performed by aircraft operators. It is not only directly related to safety in-flight, but also affects the airline economy. [1]

As stated in picture below, maintenance activities may represent up to 10% of overall airline costs. Direct and indirect costs associated with aircraft maintenance are:

- Materials and spares from damaged components
- Workers with qualified technical knowledge in aircraft/engine systems architecture
- Aircraft's hangar and industrial facilities rented for these activities
- Aircraft stopped to be performed maintenance procedures causes the inactivity of itself, with the loss of profit associate

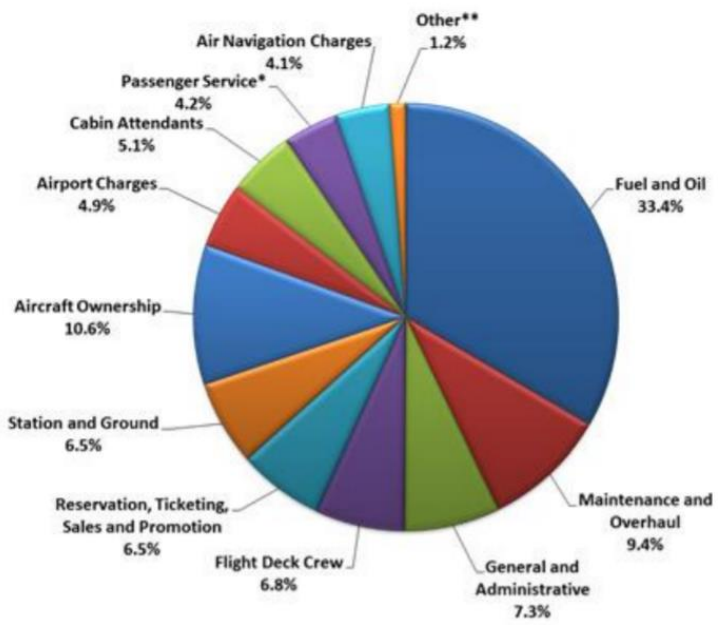


Figure 1 General Airline's costs

Along the aerospace maintenance discipline, it has been since always separated the airframe and the engine maintenance, due to the completely different technology and systems involved. In this thesis, the engine will be the subject of study. Engine represents almost half of overall purchase cost of the vehicle. [2]

Composed by complex systems and components, with a closed-loop performance where interactivity between subsystems determine the correct behavior during flight accomplishment, specific maintenance activities, schedules and needs are set apart.

In the following section, maintenance activities importance will be confirmed with a brief explanation of potential hazards during engine performance.

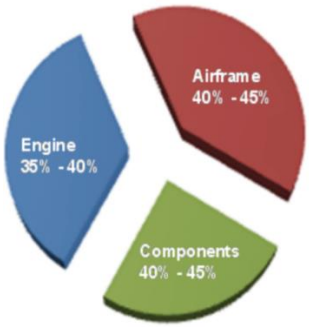


Figure 2 Maintenance expense distribution

1.2. Potential Risks during engine performance

Turbofan engines are composed systems which typical performance conditions are extremely severe (high temperatures and pressures, rotational speed of engine shaft causing strong vibrations, materials corrosion, potential external impacts...). Each flight represents many risks which may impact the correct performance of the engine and, hence, the safety of the flight potentially incurring in human lives. [3]

Figure 3 Flames in engine's exhaust during flight

Several components of these complex machines work constantly in these hard flight conditions. In order to assure continuous safety and allow the engine to perform efficiently (that means reducing fuel consumption, noise and emissions as well) the engine is put down to rigorous maintenance schedule, divided in 4 types depending on the frequency: [4]

- Line maintenance: after every flight visual inspection and superficial sensor measurement testing
- A check: every 8-10 weeks (mostly hydraulic and lubrication systems)
- C checks: every 18 months - 2 years, subsystem's calibration, and testing
- D checks: every 6 years the entire aircraft and engine are basically dismantled and put back together

However, these rigorous schedules not always assure that sometimes a fault appears between these moments. Every time a serious performance error arrives it affects not only the integrity of the overall engine and crew, but it disturbs the always tight schedule of the airline, which will have to move apart resources with the loss of money and customer confidence coming as a result. Due to that reason, it is being under development the advancement of predictive maintenance in this industry.

1.3. State of the Art for predictive maintenance in aviation

The goal of predictive maintenance is to prevent such unexpected equipment failures by continuously observing the status of the equipment and raising alerts well in advance. Hence, time between alert and failure can be used by experts to plan and perform maintenance and avoid any operational disturbance, that is, to know the remaining useful life of the engine. [5]

Several industry stakeholders (such as manufacturers and aircraft operators) have started deploying predictive algorithm to start digging in this valuable potential. It is more and more frequent to see departments of engineering analytics and data performance in these companies.

These actors have started a few years ago to use all the data retained along many years of activity.

Machine learning has developed a strong connection with this trend; however, it is not always easy or straightforward to perform effective predictive maintenance, due to some reasons: [6]

- Lack of predictive power in the data (possibly data does not contain relevant or adequate information about the task).
- Lack of annotated data (the target to be predicted is when the engine stops working and, luckily, this target is very strange to find out, due to the extreme low number of times an engine has failed during flight, the acquisition of this phenomena might be really expensive and consume many man-hours).
- Huge amounts of data (in real world scenario one would have to deal with even petabytes of data in order to extract useful knowledge about the domain)

These problems will not be covered in this project, as the baseline of the study is taken not from real operator's experience, but from real flight simulations.

2. Case of Study

2.1. NASA's Repository

As commented in previous chapter, predictive maintenance development algorithms is such a new phenomenon in aerospace industry, and relevant stakeholders are extremely strict with data privacy. Hence, as these developments work with large amount of operational information, normally companies are not quite comfortable sharing this data with public.

Due to this reason, in order to provide the academia with consistent (although not 100% real) operational information, realistic simulations are made using real-flight conditions and exceptionally sensible systems to provide datasets.

In this project, the **NASA Prognosis Excellence Center** is used.

Figure 4 NASA prognosis center webpage

This repository is a collection of datasets donated by various universities or agencies which exclusively focuses on prognosis databases, mostly going from nominal state to a failure state, It works since 2007 providing time to time with these publications and related research papers.

In this project, it was selected the Turbofan dataset, which provides necessary datasets to start developing predictive algorithms.

2.2. Dataset description and Thesis objective

Initial data that gives the shape to this project consists of 3 different sets of turbofan engines, each one containing a different number of engines.

Each set of engines as well contains three different types of files to work with.

1. Train dataset: These engine units are performed throughout specific number of cycles (flights) until they fail. This means that, at some point during their operational life, these engines find a fault which leads them after a specific number of cycles to the end of service. Each engine ends its operational life at a different moment.
2. Test dataset: similarly, to Train dataset, contains a number of engines working cycle after cycle. However, in this set, the operational life is not fully represented, and at a random cycle (different for each unit) these engines stop providing performance information, hence, it is not possible to know when they are about to fail.
3. RUL dataset: for each of the engine's units in the Test set, these last files provide the remaining number of cycles (Remaining Useful Life) of each one of them before they fail in service

The objective of this study is then clear, the aim is to use the Train set of engines (working since beginning till the end of life) to develop predictive algorithms to forecast the remaining number of cycles before fault of the test set and test the accuracy of the model with the RUL file.

In order to provide this project with more complexity, each one of the 3 sets of engines (Set 1, set 2 and set 3) have operational differences.

1. First set provides engines working at the same flight condition (Sea Level) and only 1 type of fault during performance may happen (High Pressure Compressor failure)
2. Second set provides same component fault as well, but this time 6 different flight conditions are occurring to these engines
3. Third set provides same operational conditions as First one (Sea level flight), but now 2 different types if faults may occur in this scenario (High Pressure Compressor fault or Fan failure). This set introduces additionally a classification task, to find out whether these engines fail under one or another fault class. [5]

```
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)
```

Figure 5 Dataset's characteristics of engines performance

2.3. Features interpretation

Nowadays, turbofan engines allow to measure efficiently the performance conditions thanks to multiple sensors located in different stages of the machine. Sensors can measure temperature, pressure, rotational speed, material deformation...

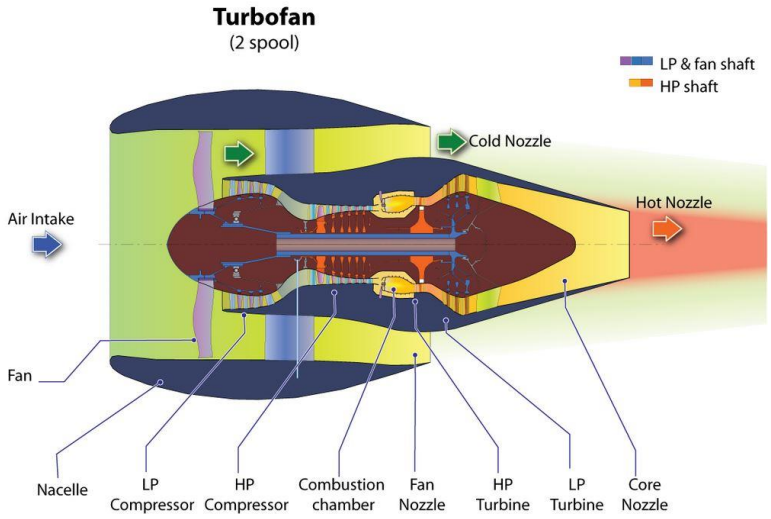


Figure 6 Turbofan stages scheme

In order to study the performance of these sets of engines. The datasets provide zip-compressed files with 26 columns of numbers. Each row of numbers is a snapshot of data taken during a single operational cycle (a flight), and each column is a different variable, corresponding:

- First column: unit engine number
- Second column: cycle (time variable)
- Columns 3-5: operational settings (1,2,3)
- Columns 6-26: sensor measurements

This data distribution provides the following aspect, using as an example the first engine's fleet train set:

```
df_train_1 = pd.read_csv('/home/dsc/Downloads/CMAPSSData/train_FD001.txt', sep=" ", header=None)
df_train_1
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	21.61	554.36	2388.06	9046.19	1.3	47.47	521.66	2388.02	8138.62	8.4195
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	21.61	553.75	2388.04	9044.07	1.3	47.49	522.28	2388.07	8131.49	8.4318
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	21.61	554.26	2388.08	9052.94	1.3	47.27	522.42	2388.03	8133.23	8.4178
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	21.61	554.45	2388.11	9049.48	1.3	47.13	522.86	2388.08	8133.83	8.3682
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	21.61	554.00	2388.06	9055.15	1.3	47.28	522.19	2388.04	8133.80	8.4294
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	21.61	551.43	2388.19	9065.52	1.3	48.07	519.49	2388.26	8137.60	8.4956
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	21.61	550.86	2388.23	9065.11	1.3	48.04	519.68	2388.22	8136.50	8.5139
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	21.61	550.94	2388.24	9065.90	1.3	48.09	520.01	2388.24	8141.05	8.5646
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	21.61	550.68	2388.25	9073.72	1.3	48.39	519.67	2388.23	8139.29	8.5389
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	21.61	550.79	2388.26	9061.48	1.3	48.20	519.30	2388.26	8137.33	8.5036

20631 rows × 28 columns

Figure 7 Sample of engines dataset (train set from FD001)

This establishes the starting point of this study. Next, the objective is to define a workable data environment and modify the initial provided information to set features and target variables.

2.4. Dataset's structure preparation

This point refers the procedure taken in the Notebook "Datasets Structure" which allows to prepare the three engine's fleet datasets in order to efficiently analyze them in their respective notebooks and prepare the path for the prediction model development. In such notebook the procedure is well explained, however, it is worth stating in this document that the target variable has been defined in the train sets using 2 features: for each engine, it was calculated the maximum cycle achieved, and each row subtracts the current cycle to this value, providing at each operating cycle the remaining ones before failure.

```
RUL_dfs = []

for df in new_list_df:

    max_cycle_value = df.groupby(["ID"])["Cycle"].max().reset_index()
    max_cycle_value = max_cycle_value.rename(columns={"Cycle":"Max_cycle"})
    df = df.merge(max_cycle_value, how="inner", on="ID")
    df["RUL"] = df["Max_cycle"] - df["Cycle"]
    df = df.drop(["Max_cycle"],axis=1)

    RUL_dfs.append(df)

df_train_1 = RUL_dfs[0]
df_train_2 = RUL_dfs[1]
df_train_3 = RUL_dfs[2]
df_train_4 = RUL_dfs[3]
```

RUL
191
190
189
188
187
...
4
3
2
1
0

Figure 8 Code for RUL computation (left), RUL results per cycle (right)

Once columns have been properly named and target variable RUL defined, it is time to jump digging into the data, starting by the first set of engines "FD001". Again, each

notebook accessible in the main repo page contains several paragraphs explaining the steps, assumptions and conclusions taken; however, these chapters will be used to synthesize the information.

The conceived way of working which will be represented in the three notebooks is the following:

1. Data analysis
2. Dataset modifications to prepare for model development
3. Predictive regression models development: in this project, three main models will be used and compared its metrics, these are:
    - **Linear Regressor** (due to simplicity and ease in obtaining results)
    - **Random Forest Regressor**
    - **Support Vector Regressor**
    - In FD002, **Multilayer Perceptron** will be used as well due to the complexity of data
    - In FD003, **Classification models** will be also used due to the duality if the fault
4. Overall metrics comparison and final model selection

The datasets need to be modified in order to be accepted afterwards for the regression models, defining a train/test split for the predictions.

```
X_train = df_train_1.drop(columns=["ID","Cycle","RUL"], axis=1) # Target variable must be out of X_train
y_train = df_train_1["RUL"]

df_test_1_red = df_test_1.groupby("ID").last().reset_index() # y_test is a single value per engine (100 units)
                                                             # hence, X_test's lenght must be equally 100
X_test = df_test_1_red.drop(columns=["ID","Cycle"], axis=1)
y_test = df_RUL_1
```

Figure 9 Train/Test split preparation for models' development

- *X_train* is the data variables for the model training, excluding the columns "ID" and "Cycle" as they do not provide any information (simple linear increasing features), and obviously "RUL" as is the target feature.
- *y_train* is the target set, which is the remaining useful life for each of the current sensor states.
- *df_test* is performed by grouping the dataset by engine ID, and retain the last row for each ID, this means the dataset keeps number of rows equal to the number of engines of the test set, which matchs directly with the number of rows of the real RULs set.
- *X_test* is produced from the df_test the same way as X_train, excluding from these last rows per engine the "ID" and "Cycle" columns
- *y_test* is the remaining useful life per each of the engines from df_test, it is the target used to compare the predictions

3. First set of Engines study (FD001)

In the first dataset, the engine's fleet conditions are the following:

- 100 train and 100 test trajectories
- 1 Flight conditions (Sea Level)
- 1 Type of engine fault (by HPC degradation)

Once NaN values have been removed (last 2 columns which had 100% of null rows), comes the Feature Analysis.

3.1. Data analysis

Dataset features are represented below, selecting a random engine to provide a first sight of an entire operational life of the engine.

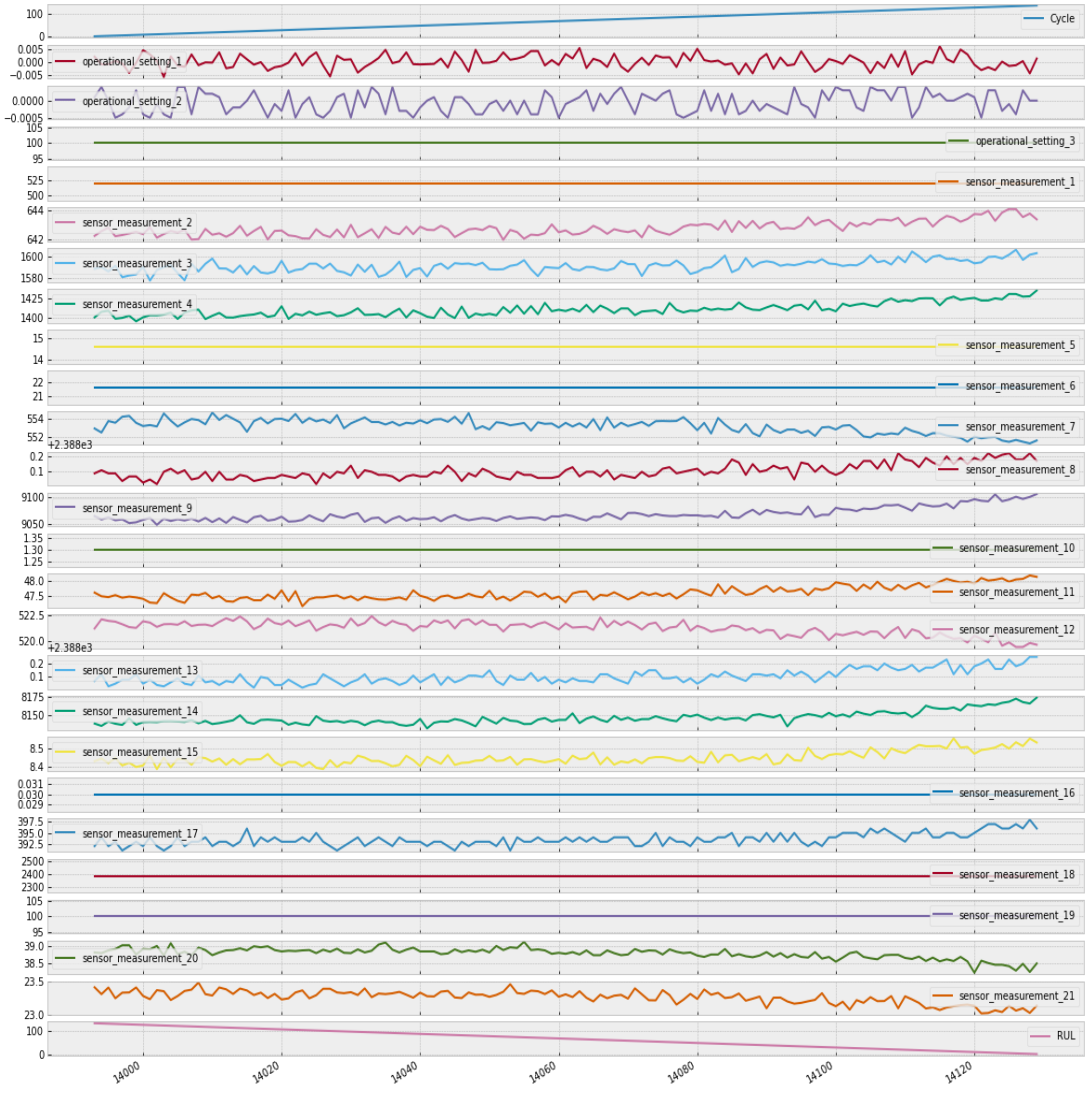


Figure 10 FD001 features evolution during cycles (one single engine)

The main target is to predict the RUL of the test set, a first approach is to have an idea of the average life (maximum number of cycles) of this set of engines, which is showed below, stating a mean of 206 cycles as maximum life per engine. In the case of test set this number is reduced to 130, as this database is not completed.

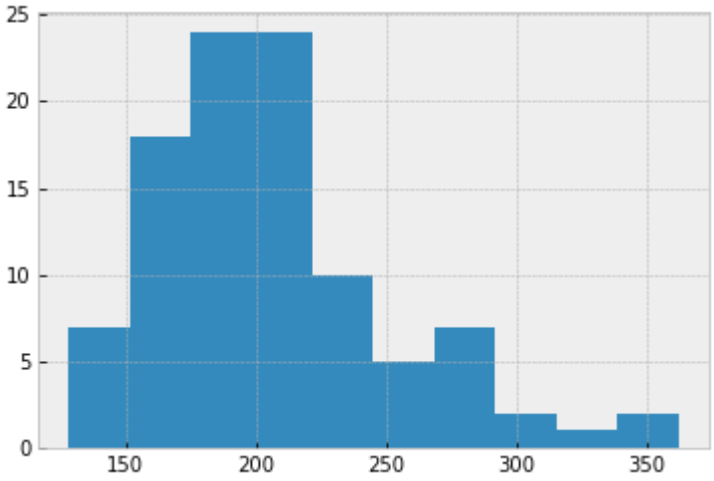


Figure 11 Maximum engine's life distribution

The life decreasing progression per engine can be found in the following plot, declaring that each engine starts performing in a different health state, and developing "faulty" behavior at a different moment.

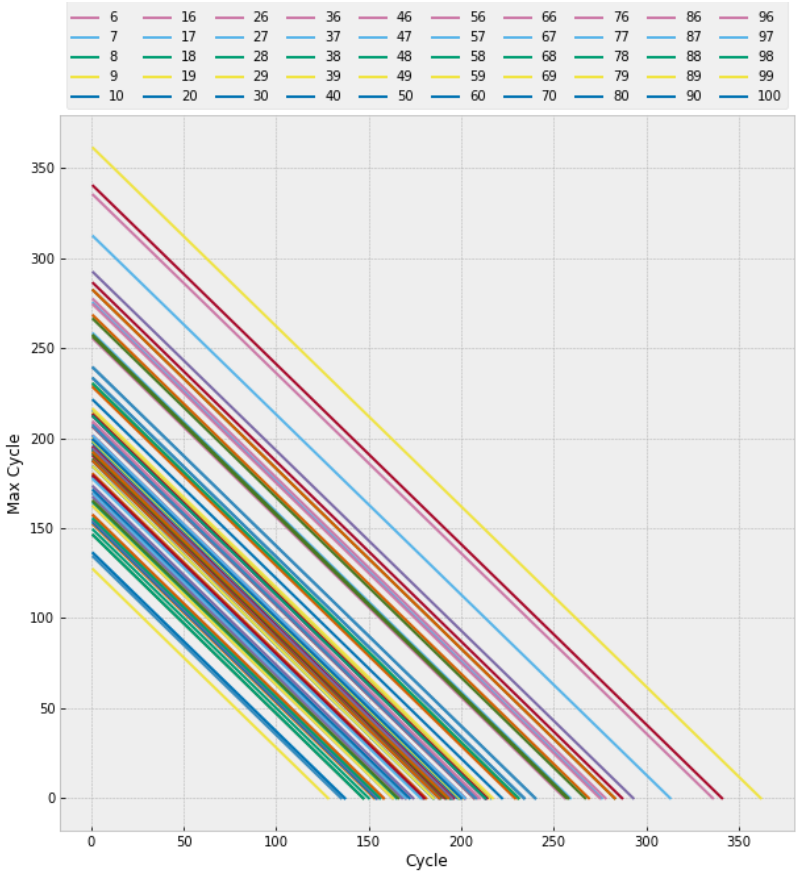


Figure 12 RUL distribution

The flight conditions of the engine's performance in each dataset are defined by the operational conditions:

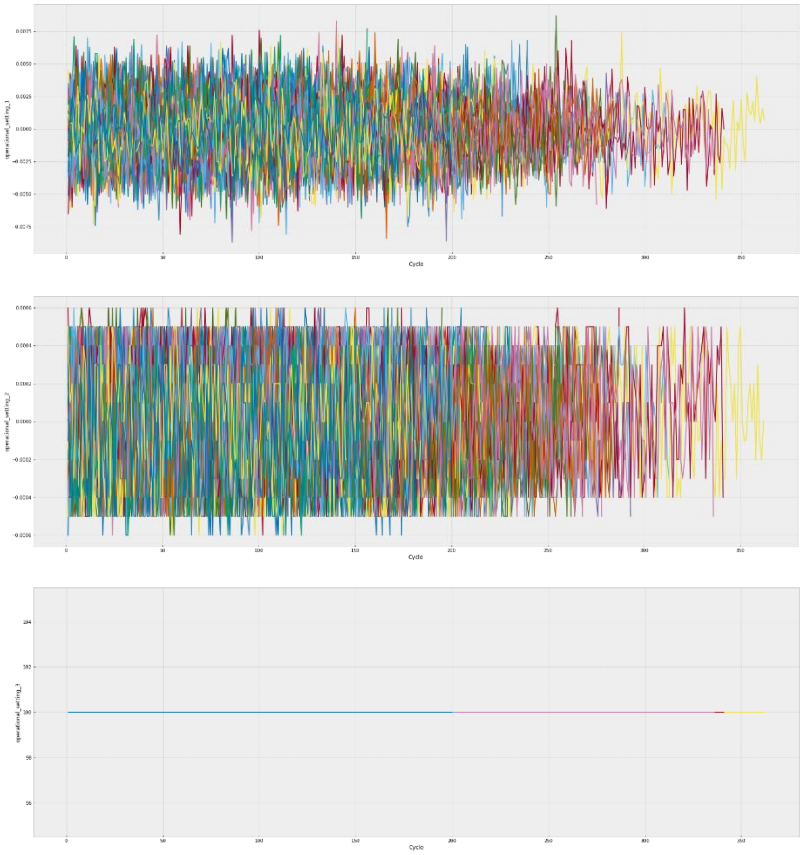


Figure 13 (From above to below) operational settings 1, 2 and 3 dataset's evolution per cycle

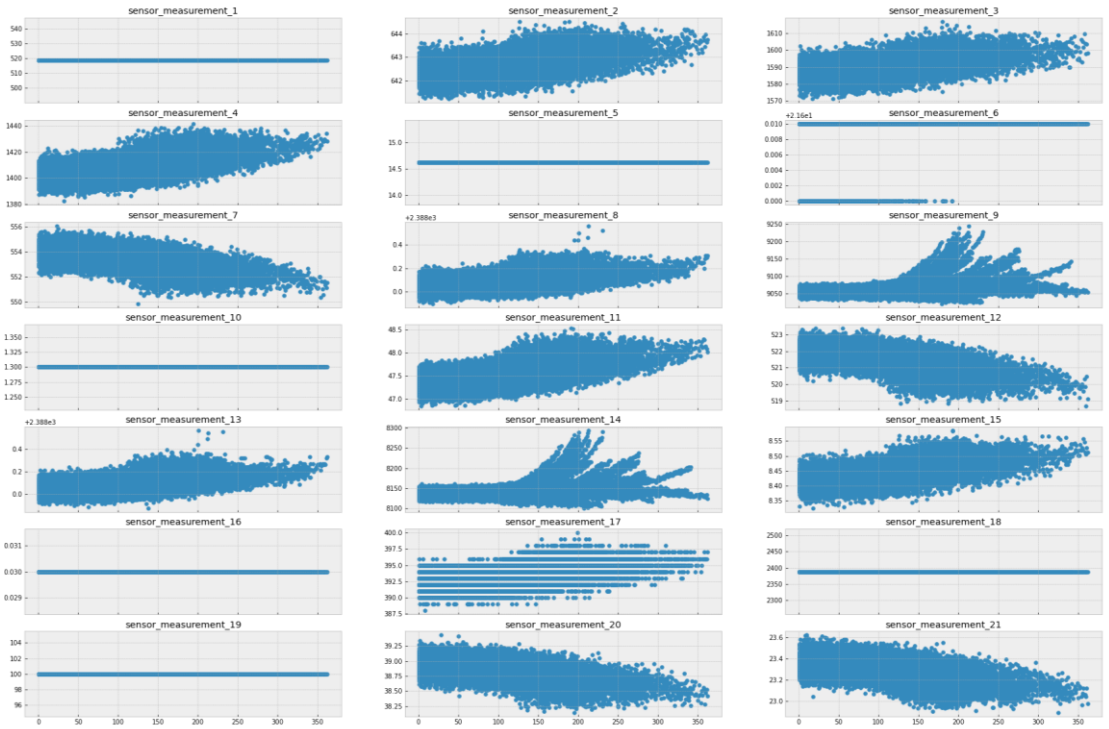


Figure 14 Sensor's dataset evolution per cycle

In these visualizations it has been identified some constant variables. This initial study is based on a regression problem, and the characteristic of this project tells that the target variable (end of life of engine) appears when a fault is found. This usually happens when any (or some) component fails during performance, hence, the sensors may have noted this change in component's behavior by increasing or decreasing out of normal standards their values. Accepting this, it is assumed that constant features will not provide any useful information, and thus, will be removed from dataset.

Additionally, and during this "removal-analysis" step, it was deployed the correlation matrix to find any strong link between features and target variables, finally it was found a correlation of 0.96 between sensors 9 and 14. Due to the lower correlation with RUL, it was selected to remove sensor 14.

Thus, the list of features removed is the following:

- Operational Setting 3
- Sensor Measurements 1, 5, 6, 10, 14, 16, 18 and 19

3.2. Regression models development

As commented in previous chapter, in this notebook 3 different models will be deployed: Linear Regression, Random Forest Regressor and Support Vector Machine. During the feature engineering process of this notebook, several stages have been taken to try improving the metrics. This procedure is detailed below step by step:

1. Baseline model
2. Reduce Dimensionality
    i. Models with top features by slope value
    ii. Delimiting RUL by above
    iii. Combination of previous 2 steps
    iv. Principal Component Analysis (combined with RUL delimited)
3. Augment Dimensionality
    i. Addition of lagged variables (combined with RUL delimited)
    ii. Addition of polynomial variables

3.2.1. Baseline Models

Structure of the train/test split for initial models was showed in Figure *9*, the results for the three models are below:

- Linear Regression metrics

```
TRAIN SCORES - - - -              TEST SCORES - - - -
 MAE:  27.68539128919656           MAE:  20.350894668448746
 MSE:  1996.609336014649           MSE:  1042.8056960525923
 RMSE:  44.68343469357127          RMSE:  32.29250216462937
 R2 score:  0.5791615760470854     R2 score:  0.3961294494659049
```

- Random Forest Metrics

  Random Forest regressor permits a wide range of parameters customization. *RandomizedSearchCV* was used to find the most optimal combination of the following features which were found to be ones which affected the most to model's accuracy: [7]

  - N_estimators
  - Max_depth
  - Min_samples_leaf

```
RANDOM FOREST TRAIN SCORES - - - -
 MAE:  18.604071701971094
 MSE:  1511.5729686704285
 RMSE:  38.878952772296074
 R2 score:  0.6813958673083034
```

```
RANDOM FOREST TEST SCORES - - - -
 MAE:  18.918653897286916
 MSE:  1065.193792884029
 RMSE:  32.637306765173335
 R2 score:  0.38316489393059594
```

- SVR Metrics

  Similar to Linear regressor, but SVR sets a boundary at specific distance (epsilon) from reference data and all the points that fall into the boundary area are ignored. This allows to reduce the computational load. The problem comes when data features ranges vary importantly, as distance comparisons get eclipsed by outliers. Due to this reason, this model needs a scaler before fitting into the model data. [8]

  In this project, *StandardScaler()* was used as a preprocessor for X_train and X_test.

  Regarding epsilon, during the project evolution it was found that best metrics were found with epsilon=0.2, and practically null difference in accuracy was encountered if changed, however addition of extra loop slows the computations, hence, epsilon=0.2 was chosen for the whole project.

```
SVR TRAIN SCORES - - - -
 MAE:  20.207105282145022
 MSE:  1827.7211803380164
 RMSE:  42.75185587010249
 R2 score:  0.6147592385327962
```

```
SVR TEST SCORES - - - -
 MAE:  10.461170640728824
 MSE:  695.2252110588136
 RMSE:  26.36712367814915
 R2 score:  0.5974072326834559
```

The best baseline metrics are achieved by the SVR with a R2 score of 0.597. Meanwhile the best train score comes from the RFR, which usually tends to overfit.

3.2.2. Top Features Selection

As commented in the notebook, aiming to increase accuracy by reducing dimensionality, it was aimed to only retain a portion of the features. Based in the trend value, the ordered top features were selected.

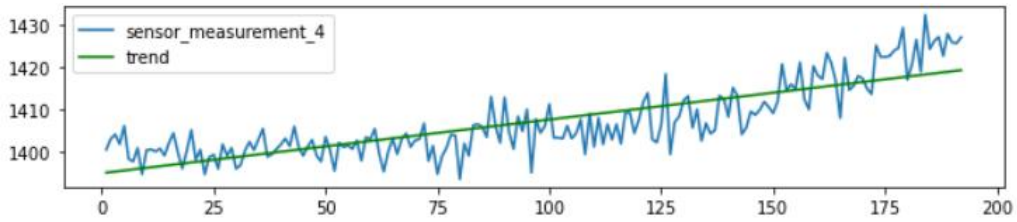
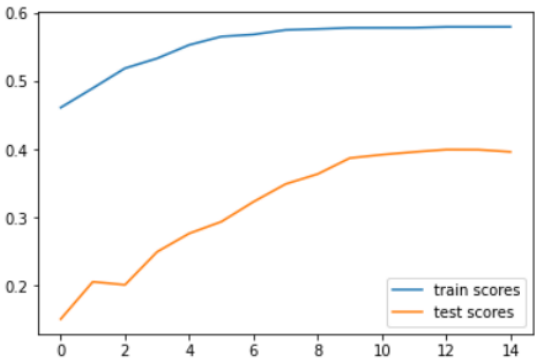


Figure 15 Sample of sensor measurement trend

Initially, selecting only the top 10 features sorted by trend was not very useful, hence, it was created a loop to find the best number of features. Each loop simulation, the next column by trend importance was added to the dataset, and the models were fitted again.

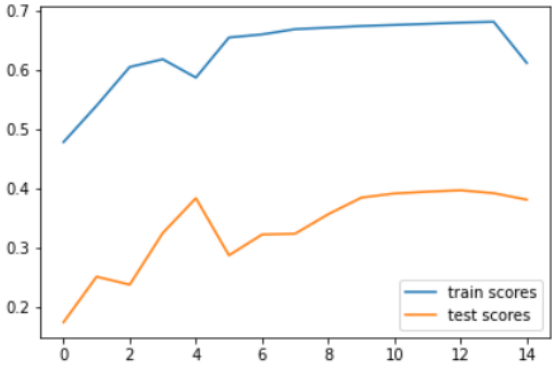
- Linear Regressor + Top Features metrics

```
Train Score from max Test Score:  0.5791177107952363
 Max Test Score:  0.3994385598171166
 Achieved with 12 features out of 15
```



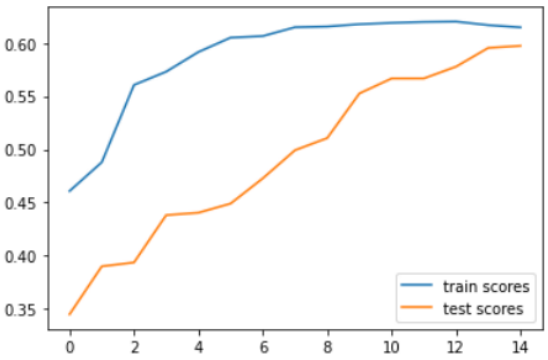
- Random Forest + Top Features metrics

```
Train Score from max Test Score:  0.6804065932666001
 Max Test Score:  0.3969872519003438
 Achieved with 12 features out of 15
```



- SVR + Top Features metrics

```
Train Score from max Test Score:  0.6147592385327965
 Max Test Score:  0.5974072326834532
 Achieved with 14 features out of 15
```



Again, best metrics are found in SVR. Overall, very small improvement regarding baseline models (around 0.7%).

3.2.3. RUL upper limitation

This aspect is already explained in notebook, but basically the intention is to reduce the load of computation by delimiting the RUL lineal function. [9]If the user checks the sensor's performance, it can be observed that the trend of sensors does not start at the very beginning, but at some point, when the fault begins. Hence, values of RUL before that point may not provide much info and additionally increase the range value of the target.

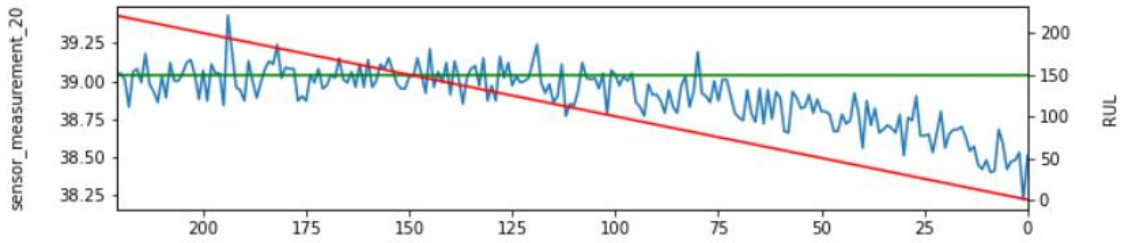


Figure 16 Clear example where the moment when sensor starts to decrease may be the intersection of the horizontal line (first value) and the RUL

Checking the different graphs, the user shall notice that these potential upper values could be in the range between 100-200 cycles.

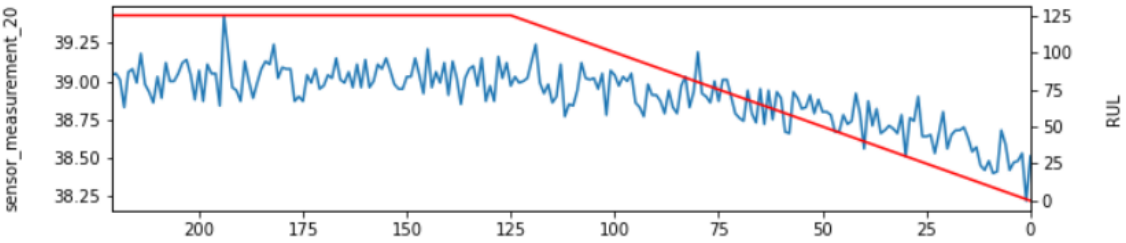
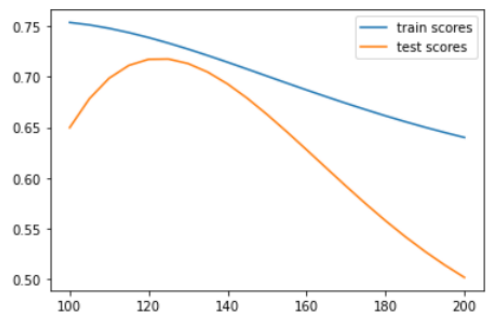


Figure 17 Same sensor measurement with Limited RUL in 125 initially

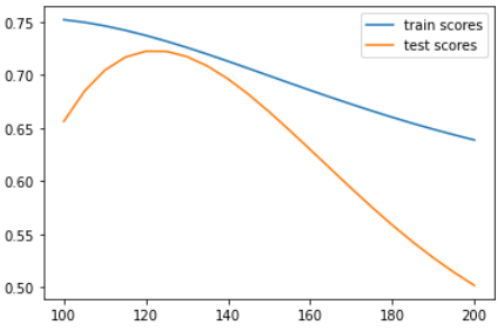
This featuring was achieved thanks to the *.upper(clip)* tool, the three models improved drastically their performance selecting a random limit of 125, however, the same way as performed with the top features, a limit selected loop is performed to find the most optimal upper RUL for each model.

- Linear Regressor + RUL limit (right with 12 top features)

```
Train Score from max Test Score:  0.7334354893554875
 Max Test Score:  0.7176799423105168
 Achieved with 125 upper clipped RUL
```

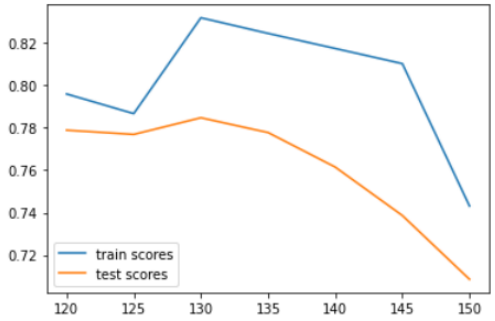



```
Train Score from max Test Score:  0.7370113007464004
 Max Test Score:  0.7221952356845288
 Achieved with 120 upper clipped RUL
```

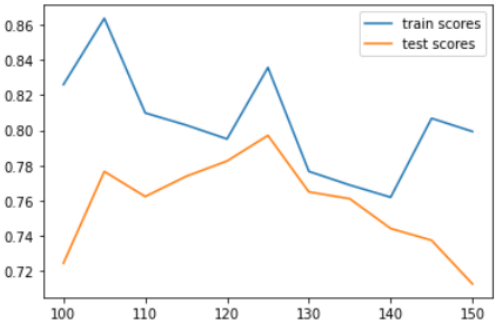


- Random Forest + RUL clip

```
Train Score from max Test Score:  0.8317160477687705
 Max Test Score:  0.7846488057987547
 Achieved with 130 upper clipped RUL
```

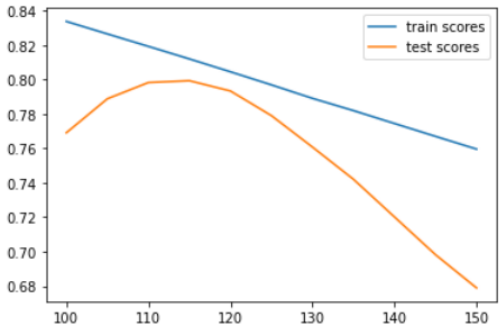



```
Train Score from max Test Score:  0.8356520989631885
 Max Test Score:  0.7970407645565291
 Achieved with 125 upper clipped RUL
```

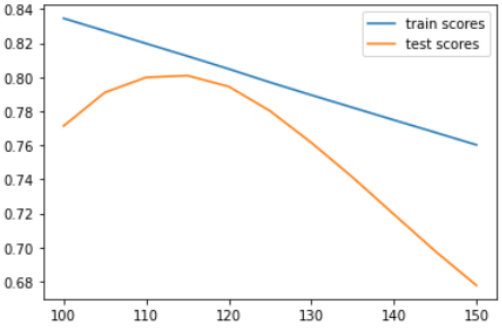


- SVR + RUL clip

```
Train Score from max Test Score:  0.8121014574002057
 Max Test Score:  0.7994286961760717
 Achieved with 115 upper clipped RUL
```




```
Train Score from max Test Score:  0.8124484077557991
 Max Test Score:  0.8010054292450628
 Achieved with 115 upper clipped RUL
```



As commented, it is observed a high improvement in the three models (around 40-80% depending on which). Additionally, the model was tested selecting only the n top features stated in the previous section, in the three cases the result was slightly improved, and sometimes the best limit value for RUL was changed

3.2.4. Principal Component Analysis

Continuing the line of reducing the model dimensionality, PCA was used as well. In this case, the trend value was not used as a factor to determine whether a feature is included or not, but the eigenvectors and eigenvalues. [10] In PCA, each of the components corresponds to an eigenvector, and the component order is based in the eigenvalue. These measures allow to compute the principal components, which are condensed information provided by multiple variables in just a few ones. Each principal component (Z) is obtained from lineal combination of the original variables. The first principal component of a group of variables is the normalized lineal combination of these variables which has higher variance. Due to this, PCA is very sensible to outliers, and previous standardization will be needed.

Given a data matrix of *n* x *p* dimensions, the maximum number of principal features to be obtained is *n-1* or *p* (the lower of the two values is the limitation). [11] However, due to the objective of reducing dimensionality, it could be interesting to use the minimum number of

features that provide competent data understanding (for example, evaluate the proportion of accumulative variance):

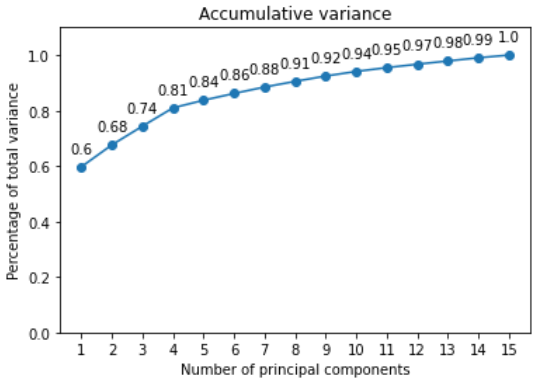
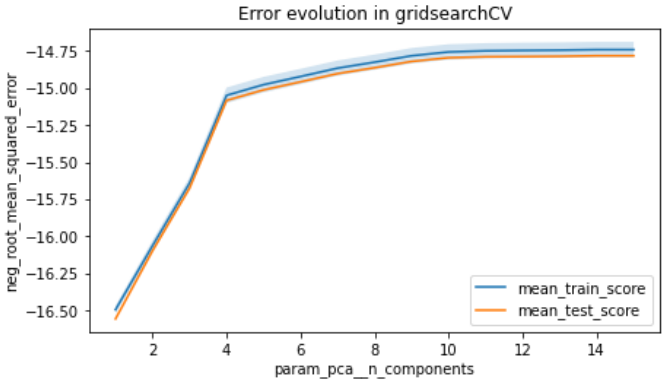


Figure 18 Accumulative variance per principal component in X_train set

As observed in graph above, the accumulative variance tends to be practically linear when reaching 0.8 or total variance. This is a signal that, initially, there is no minimal number of features needed to provide a competent variance result. When the pca model is fitted and a GridSearchCV is used to find the best number of parameters, the results are obtained based on the error evolution.

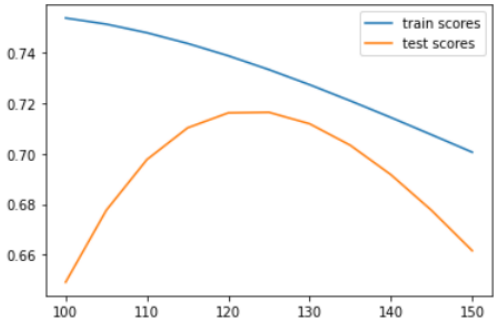



```
Best number of components found:  {'pca__n_components': 14}
```

Hence, the PCA has provided a new number of features to be fitted in the models, so let's try again and calculate the new metrics:

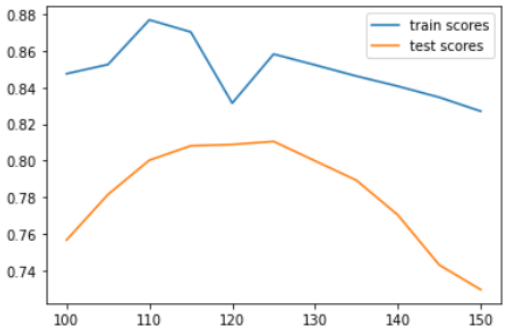
- Linear Regression + PCA + RUL clip

```
Train Score from max Test Score:  0.7333701974416749
 Max Test Score:  0.716460897932507
 Achieved with 125 upper clipped RUL
```



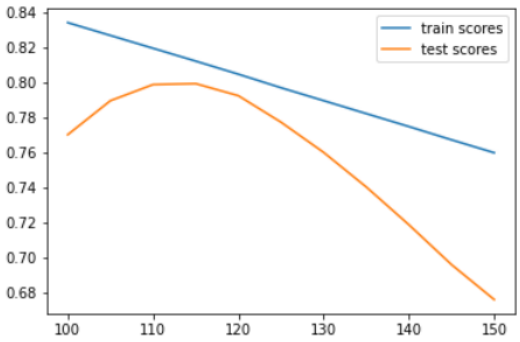
- Random Forest + PCA + RUL clip

```
Train Score from max Test Score:  0.8581559355084927
 Max Test Score:  0.8104571164428893
 Achieved with 125 upper clipped RUL
```



- SVR + PCA + RUL clip

```
Train Score from max Test Score:  0.812084411605407
 Max Test Score:  0.7992138904937213
 Achieved with 115 upper clipped RUL
```



This new methodology has provided a new record in the model accuracy, this time allowing the RandomForest Regressor to achieve more than 0.81 score when using a RUL limited in 125.

3.2.5. Addition of Lagged Variables

Changing the current methodology of reducing the dimensionality of the dataset, let's turn around 180º and try to increase it looking for metrics improvement. As the dataset may be observed as a timeseries analysis, the inclusion of lagged variables will be now performed.

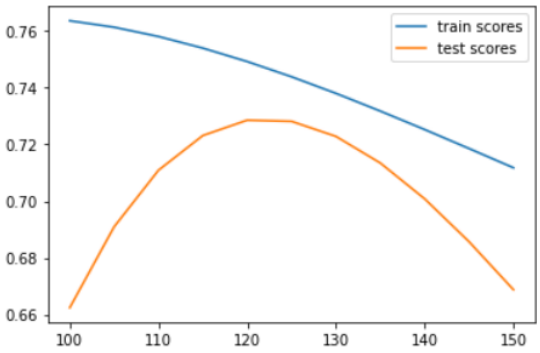
Lagged dependent variables (LDVs) have been used in regression analysis to provide robust estimates of the effects of independent variables. [12] The computing of lagged variables simply includes additional columns with the lagged values of each column. This is normally performed when expected that model behavior is strongly determined by its past levels.

```
df_train_1_lagg = df_train_1.copy()
sensor_cols = df_train_1.columns[2:len(df_train_1.columns)-1]
lag_cols = []
for col in sensor_cols:
    col = col + "_lag1"
    lag_cols.append(col)
df_train_1_lagg[lag_cols] = df_train_1.groupby("ID")[sensor_cols].shift(1)
df_train_1_lagg.dropna(inplace=True)
```

Figure 19 Preparation of lagged dataset (RUL is excluded in lagging)

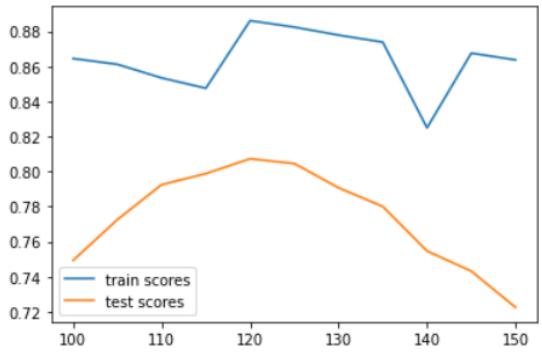
- Linear Regression + Lagged variables + RUL clip

```
Train Score from max Test Score:  0.7492616236267262
 Max Test Score:  0.728586775182325
 Achieved with 120 upper clipped RUL
```



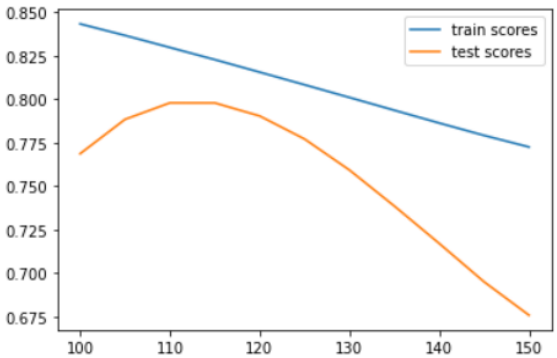
- Random Forest + Lagged Variables + RUL clip

```
Train Score from max Test Score:  0.8860034496696573
 Max Test Score:  0.8072395484579106
 Achieved with 120 upper clipped RUL
```



- SVR + Lagged Variables + RUL clip

```
Train Score from max Test Score:  0.8297024571727718
 Max Test Score:  0.7978380071846183
 Achieved with 110 upper clipped RUL
```



Best metrics are observed again in the Random Forest regressor which improved from based line the test accuracy 0.01. However, train score starts to be quite high, which means that model starts to overfit due to the high number of features.

In this case the number of lagging processes has been set as 1 extra lagged variable set, but let's see what happens when it is applied a loop of different lagged number variables, which is shown below:

```
laggings = np.arange(1,11).tolist()
best_tr_scores = []
best_ts_scores = []

for number in laggings:

    df_train_lagg = df_train_1.copy()
    df_test_lagg = df_test_1.copy()
    lag_cols = []

    for col in sensor_cols:

        col = col + "_lag1"
        lag_cols.append(col)

    df_train_lagg[lag_cols] = df_train_1.groupby("ID")[sensor_cols].shift(number)
    df_train_lagg.dropna(inplace=True)

    df_test_lagg[lag_cols] = df_test_1.groupby("ID")[sensor_cols].shift(number)
    df_test_lagg.dropna(inplace=True)

    X_trainL = df_train_lagg.drop(columns=["Cycle","RUL"], axis=1)
    df_test_redL = df_test_lagg.groupby("ID").last().reset_index()
    X_testL = df_test_redL.drop(columns=["Cycle"], axis=1)
    y_testL = df_RUL_1
```

Figure 20 Code for lagged variables loop

This loop for lag variables is combined as well with RUL limit loop:

- Linear Regression+ Lagging variation + RUL clip variation

```
best metrics with 10 number of lagging are:
 train: 0.7415702407007141 test 0.7343937851505395 with clip 125
```

- Random Forest + Lagging variation + RUL clip variation

```
best metrics with 20 number of lagging are:
 train: 0.8356608840452703 test 0.8032197084430521 with clip 120
```

  In this case, the parameters of the random forest were chosen previously, due to the enormous time that was taking to run the multiple data combinations. The chosen parameters were selected attending at previous best estimators when fixed RUL:

  - N_estimators = 80
  - Max_depth = 14
  - Min_samples_leaf = 60

- SVR + Lagging variation + RUL clip variation

```
best metrics with 15 number of lagging are:
 train: 0.8178703459789346 test 0.8098110522098704 with clip 115
```

From this featuring, the best metric is obtained from the SVR, with variables lagged 15 times and with RUL target clipped in 115 cycles. Some improvements have been notices by using larger lagging, but not as important as expected due to the higher time taken to run the simulations. Finishing with the trend of augmenting the dimensionality, the following section ends the FD001 modelling.

3.2.6. Polynomial Features

Still in the line of augment dimensionality, the last resource taken in this analysis comes by the inclusion of polynomial combinations of the dataset features. This has been studied that may provide the revealing of patterns that are not obvious at first sight for the model. The inclusion of this polynomial featuring increases hugely the dimensionality of the dataset. [13]

```
polynomial = PolynomialFeatures(2)

X_train_pol = polynomial.fit_transform(X_train)
X_test_pol = polynomial.fit_transform(X_test)

print("X_train columns shape increment: ", "\n",
     "Initially: ", X_train.shape[1]," columns, After polynomial", X_train_pol.shape[1], "columns")
```

```
X_train columns shape increment:
 Initially:  15  columns, After polynomial 136 columns
```

Figure 21 Dataset dimension before and after polynomial combination of features

- Linear Regressor + Polynomial features + RUL clip

```
MAE:  13.020617485046387
MSE:  386.73539660340026
RMSE:  19.665589149664452
R2 score:  0.7773047897111629
```

```
MAE:  11.996847867965698
MSE:  411.4325646676223
RMSE:  20.283800547915625
R2 score:  0.7617465935658243
```

- RandomForest + Polynomial features + RUL clip

```
MAE:  8.783218898844268
MSE:  260.7060867249136
RMSE:  16.14639547158788
R2 score:  0.8498766926516341
```

```
MAE:  10.169760516416446
MSE:  339.42049500794155
RMSE:  18.42336817761458
R2 score:  0.8034475243481374
```

- SVR + Polynomial features + RUL clip

```
MAE:  8.435562935364999
MSE:  360.44667791627353
RMSE:  18.985433308625684
R2 score:  0.7924427155066067
```

```
MAE:  8.583750681493942
MSE:  374.5251898268563
RMSE:  19.352653301985654
R2 score:  0.7831190092020519
```

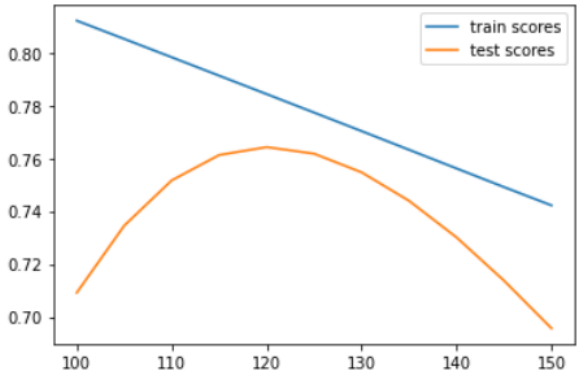
The addition of polynomial combination of the dataset features provides interesting results, almost beating the best metrics acquired so far by the PCA. As a last step in this FD001 study, a final loop of different RUL clips will be applied, in order to find if most optimal RUL limit has changed.

Due to endless time of simulation needed to perform this last study (as a result of the high dimensionality of the dataset) some features might be reduced its range, for example, there will be less clips to test (selecting the range where previous best values in this study were found) and in the case of the Random Forest Regressor, it won't be possible to perform the RandomizedGridSearch due to the large simulation time, hence, best parameters in the fixed RUL case will be re-used.

Final results are below showed:

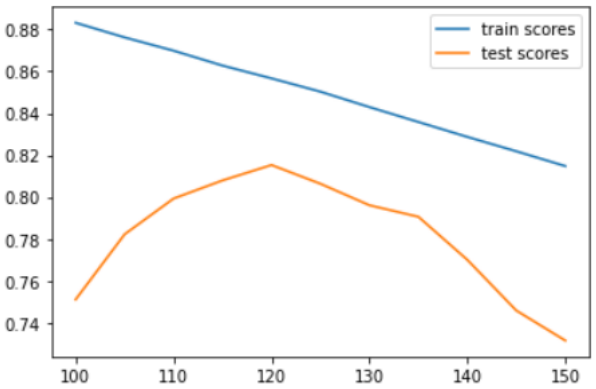
- Linear Regressor + Polynomial Features + RUL limit variation

```
Train Score from max Test Score:  0.7843018713524302
 Max Test Score:  0.7642406815043365
 Achieved with 120 upper clipped RUL
```



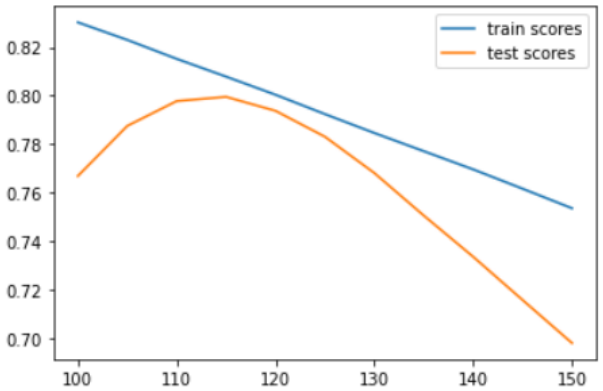
- Random Forest + Polynomial features + RUL limit variation

```
Train Score from max Test Score:  0.8566766403696677
 Max Test Score:  0.8153106249042961
 Achieved with 120 upper clipped RUL
```



- SVR + Polynomial features + RUL limit variation

```
Train Score from max Test Score:  0.8078793426575087
 Max Test Score:  0.799546408531281
 Achieved with 115 upper clipped RUL
```



With the above results, it is confirmed that thanks to this last featuring thebest score has been beaten. Once the different models and featurings have been showed up, the final model to predict remaining useful lifes for engines which perform at 1 single flight condition and may fail due to only HPC degradation would be:

- A random forest regressor, optimised with a RandomizedSearchCV
  - Max_depth = 9
  - Min_samples_leaf = 10
  - N_estimators = 64
- Featured with polynomial features combination (with 2^{nd} degree)
- Limiting the RUL from above at 120 cycles

This provides **0.856** r2 score for train set and **0.815** for test score.

4.Second set of engines (FD002)

In second dataset, conditions are the following ones:

- 260 train units and 259 test units
- Six flight different conditions
- 1 single fault mode (HPC degradation)

Initial visualization of dataset features, selecting random engine provides the below graphs:

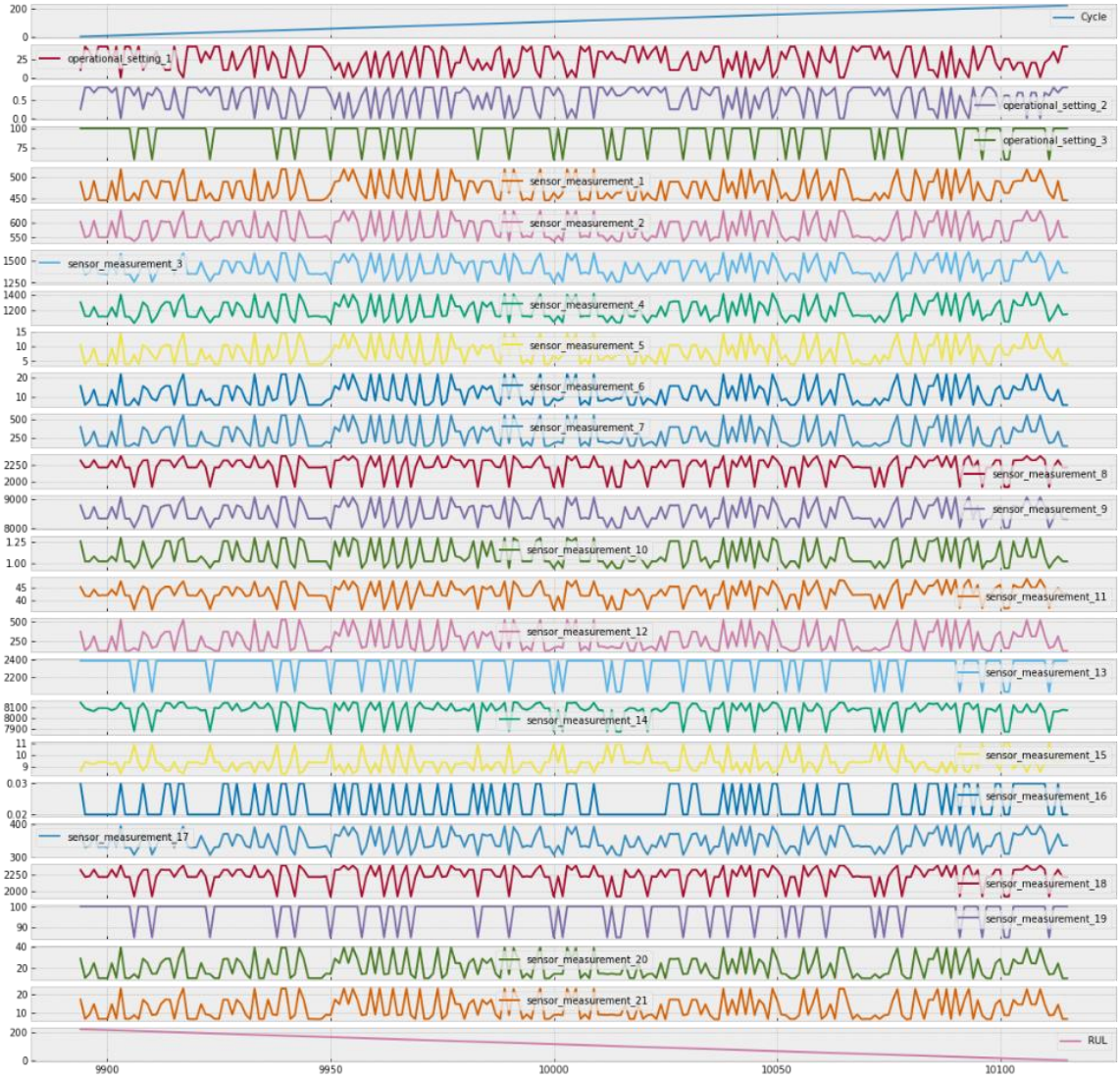


Figure 22FD002 features evolution during cycles (one single engine)

4.1. Data analysis

Aiming directly to the features analysis, and starting with the operational settings:

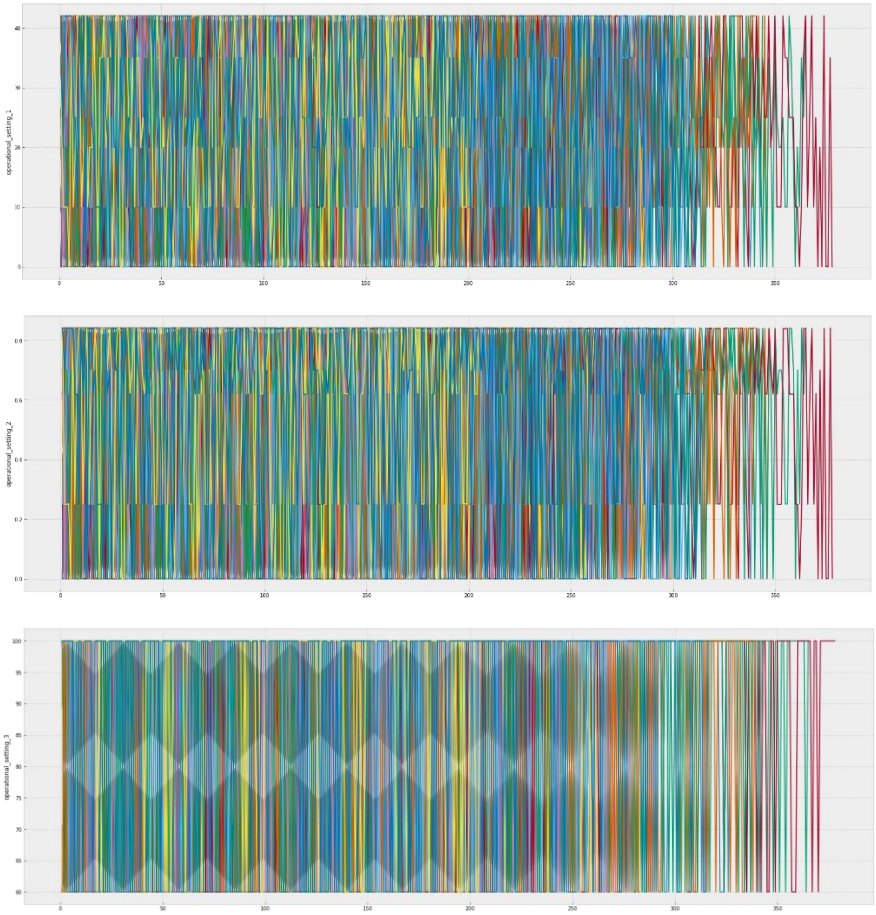


Figure 23 (From above to below) operational settings 1, 2 and 3 dataset's evolution per cycle

At this stage, it is observed that operational setting 3 is no longer constant during performances, that means that the different flight operations have a direct effect on this particular setting.

These 6 different flight conditions should be noticed somehow in the engine's dataset, and as the 3 operating conditions are supposed to represent these characteristics, the 6 different operation modes should be observed. In order to get a better idea, the column values will be rounded:

operational_setting_1	operational_setting_2	operational_setting_3	
0.0	0.00	100.0	8044
10.0	0.25	100.0	8096
20.0	0.70	100.0	8122
25.0	0.62	60.0	8002
35.0	0.84	100.0	8037
42.0	0.84	100.0	13458

Figure 24 6 different flight conditions according to operational setting values

Checking at the sensors:

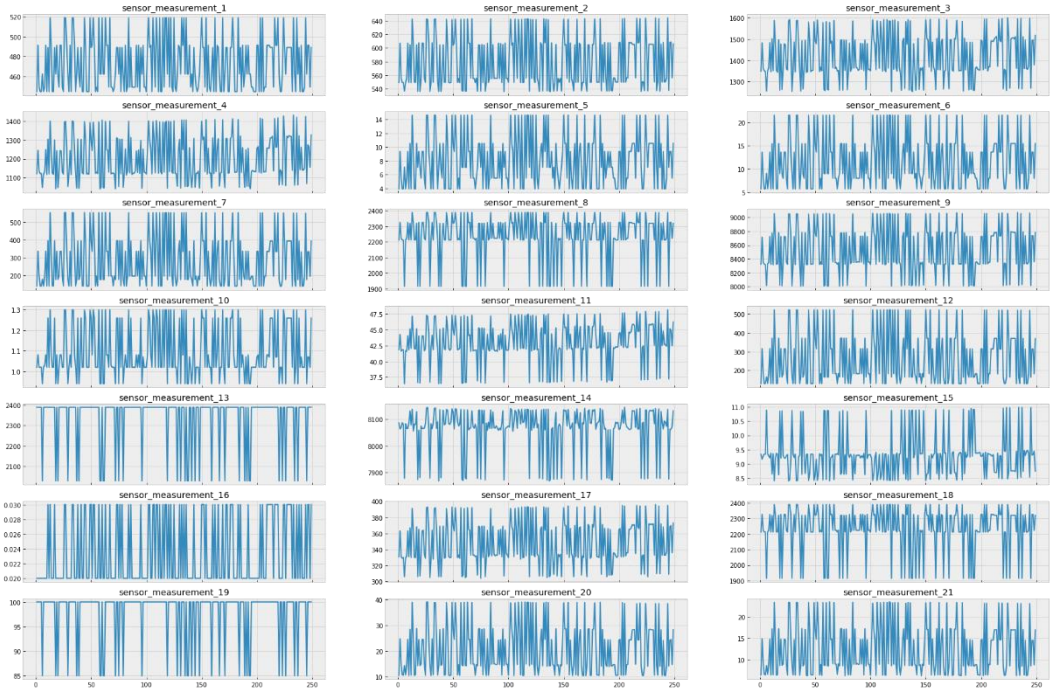


Figure 25 FD002 single engine sensor measurements

Unlikely to FD001, when sensors are plotted, there is no trend initially observed. That would initially mean a difficulty to find most useful sensors from the dataset, and to obtain potential insight from performance.

4.2. Models Development

As commented in previous chapter, in this notebook 3 different models will be deployed, during the feature engineering process of this notebook, several stages have been taken to try improving the metrics. This procedure is detailed below step by step:

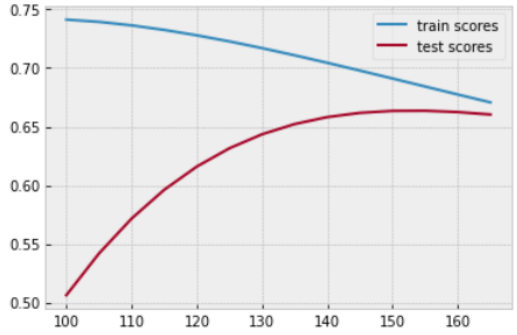
1. Baseline model
2. Previous used featuring
    i. Delimiting RUL by above
    ii. Principal Component Analysis (combined with RUL delimited)
3. Neural Network development (Multilayer Perceptron)
    i. Baseline NN
    ii. Addition of One Hot Encoder
    iii. Grouping operating conditions
    iv. Lagged variables
    v. Smoothed lagged variables

4.2.1. Baseline Models

Beginning with baseline models, from previous notebook it was observed that delimiting the RUL provided huge improvement from the very basic models, hence, the starting point of the different metrics will begin with the search of most optimal upper RUL clip:

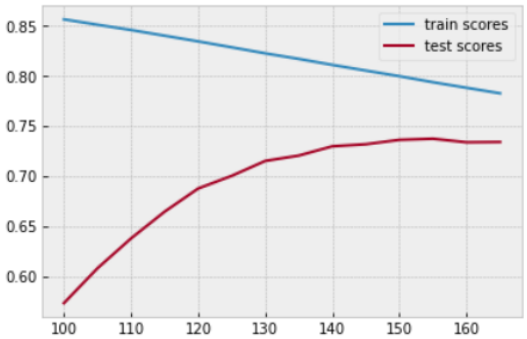
- Linear Regression + RUL clip

```
Train Score from max Test Score:  0.6842001682342748
 Max Test Score:  0.6635389351331296
 Achieved with 155 upper clipped RUL
```



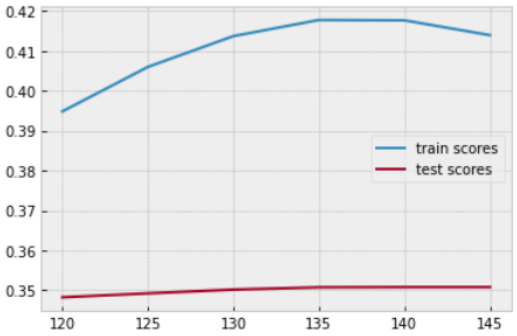
- Random Forest + RUL clip

```
Train Score from max Test Score:  0.7937520083031044
 Max Test Score:  0.7372931863543086
 Achieved with 155 upper clipped RUL
```



- SVR + RUL clip

```
Train Score from max Test Score:  0.4139925686080659
 Max Test Score:  0.35076913939226606
 Achieved with 145 upper clipped RUL
```



General metrics are quite worse compared with previous notebook, maybe excepting the Random Forest which already reached quite competent score. However, due to the more complex format of this dataset compared with previous one, and looking for introduce new ways of prediction, a neural network will be the next model to be developed.

As commented in Notebook FD002, the multilayer perceptron (MLP) was chosen as the model to work with due to the good combination of its accuracy with non-linear predictions, together with its ease to use. [14]

4.2.2. Baseline Neural Network

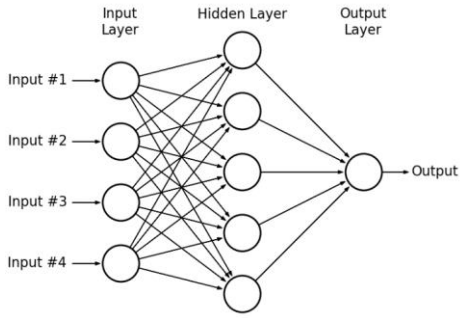


Figure 26 Example of Multilayer Perceptron Network

It is recommended when working with neural networks to define a validation set, allowing to avoid overfitting. The split in train and validation different sets was aimed at trying to find a good division which would keep similar distributions between both groups. It was selected the average max number of cycles of engines as a factor to determine similarity.

In order to obtain 2 similar groups, the p-values were studied as a sign of similarity between 2 datasets (Kolmogorov-Smirnov test), picking random samples of engines and making several comparisons a validation set composed with 20% of engines from training set was found. [15]

```
p_values = []
df_2 = df_train_2.copy()
engines_list = []
for i in range(100): # 100 loops to find the best random selection of engines
    overall_units = np.arange(1,261).tolist()
    engines = random.sample(range(1,261), 52)
    for element in overall_units:
        if element in engines:
            overall_units.remove(element)

    df_train = df_2[df_2.ID.isin(overall_units)]
    df_val = df_2[df_2.ID.isin(engines)]

    x = df_train.groupby("ID")["Cycle"].max()
    y = df_val.groupby("ID")["Cycle"].max()

    engines_list.append(engines)
    p_values.append(ks_2samp(x,y)[1]) # append all the p-values to finde the closest to 0.95
```

Figure 27 code that allowed to find engines split which provided highest similarity

```
Seed with closest similarity between train and validation set is:  48  with a p-value of:
 0.9492992306166091

 list of engines selected to be in validation:
 [92, 213, 55, 184, 235, 180, 171, 110, 227, 27, 43, 203, 47, 141, 103, 130, 87, 169, 41, 137, 154, 114, 91, 232, 81, 247, 84,
217, 96, 106, 196, 163, 111, 212, 186, 83, 156, 62, 145, 140, 258, 97, 197, 256, 191, 66, 93, 116, 115, 102, 1, 226]
```

Once training and validation set are defined, the baseline NN is deployed. In order that the predictions made by the model can user the appropriate efficient computation from the backend, each prediction is preceded with *save_weights.* [16]

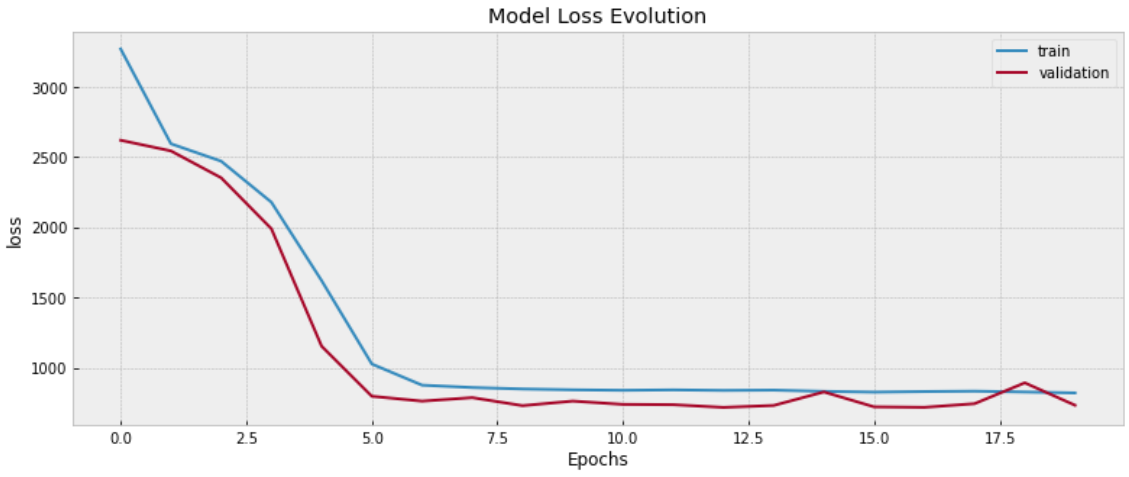
The neural network needs to be preceded as well by a pre-scaling. Initial three-layer, typical number of neurons and *relu* function activation was chosen to start the model featuring.

```
X_train = training_2.drop(["ID","Cycle","RUL"], axis=1)
y_train = training_2["RUL"].clip(upper=155)
X_test = df_test_2.groupby("ID").last().reset_index().drop(["ID","Cycle"], axis=1)
y_test = df_RUL_2

scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_val = validation_2.drop(["ID","Cycle","RUL"], axis=1)
y_train_val = validation_2["RUL"].clip(upper=155)
X_train_val_scaled = scaler.transform(X_train_val)
```

```
input_dim = len(X_train.columns)

model = Sequential()
model.add(Dense(16,input_dim=input_dim, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(1))

model.compile(loss="mean_squared_error", optimizer="adam")
model.save_weights('MLP_weights_initial.h5')
```

```
epochs = 20

model.compile(loss='mean_squared_error', optimizer='adam')
model.load_weights('MLP_weights_initial.h5')

history = model.fit(X_train_scaled, y_train,
                    validation_data=(X_train_val_scaled, y_train_val),
                    epochs=epochs)
```

Figure 28 Baseline MLP coding




```
TRAIN METRICS for Model
 MAE:  18.471939086914062 MSE:  801.3333063444104 RMSE:  28.30783118404535 R2 score:  0.6975802470333722

TEST METRICS for Model
 MAE:  17.77184295654297 MSE:  914.4340400026721 RMSE:  30.239610447270515 R2 score:  0.6838230077600913
```

Figure 29 Baseline coding metrics

So far it was achieved better results compared to supervised learning initial models, but let's see what else it can be applied in order to further improve results.

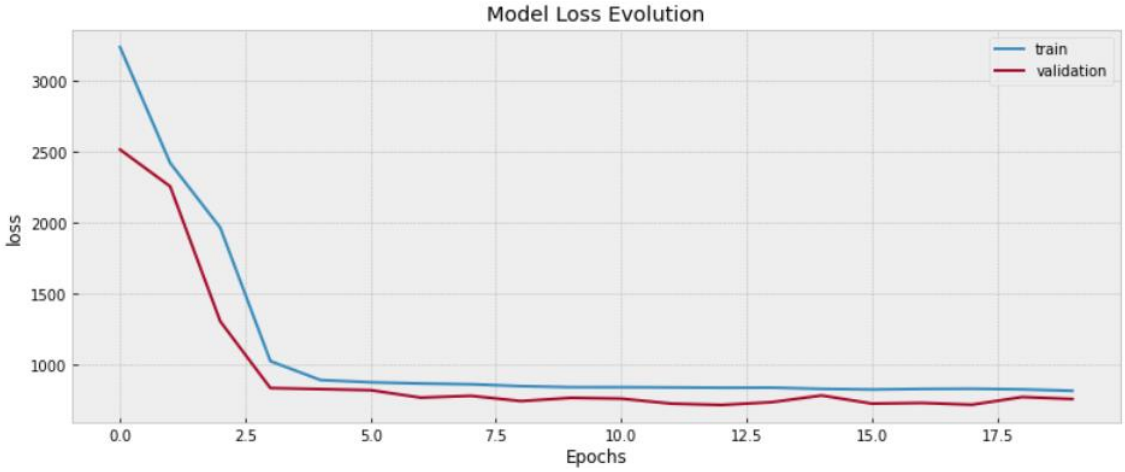
4.2.3. Operating conditions One-Hot-Encoding

The operating conditions were initially taken as values into the train and test data; however, it was confirmed that the separated conditions are found corresponding to the different flight conditions. Hence, in this section it will be tried to substitute these 3 columns by a one hot encoder to point out which specific flight condition is taking place at the moment of the sensor measurement.

Flight Condition_10.0	Flight Condition_20.0	Flight Condition_25.0	Flight Condition_35.0	Flight Condition_42.0
0	0	0	1	0
0	0	0	0	1
0	0	1	0	0
0	0	0	0	1
0	0	1	0	0

Figure 30 Addition of Boolean columns representing which condition is taking place (referring the value of the op. condition 1)

With this new dataset, the NN is performed again.




```
TRAIN METRICS for Model
 MAE:  18.456497192382812 MSE:  821.1900143599692 RMSE:  28.65641314540201 R2 score:  0.6900864105919666

TEST METRICS for Model
 MAE:  18.30120086669922 MSE:  964.6066480505816 RMSE:  31.058117265065853 R2 score:  0.6664752017823375
```

Figure 31 Metrics from One Hot Encoder MLP

The metrics are slightly worse with the addition of one-hot encoding; thus, the operational setting columns will be recovered.

4.2.4. Scaling operating conditions

With the observation of the sensor measurements representation, together with the distribution of the operating conditions rounded values, the user should have arrived at the idea that the different flight conditions do not happen depending on the engine. Actually, different flight conditions occur during the same flight for a single engine.

This causes the strange sensor distribution that seems more like a “seasonal” function. In order to have a better vision of this, let’s plot for a single engine a sensor measurement and dividing the different values by the flight operation that takes place.

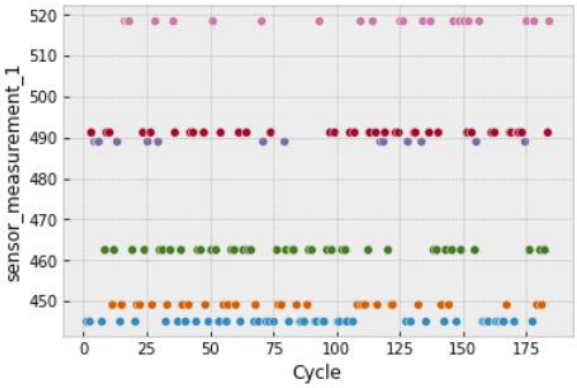


Figure 32 Random engine sensor measurement 1 evolution throughout cycles

Classified by color, the six different conditions are observed.Due to the range values differences, it is impossible to detect if the sensor general behavior is showing any change, just like previous notebook where it was possible to see if sensor measurement trend was positive or negative. In order to tackle this issue, the sensors will be scaled according to the operating condition using *StandardScaler.* The use of this scaling will lose the ranges of these sensors, and move them to values with mean=0, however, their differences will be, thus, minimized and made possible to compare their behavior. [17]

As an example, below graphs show the engines distribution for sensor measurement 2 before and after scaling:

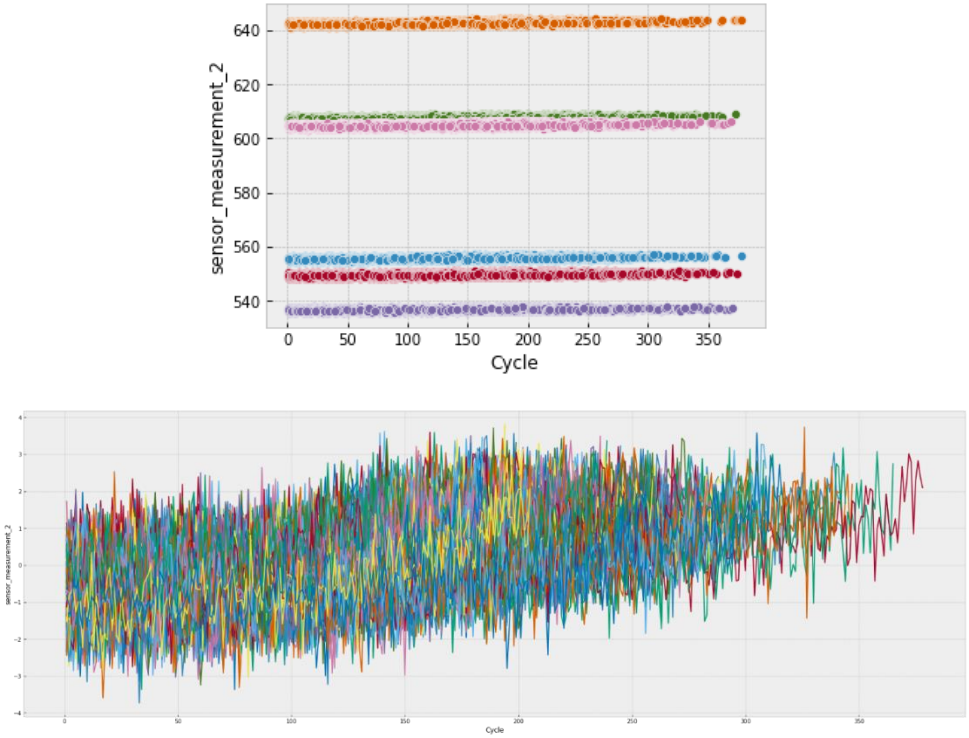
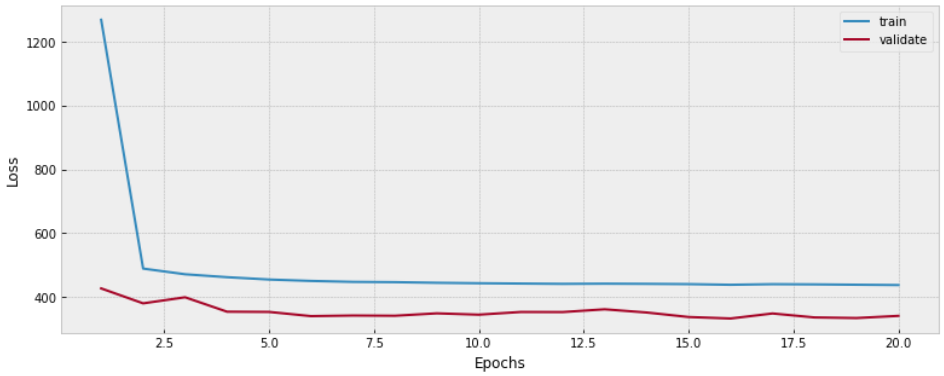


Figure 33 sensor_measurement_2 distribution per engine with separated operating conditions (above), same plot with scaled operating conditions (below)

Thanks to this grouping and scaling by the operating conditions, the sensor measurements behavior tends now to be more prone to extract trend insights. As commented in notebook, due to these specific trajectories, sensors 1, 5, 6, 10, 16, 18 and 19 are removed due to constant value throughout entire performance or extreme toggling behavior which would rely on worse predictions. With these new features, the model is tested again.



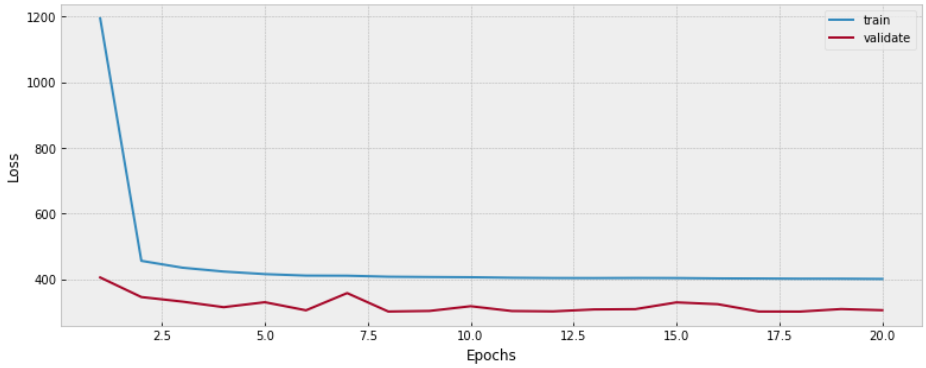

```
TRAIN METRICS for Model
 MAE:  11.264030456542969 MSE:  432.69673314446123 RMSE:  20.801363732805147 R2 score:  0.7706022250945936

TEST METRICS for Model
 MAE:  13.578105926513672 MSE:  836.9214886864205 RMSE:  28.92959537716386 R2 score:  0.7106239406474348
```

Figure 34 Metrics from standardized MLP

Metrics have been improved from baseline a 7.5%. This is a good step and following featuring will be applied to this new dataset.

4.2.5. Addition of Lagged Variables

As the dataset has been turned into a “timeseries” dataset with this last transformation, it can be applied what it was found to be beneficious in previous notebook, that is, the lagged variables. Initially, simply adding 1 level of lagged features provides following results:




```
TRAIN METRICS for Model
 MAE:  11.084419250488281 MSE:  395.30920042268923 RMSE:  19.882384173501155 R2 score:  0.7905286044021365

TEST METRICS for Model
 MAE:  13.134803771972656 MSE:  812.7711192820095 RMSE:  28.509140977623467 R2 score:  0.7189742325501149
```

Figure 35 Metrics from lagged standardized MLP

Very slight improvement found compared with no lagged variables, less than 0.01 points increase.

4.2.6. Smoothing dataset

Lagged variables have not provided as good results as expected, compared with previous notebook, and an additional feature is performed coupled with the lagged variables, that is the variable smoothing. Although the dataset contains now only defined trends in sensor measurements, it may lack the model's accuracy the little outliers of these functions. In order to reduce this "noise", a smoothing process is applied to the sensors, using the *.rolling(window)* tool, and moving these outliers to the mean of the feature values. [18]

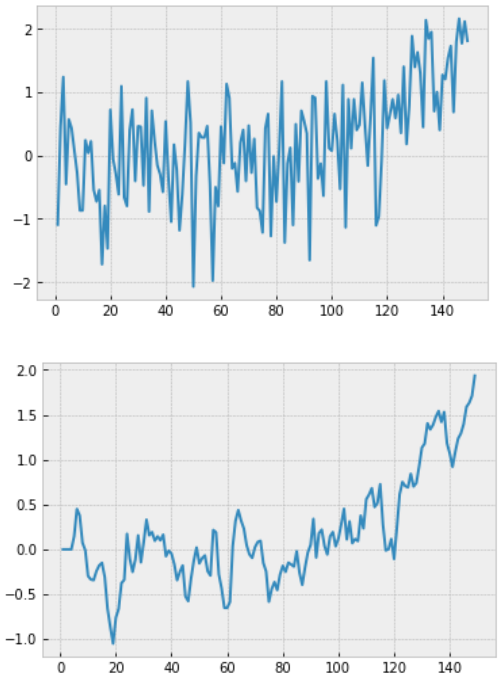
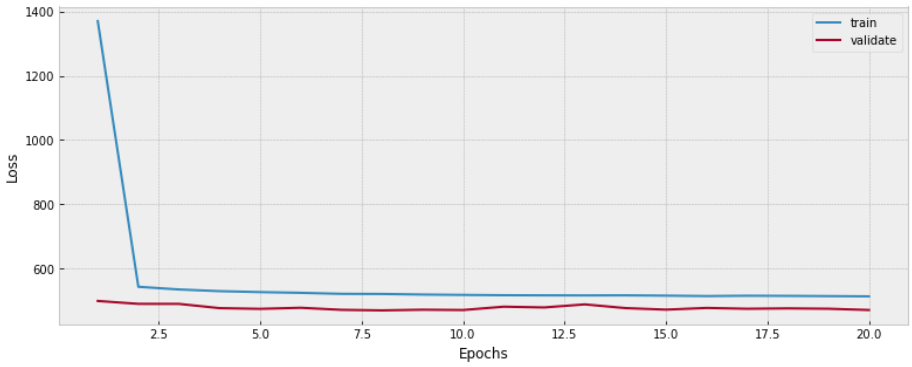


Figure 36 sensor measurement 2 before smoothing (above), and after smoothing with window=5

Once the sensor measurement columns have been smoothed according to this, new model metrics are computed:




```
TRAIN METRICS for Model
 MAE:  11.428146362304688 MSE:  511.4941359639327 RMSE:  22.616236113994137 R2 score:  0.7288271260691133

TEST METRICS for Model
 MAE:  13.001335144042969 MSE:  809.504696793069 RMSE:  28.451796020516333 R2 score:  0.7201036389290977
```

Figure 37 Metrics from smoothed MLP

Very slight improvement has been again obtained. This was also tried with a loop in different smoothing values, with no better results than showed, meaning that the higher smoothing for these non-lagged variables does not result in better accuracy. However, now it will be tried to implement the smoothing loop with lagged variables, looping the number of lags as well, and as an extra, let's try again with different RUL clips.

This will provide following results, from all the metrics obtained and saved in corresponding list, the best achieved are:

```
TRAIN METRICS for Model with clip 150
 MAE:  15.126651763916016 MSE:  609.5413077156813 RMSE:  24.68889037027953 R2 score:  0.7558978260448195
TEST METRICS for Model with clip 150
 MAE:  11.796882629394531 MSE:  722.1999621733578 RMSE:  26.87377833824931 R2 score:  0.7502903415154133
```

The best metric achieved so far, that was around 0.72 has been importantly improved with the combination of lagged variables and smoothing. Best score has been substituted by 0.75 using 1 extra lagged level of variables, with a window smoothing of 2 and an upper RUL limit of 150.

4.2.7. Hyper parametrization of Neural Network

So far, the several attempts performed have been related to modify the information income, by reducing or augmenting the dimensionality, looking for beneficial combinations or featuring the data format.

The last step of this study will use all these useful insights caught in previous sections, with the featuring of several parameters of the MLP. Some were assumed at the beginning based on basic examples provided in different internet sources, but now the dataset has been featured to its optimal shape, it is time to start iterating throughout different parameter values in order to find best metrics. [19]

The model parameters that will be featured will be: [14]

- Epochs: number of cycles which the model will be trained using training data, increasing this parameter may lead to better score but risks to overfit the model.
- Neurons per layer: tried with different linear combinations of previous layer
- Dropout: technique used to prevent overfitting, the number is the probability of a hidden neuron to be set to 0, so the higher value it takes, the lower risk to overfit.
- Batch size: defines the number of samples propagated through the network, chosen values similar to the neurons per layer length. When using lower number, it allows to train faster.
- Activation function: 3 different activation functions are used in this iteration process ("relu" (which was selected by default so far), "sigmoid" and "tanh"). Randomly using these three activations will provide a wider range of possible best scores chance.

```
number_of_iters = np.arange(1,101,100).tolist()
laggings = np.arange(1,10).tolist()
windows = np.arange(2,4)
dropouts = np.arange(0.1,0.5,5).tolist()
epoch_list = np.arange(20,51,10).tolist()
neurons_list = [[16, 32, 64], [32, 64, 128], [64, 128, 256], [128, 256, 512]]
activation_functions = ['tanh', 'sigmoid',"relu"]
batch_size_list = [32, 64, 128, 256, 512]


hyper_train_scores_lagg = []
hyper_test_scores_lagg = []

def create_model(input_dim, neurons, dropout, activation, weights_file = "weights_file_hyper.h5"):
    model = Sequential()
    model.add(Dense(neurons[0], input_dim=input_dim, activation=activation))
    model.add(Dropout(dropout))
    model.add(Dense(neurons[1], activation=activation))
    model.add(Dropout(dropout))
    model.add(Dense(neurons[2], activation=activation))
    model.add(Dropout(dropout))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer="adam")
    model.save_weights("weights_file_hyper.h5")
    return model
```

Figure 38 Model parameters preparation

The model is defined using a function which, at each iteration will provide final model with different parameters. The loop also considers different lagged levels and different window values for smoothing. At each iteration (100 different for each combination of lag and window) it also uses a random upper limit from the ranges used before.

```
for i in number_of_iters:

    mse_error = []

    epochs = random.sample(epoch_list, 1)[0]
    neurons = random.sample(neurons_list, 1)[0]
    dropout = random.sample(dropouts, 1)[0]
    activation = random.sample(activation_functions, 1)[0]
    batch_size = random.sample(batch_size_list, 1)[0]
    clip = random.sample(clips, 1)[0]

    X_train = training_2_scaled.drop(["ID","Cycle"]+op_settings, axis=1)
    y_train = df_train_2_lagg[~df_train_2_lagg.ID.isin(best_engines)]["RUL"].clip(upper=clip)
    X_test = df_test_sensors_lagg.groupby("ID").last().reset_index().drop(["ID","Cycle"]+op_settings, axis=1)
    y_test = df_RUL_2

    X_train_val = validation_2_scaled.drop(["ID","Cycle"]+op_settings, axis=1)
    y_train_val = df_train_2_lagg[df_train_2_lagg.ID.isin(best_engines)]["RUL"].clip(upper=clip)

    results_lagg = pd.DataFrame(columns=['MSE', # bigger std means less robust
                                'epochs',
                                'nodes', 'dropout',
                                'activation', 'batch_size', 'clip'])

    input_dim = len(X_train.columns)
    model = create_model(input_dim, neurons, dropout, activation, "weights_file_hyper.h5")


    model.compile(loss='mean_squared_error', optimizer="adam")
    model.load_weights("weights_file_hyper.h5")

    history = model.fit(X_train, y_train,
                        validation_data=(X_train_val, y_train_val),
                        epochs=epochs,
                        batch_size=batch_size,
                        verbose=0)
```

Figure 39 Iteration process MLP preparation

For each iteration, the values of the mean square error for each simulation using different random values from the descripted ranges are stored, together with those selected parameter. Each row builds up a dataframe with all these values, where at the end, sorted by the lowest

MSE, the best combination is selected to again run the model with the parameters from this low MSE.

```
    # append results
    results_appended = {'MSE':np.mean(mse_error),
         'epochs':epochs, 'nodes':str(neurons), 'dropout':dropout,
         'activation':activation, 'batch_size':batch_size, 'clip':clip}
    results_lagg = results_lagg.append(pd.DataFrame(results_appended, index=[0]), ignore_index=True)
best = results_lagg.sort_values("MSE").iloc[0]

nodes = best["nodes"][1:-1].split(",")
activ = best["activation"]
drops = best["dropout"]
batchs = best["batch_size"]
epoch = best["epochs"]
clipp = best["clip"]


y_train = df_train_2_lagg[~df_train_2_lagg.ID.isin(best_engines)]["RUL"].clip(upper=clip)
y_train_val = df_train_2_lagg[df_train_2_lagg.ID.isin(best_engines)]["RUL"].clip(upper=clip)

model = Sequential()
model.add(Dense(int(nodes[0]),input_dim=input_dim, activation=activ))
model.add(Dropout(drops))
model.add(Dense(int(nodes[1]), activation=activ))
model.add(Dropout(drops))
model.add(Dense(int(nodes[2]), activation=activ))
model.add(Dropout(drops))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer="adam")
model.save_weights("weights_file_hyper.h5")

model.compile(loss='mean_squared_error', optimizer="adam")
model.load_weights("weights_file_hyper.h5")

history = model.fit(X_train, y_train,
                    validation_data=(X_train_val, y_train_val),
                    epochs=epoch,
                    batch_size=batchs)
```

Figure 40 Final MLP preparation with best parameters, product of the iteration of chosen lag and window

As in the previous picture, the best parameters are fitted in the MLP, and this is computed and its metrics are stored, to at the end be compared with the rest from this overall study.

```
TRAIN METRICS for Model
 MSE:  481.6816995464166 R2 score:  0.7803035661145481
TEST METRICS for Model with clip 140 with following features:
 nodes: ['128', ' 256', ' 512']  activation: relu  dropout: 0.1  batch size: 256  and epochs: 20
 MSE:  727.7564808819637 R2 score:  0.7483691057611027
```

Althought these are quite better results compared with baseline ones, it is true that it was expected far better metrics with all this iterative process. In the end, the best score achieved was the one reached by the simple combination of lagged and window with RUL clip, without modifying any parameter from NN.

This might be explained because maybe it was needed higher number of iterations, maybe the use of dropout and batch size allowed to maintain train scores below 0.8 but causing not as good as before test score; or perhaps simply this MLP model was not able to provide better results.

In any case, the best model developed in this study is the MLP with initial parameters:

- 4 layers of 16, 32, 64 and 1 nodes
- Without dropout
- Without batch size
- Activation "relu"
- 20 epochs

Together with 1 extra level of lagged variables, 2 for window and RUL limited to 150. This achieves **0.755** for train score and **0.750** for test score, and turns to be the best model developed in this study to forecast remaining useful lifes for engine performing at 6 different flight operations.

5. Third set of engines (FD003) - Regression

In the first dataset, the engine's fleet conditions are the following:

- 100 train and 100 test trajectories
- 1 Flight conditions (Sea Level)
- 2 Types of engine fault (by HPC and Fan degradation)

As exactly performed before, variables analysis is performed.

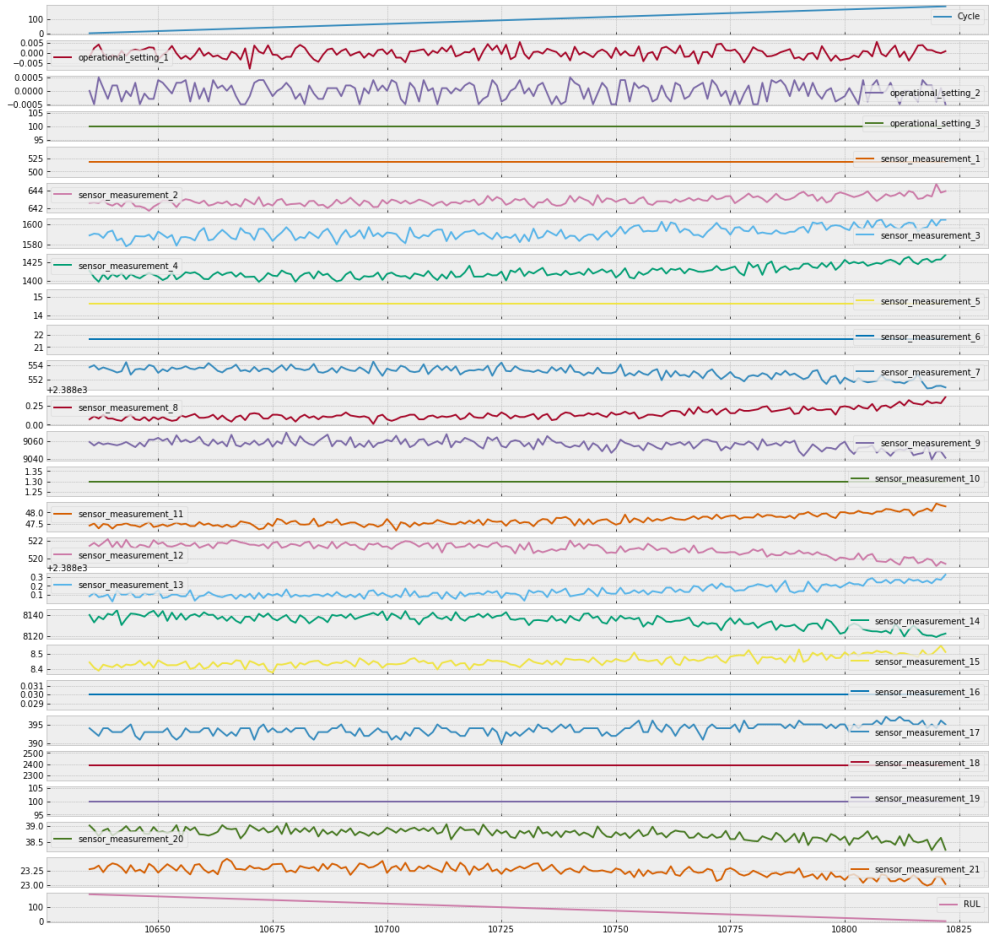


Figure 41 FD003 columns performance of single engine

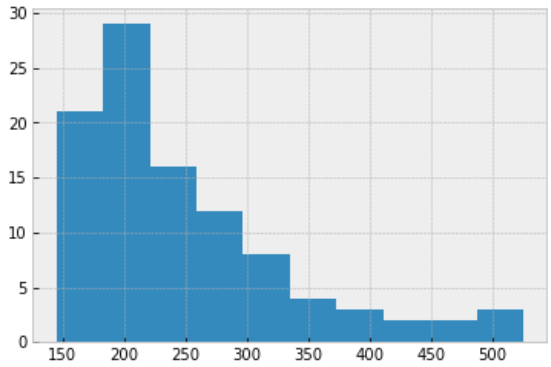


Figure 42 Average max cycles value per engine in FD003

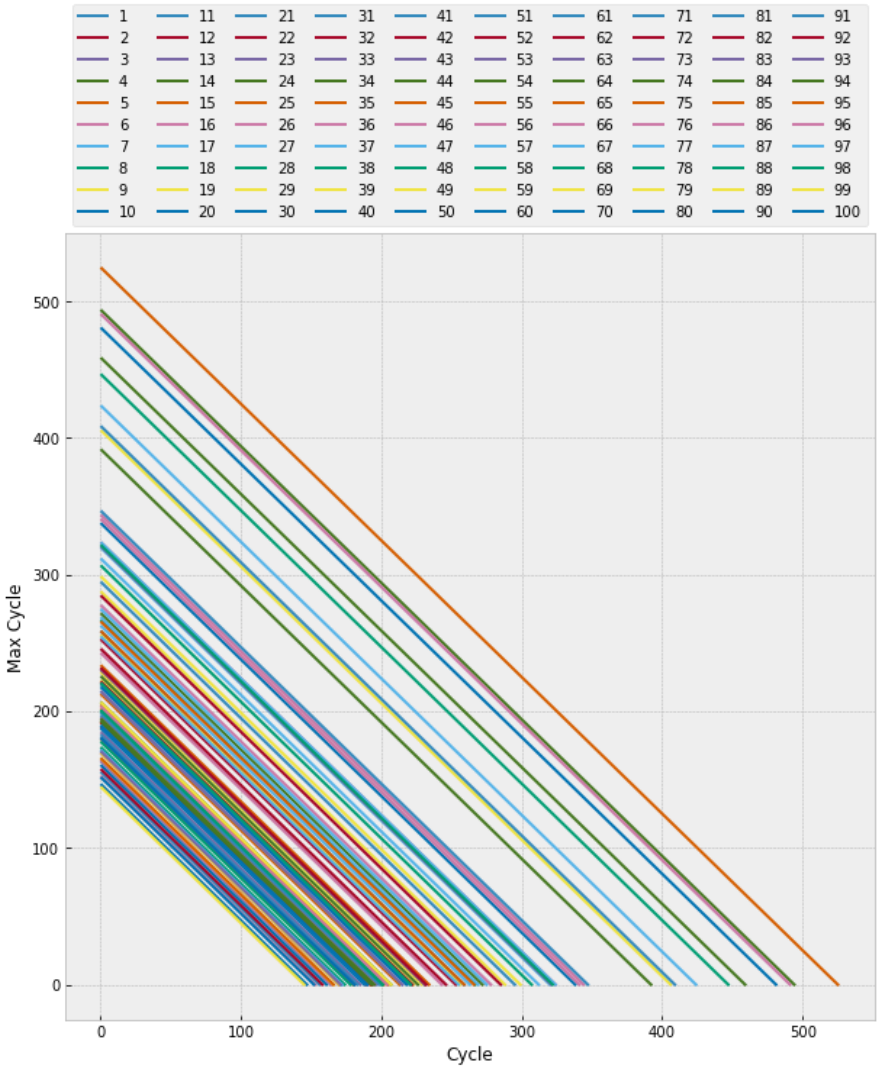


Figure 43 FD003 RUL variation

With these 3 plots above showed, following conclusions may be obtained:

- Sensor's performances seem more similar to the ones from FD001, as the flight condition is again only one, there are no different operating ranges throughout the same engine performance.
- Average max engine life is higher than previous 2 sets, meaning that the second type of fault might occur at later cycles.
- This is also observed in the RUL variation graph, where it could be observed a division in the Max cycle per engine above and below 350 cycles.

Data Analysis

Same process performed in previous sets, in this case, the features picture seems familiar due to its closeness to FD001. Starting by the operational settings, it is again detected that operational_setting_3, as it is directly linked to flight operation type, is constant.

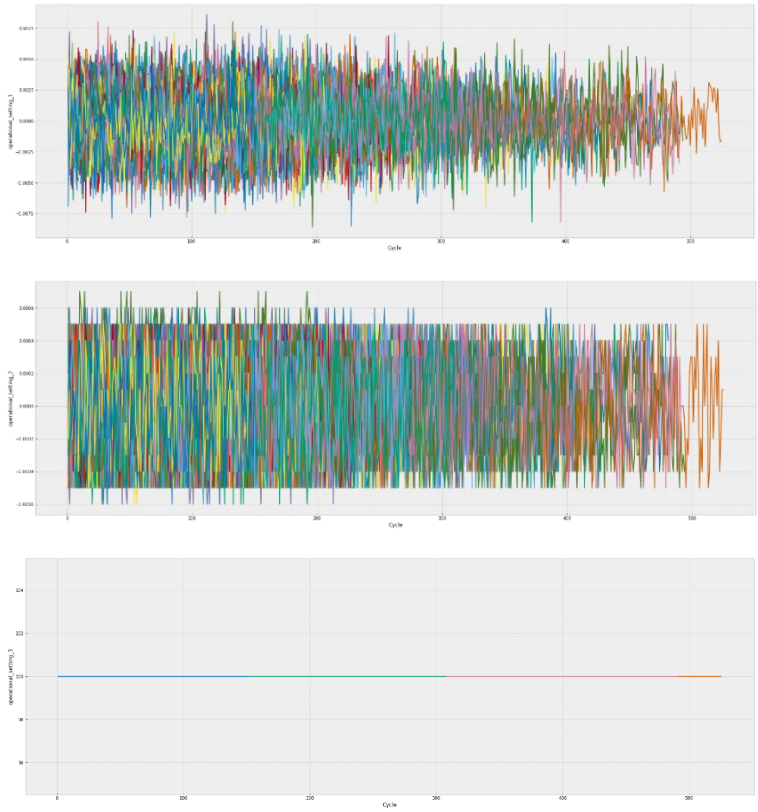


Figure 44 Operational settings 1, 2 and 3 behaviors in FD003

Sensor's trajectories show both trends and constant behaviors, comparable to FD001.

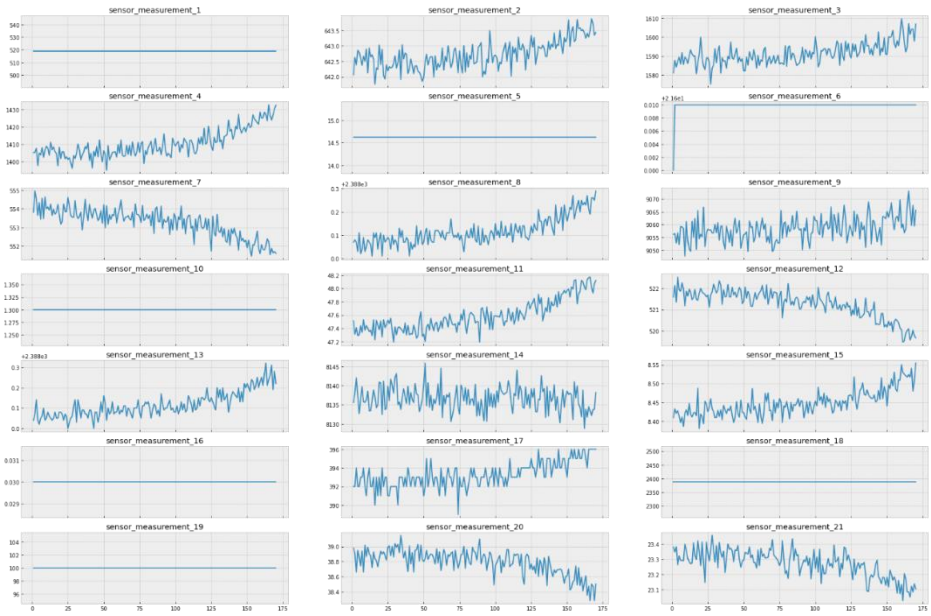


Figure 45 Sensor's measurements behaviors in FD003

All coupled these plots, together with correlation matrix (where three strong correlations were found) the dataset in order to proceed with the regression problem will remove the following features:

- Operational setting 3
- Sensor measurements 1, 5, 6, 7, 8, 10, 14, 16, 18, 19

It is observed that the number of initially removed features is higher than FD001, which means that some features could be useful and provide information for the classification problem, however, when at this point what it is being aimed is to find potential trends, these shall be removed.

5.1. Regression models development

In this notebook, due to the initial similarity with FD001 (real difference comes by the classification problem) the 3 different models explained in FD001 will be deployed again. During the feature engineering process of this notebook, saving some exceptions the same steps are taken to improve the baseline metrics:

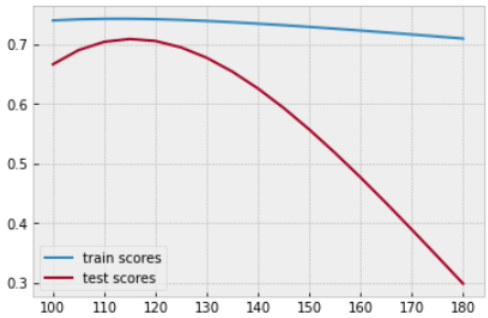
4. Baseline model (coupled with RUL limit)
5. Reduce Dimensionality
    i. Models with top features by slope value & RUL limited
    ii. Principal Component Analysis (combined with RUL limited)
6. Augment Dimensionality
    i. Addition of lagged variables (combined with RUL delimited)
    ii. Addition of polynomial variables (combined with RUL limited)

5.1.1. Baseline Model

As expected, just simple baseline models do not perform quite good, so from this point the analysis will start by directly applying a loop of different RUL limits for each model.

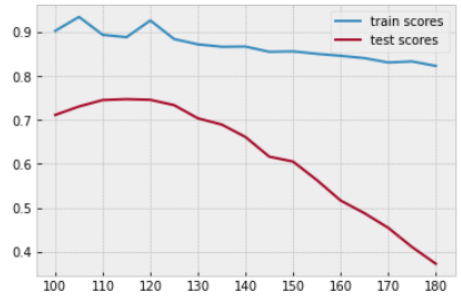
- Linear Regressor + RUL limit variation

```
Train Score from max Test Score:  0.7428030957935234
 Max Test Score:  0.7089574969427694
 Achieved with 115 upper clipped RUL
```



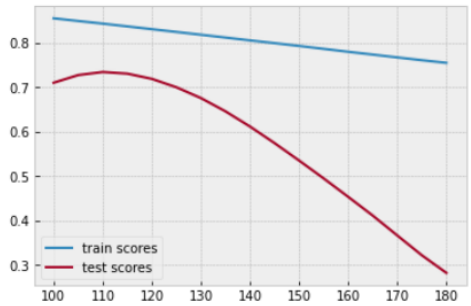
- Random Forest + RUL limit variation

```
Train Score from max Test Score:  0.8889775121499941
 Max Test Score:  0.7475304696160692
 Achieved with 115 upper clipped RUL
```



- SVR + RUL limit variation

```
Train Score from max Test Score:  0.8437499766203382
 Max Test Score:  0.7347840534356809
 Achieved with 110 upper clipped RUL
```



It is demonstrated again the rapid benefits of using a RUL limited from above. In this case it has been already reached a maximum test score of 0.747 with the *RandomizedSearchCV* of the Random Forest, using a limit in 115 cycles.

5.1.2. Top Features

As performed in the previous notebook, it was confirmed that using a smaller number of features from the dataset could provide better results by reducing the dimensionality of the dataset. The strategy to select these top features will be the same as before, that is, based in the slope value of each one.

In this case, aiming to reduce the number of steps, it will be performed at the same time the loop of different features included in set and the different limit values for RUL. Hence, results for the double loop are presented below:

- Linear Regression + Top features + RUL limit variation (best at 115)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 10
Best Train Score:  0.740542559192096  Best Test Score: 0.7127864127370126
```

- Random Forest + Top features + RUL limit variation (best at 115)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 11
Best Train Score:  0.8755883859714179  Best Test Score: 0.7534291557648132
```

- SVR + Top features + RUL limit variation (best at 110)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 16
Best Train Score:  0.8437499766203382  Best Test Score: 0.7347840534356807
```

As happened in FD001, it has been noticed some improvement with this dimensionality reduction, the three models have slightly increased their test scores, reaching this time the RandomForest a r2 test score of 0.753.

5.1.3. Principal Component Analysis

Keeping the line of reducing dimensionality, and as a result of the good results obtained in FD001, the PCA will be applied again. The procedure was detailed in first chapter, hence, now only results will be exposed.

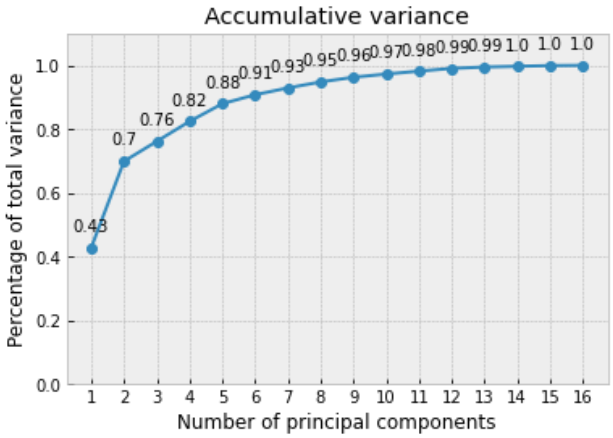


Figure 46 Accumulative variance from components selection in FD003

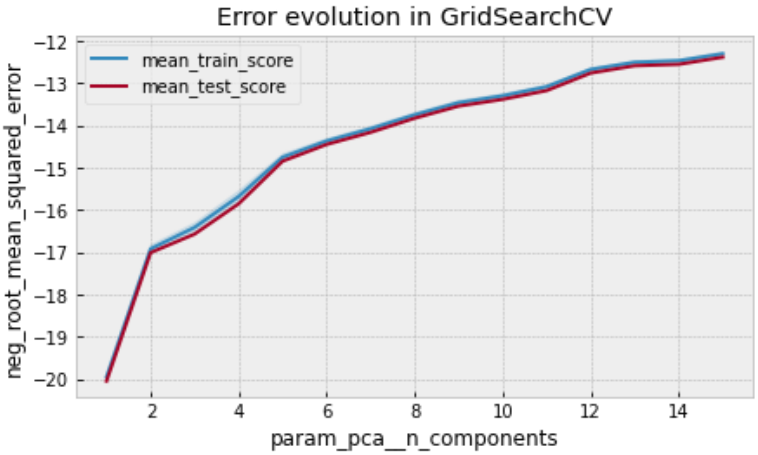


Figure 47 Error evolution with the component's selection in FD003

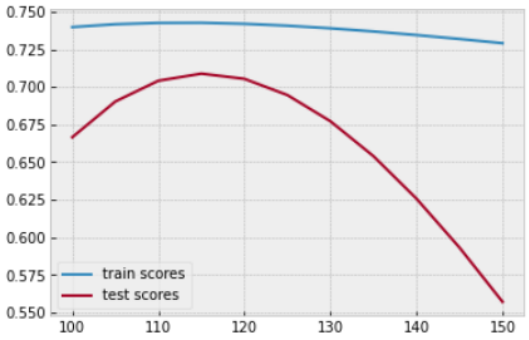
With the above study, the PCA final chosen number of features is:

```
Best number of components found:  {'pca__n_components': 15}
```

Again, there is no much of dimensionality lost due to PCA introduction, as all of the variables in this set are important to determine insights, at least it has been removed 1 of the components. Now, the 3 models will be tested with this PCA, together with the RUL limit variation:

- Linear Regression + PCA + RUL limit variation

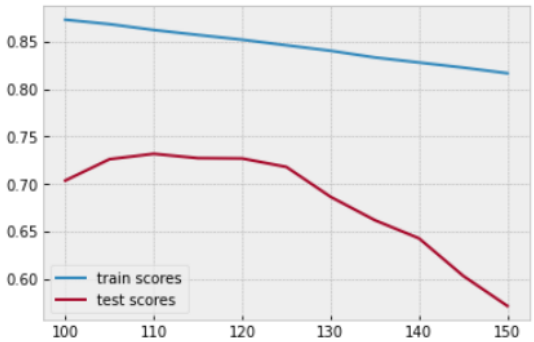
```
Train Score from max Test Score:  0.7427306632687565
 Max Test Score:  0.7088956861632699
 Achieved with 115 upper clipped RUL
```



- Random Forest + PCA + RUL limit variation

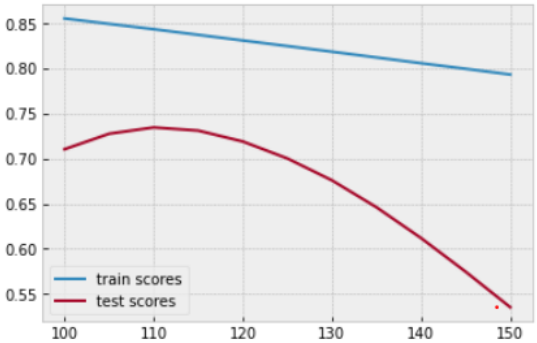
  (Due to time limitations, the RandomizedSearchCV was shortened including only the optimal values obtained from RF with PCA and fixed RUL limit)

```
Train Score from max Test Score:  0.862488824140664
 Max Test Score:  0.7318207941025461
 Achieved with 110 upper clipped RUL
```



- SVR + PCA + RUL limit variation

```
Train Score from max Test Score:  0.8437356949635394
 Max Test Score:  0.7346514897213334
 Achieved with 110 upper clipped RUL
```



Results are better than baseline ones, but do not reach the metrics reached by the combination of top features and RUL limit. Thus, let's change now the strategy and start performing featuring for increase dimensionality.

5.1.4. Addition of Lagged Variables

The direct use of 1 level of lagged variables addition has provided good results in the past, in this case, provides the following results:

- Linear Regression + Lagged Variables + RUL limit

```
TRAIN SCORES - - - -
 MAE:  13.485765698591422
 MSE:  402.6900385673843
 RMSE:  20.067138275483735
 R2 score:  0.7563974317077706
TEST SCORES - - - -
 MAE:  14.784431094583852
 MSE:  508.27932837995627
 RMSE:  22.54505108399527
 R2 score:  0.7033846637157077
```

- Random Forest + Lagged Variables + RUL limit

```
TRAIN SCORES - - - -
 MAE:  4.699830428932827
 MSE:  137.71215709658156
 RMSE:  11.73508232167894
 R2 score:  0.9166926619959669
TEST SCORES - - - -
 MAE:  12.529298856935306
 MSE:  464.8377129195627
 RMSE:  21.560095382895753
 R2 score:  0.7287357820064859
```

- SVR + Lagged Variables + RUL limit

```
TRAIN SCORES - - - -
 MAE:  5.087233365238962
 MSE:  265.14579750741234
 RMSE:  16.283298115167344
 R2 score:  0.839603190894704
TEST SCORES - - - -
 MAE:  11.784173742647006
 MSE:  524.014485759725
 RMSE:  22.89136268900838
 R2 score:  0.6942021360442353
```

As in first notebook study, the addition of lagged variables provides fairly good results, however, the risk of overfitting starts to appear in the case of RandomForest, so let's try to combine these new features appearing with a previous filter which demonstrated good results, that is the top features by slope. In this case, the order may change due to some features which lagged version provides practically same slope as real ones.

For the case of Random Forest and SVR, due to the longer time it takes to run these simulations, the minimum number of features chosen to find the best final number was 15, not one, based on the results from Linear Regression. Let's check the results:

- Linear Regression + Lagged Variables + Top Features + RUL limit variation (120)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 18
Best Train Score:  0.751228423139392  Best Test Score: 0.7140586712340399
```

- Random Forest + Lagged Variables + Top Features + RUL limit variation (120)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 20
Best Train Score:  0.8906888880936629  Best Test Score: 0.7646980464121096
```

- SVR + Lagged Variables + Top Features + RUL limit variation (110)

```
BEST METRICS for combination of top features and upper RUL clip:
Features number: 24
Best Train Score:  0.848295412541132  Best Test Score: 0.7471085913102131
```

Important score increase has been achieved thanks to this additional loop, reaching a test score of 0.764 with Random Forest regressor, with a level or lagged variables, filtered by the top 20 in terms of slope and with a RUL limit of 120.

5.1.5. Polynomial Features

The last step in this chapter finishes, similarly to FD001, with the addition of polynomial features as a dimensionality increase resource, due to the good results obtained in the previous step. Hence, polynomial combination of second degree of the dataset features is applied, together with the RUL limit, providing the following results:

- Linear Regression + Polynomial Features + RUL limit

```
TRAIN SCORES - - - -
 MAE:  12.121377497911453
 MSE:  341.5265181990029
 RMSE:  18.480436093312377
 R2 score:  0.7930764944189086

TEST SCORES - - - -
 MAE:  15.365231037139893
 MSE:  468.21644047371257
 RMSE:  21.638309556749405
 R2 score:  0.7267640661531549
```

- Random Forest + Polynomial Features + RUL limit

```
RANDOM FOREST TRAIN SCORES - - - -
 MAE:  5.254846803393978
 MSE:  178.4998487270526
 RMSE:  13.360383554638416
 R2 score:  0.8918508154532989

RANDOM FOREST TEST SCORES - - - -
 MAE:  12.890054674902544
 MSE:  434.8214261919725
 RMSE:  20.852372195795194
 R2 score:  0.7462523137334154
```

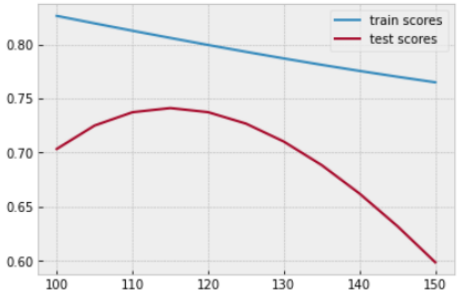
- SVR + Polynomial Features + RUL limit

```
MAE:  5.425135841073185
MSE:  291.79307015974524
RMSE:  17.081951591072528
R2 score:  0.8232089112724713

MAE:  11.890778211702692
MSE:  506.9291941881846
RMSE:  22.515088145245727
R2 score:  0.7041725582551093
```

Fairly good results, but not very close to the best ones achieved so far. Let's apply one more final loop, with different RUL limits to check if the best metric can be beaten.

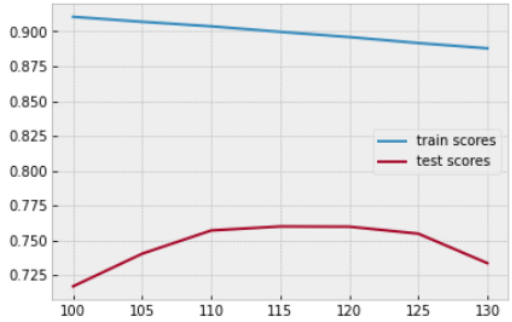
- Linear Regression + polynomial features + RUL limit variation

```
Train Score from max Test Score:  0.805908668443573
 Max Test Score:  0.741134606375021
 Achieved with 115 upper clipped RUL
```



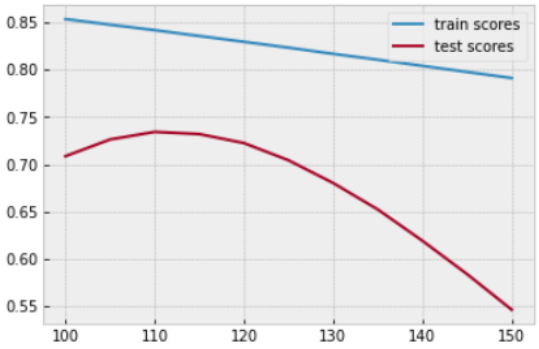
- Random Forest + polynomial features + RUL limit variation
  (Again, due to time limitations, the RandomizedSearchCV parameters will be the best acquired from the fixed RUL limit model before)

```
Train Score from max Test Score:  0.89971081420018
 Max Test Score:  0.7599274081478097
 Achieved with 115 upper clipped RUL
```



- SVR + polynomial features + RUL limit variation

```
Train Score from max Test Score:  0.8417142977288518
 Max Test Score:  0.7342276815930018
 Achieved with 110 upper clipped RUL
```



This last loop has increased overall previous results from the three models, compared with fixed RUL. The best metric achieved previously was not beaten (with very small difference) and started to be close to overfit in the case of Random Forest. It is true that, if the GridSearch would have been entirely applied again, better metrics could have been seen, in any case, this FD003 regression problem ends with a notable final accuracy from the following final mode, determined as the best option in this study to be applied for engines performing at 1 flight condition (Sea Level) and expected to fall in two different failure categories (Fan and HPC deterioration):

- A random forest regressor, optimised with a RandomizedSearchCV
  - Max_depth = 20
  - Min_samples_leaf = 17
  - N_estimators = 93
- Featured with lagged variables, filtered with top 20 features by slope.
- Limiting the RUL from above at 120 cycles

This provides **0.890** r2 score for train set and **0.764** for test score.

6. Third set of engines (FD003) - Classification

Further from the regression problem performed in previous chapter for FD003, as this dataset contains 2 different fault modes, in this chapter a classification investigation will be performed, trying to determine features and potential behaviors that may suggest whether the engine is falling into one type of fault or another. With the obtained insights, at the end it will be trained a classification model that will be tested on the test set of FD003. As no real values for this modelling are provided in the data, only potential conclusions will be achieved.

Initially, this research will be based in the sensors that were removed from dataset for the regression problem, but right now could provide useful interpretations, and the initial characteristics of this dataset feature distribution observed in previous chapter.

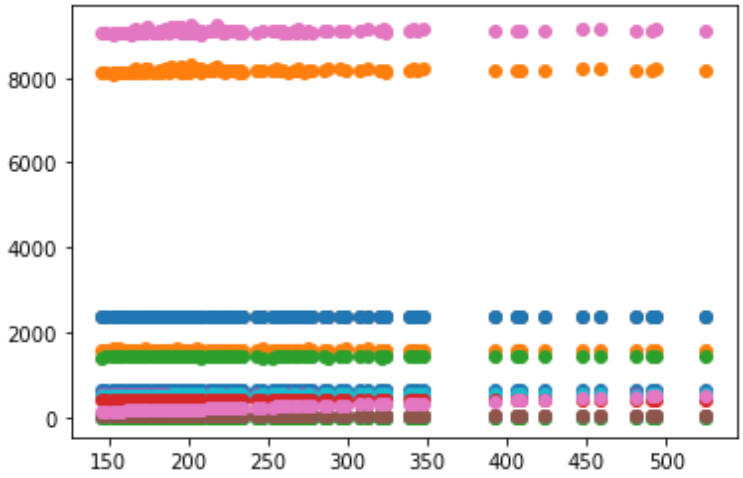


Figure 48 Last cycle measured sensor of each engine in FD003 train

For example, in the above plot, it can be noticed a particular separation between last cycle of each sensor from the engines of the set. There is a clear split in the range between 350-400 cycles which, initially, could be linked to the type of fault.

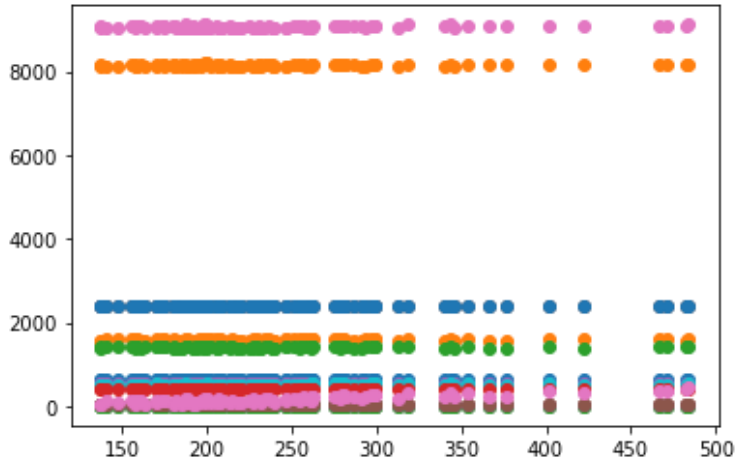
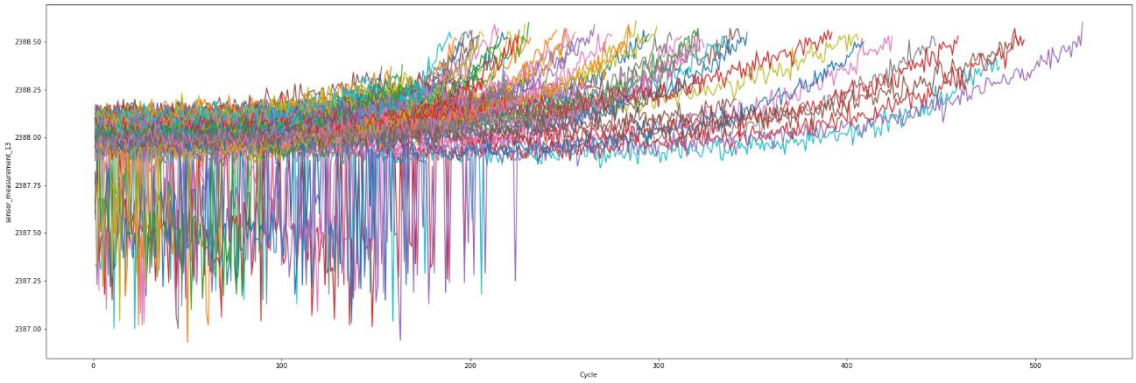


Figure 49 Last cycle measured sensor of each engine in FD003 test + real RUL

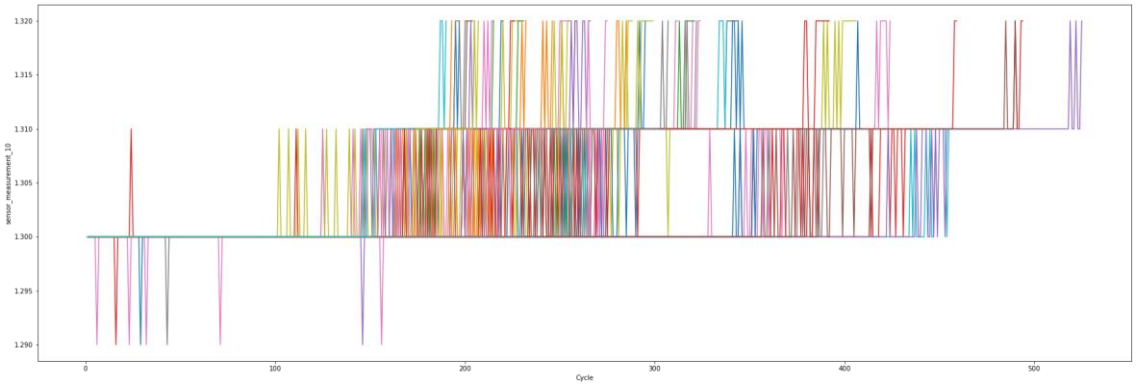
Nevertheless, same plot was taken adding the max cycles from test set to the real RUL of each engine (having then, maximum total life of each engine) and this separation was not that clear.

From now on, exhaustive analysis of the sensor measurements dropped in previous chapter will be performed. On each feature, the distribution will be analyzed, potential categorization will be studied, and possible link with the maximum life will be considered. These sensors are:

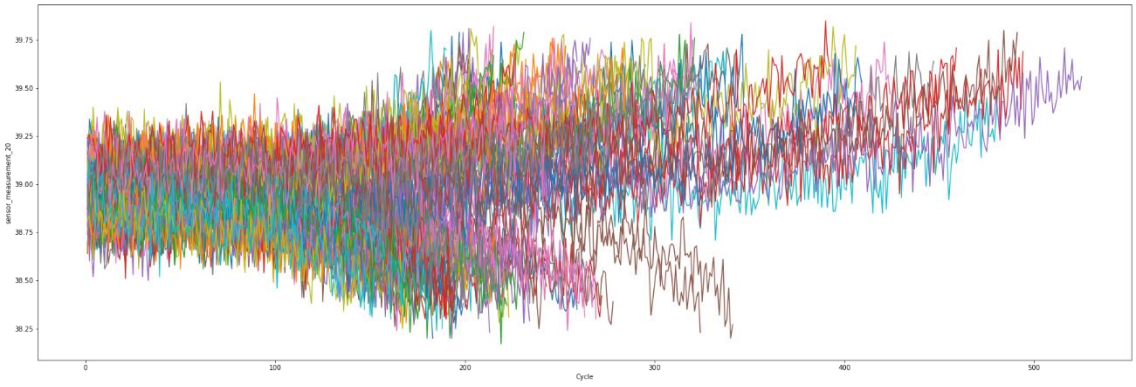
- Initially, sensors 8 and 13 which have similar trend trajectory but with several cases of toggling:



- Sensor 10 which shows a particular trajectory with many constant values and some steps:



- Last, sensors 12, 14, 15, 20 and 21 which show 2 clear trajectories, one with positive trend and the other with a negative one:



This analysis is extensively explained in the notebook, and aiming to not lengthen much more this document, only the key elements of each group of sensors analyzed will be exposed in the following sections.

6.1. Sensors Analysis

6.1.1. Sensor measurements 8 and 13

Both sensor's behaviors are practically identical and hence, they will be explained together.

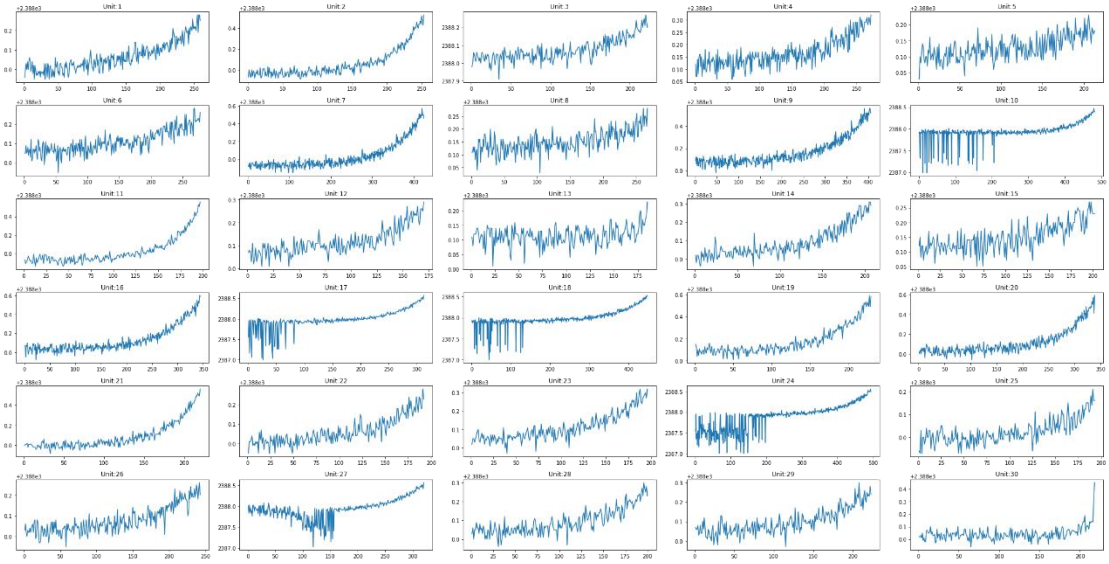


Figure 50 Sensor measurement 8 sample of engines

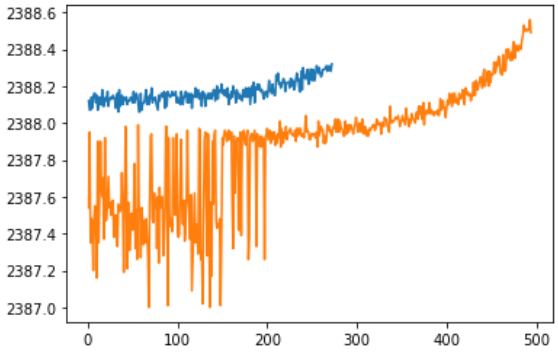


Figure 51 2 sensor measurement 8 plot of two engines with different behavior

As commented before, two different styles were found in these sensors, one with simple increasing trajectory and another preceded by a toggling beginning. It is observed as well that the engines with toggling behavior tend to reach higher number of cycles.

In order to obtain clear division, the minimum value reached by this sensor is used as a threshold to distribute the engines in the two groups assumed.

```
engines from group 1: 87
[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 19, 20, 21, 22, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 4
0, 41, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 69, 70, 72, 74, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

engines from group 2: 13
[10, 17, 18, 24, 27, 37, 42, 55, 62, 71, 73, 75, 89]
```

```
sensor_8_G1 == sensor_13_G1, sensor_8_G2 == sensor_13_G2
(True, True)
```

Figure 52 Engine group distribution by sensors 8 and 13

Once these groups have been confirmed, let's check any potential link between these categories (again based on the minimum value reached during performance) and the maximum life of engine.

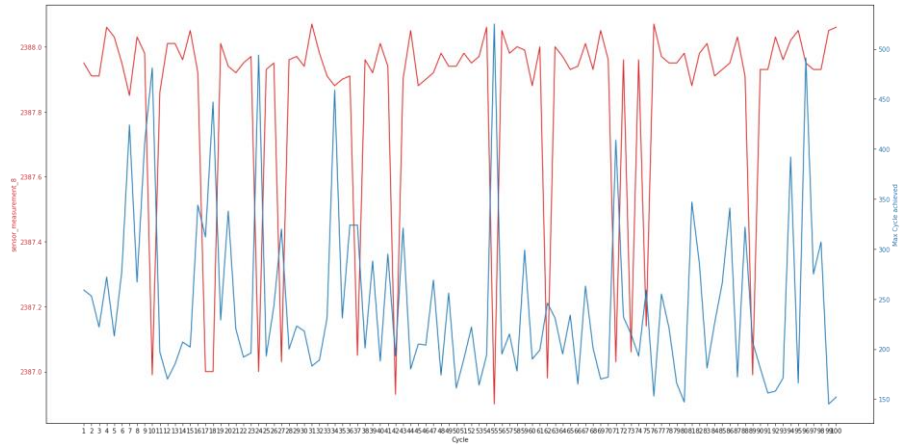


Figure 53 Relationship between engine minimum sensor 8 value achieved and maximum cycle

Observing this graph above, it is possible to detect some cases where exist a direct link when the engine reaches a low value of sensor 8 (saying this engine belongs to group 2) and reaches a high number of cycles; however, not strong enough to confirm this hypothesis.

6.1.2. Sensor Measurement 10

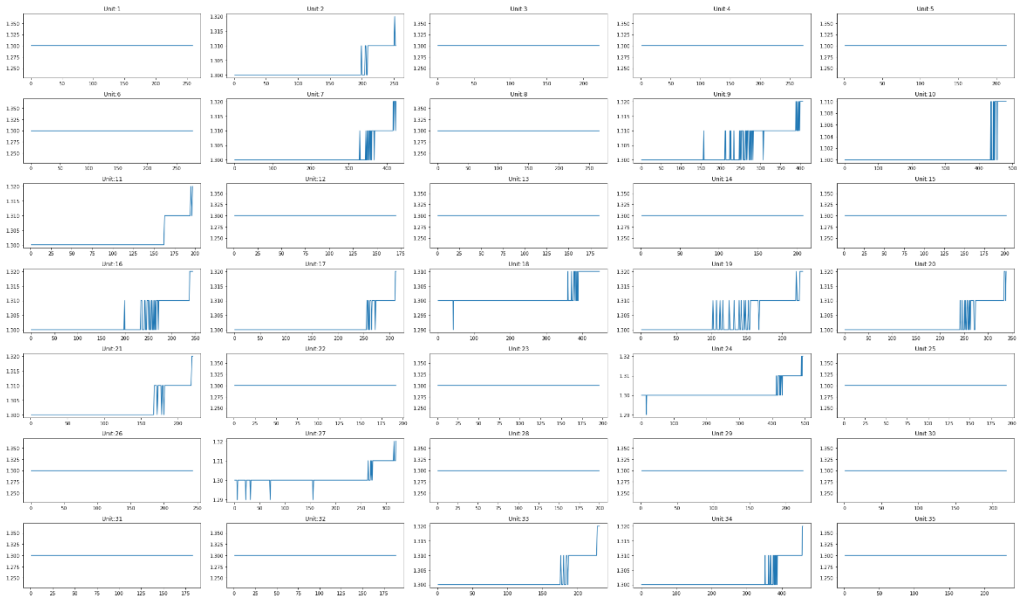


Figure 54 Sensor measurement 10 sample of engines

Two different behaviors observed in the dataset, the first one is a completely constant value through entire performance, and the other shows some immediate vertical steps with some toggling cases. Thus, a good way to split this set in two groups is to look at the standard deviation:

```
sensor_10_G1 = constants_s10
sensor_10_G2 = not_constants_s10

print(len(not_constants_s10), len(constants_s10))
```

```
44 56
```

More equal distribution is found with this sensor, compared with previous 2, let's check now the potential link with max cycles achieved.

```
Max_cycle achieved by constant s10 engines:  341
 Max_cycle achieved by NON-constant s10 engines:  525
Mean max_cycle value of constant s10 engines:  202.05357142857142
 Mean max_cycle value of NON-constant s10 engines:  304.65909090909093
```

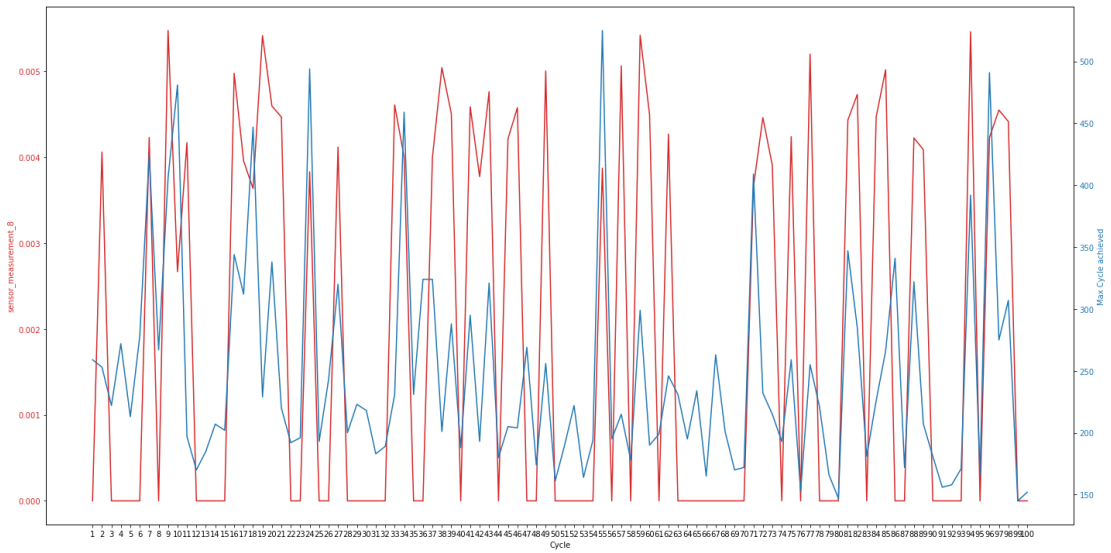


Figure 55 Relationship between engine sensor 10 value achieved and maximum cycle

There is an important difference in terms of maximum achieved number of cycles between groups, and in this case, it is noticed a stronger link with this feature (easily checked when looking at the "valleys" of the sensor 10.

6.1.3. Sensor Measurements 12, 14, 15, 20 and 21

Although these sensors do not finally result all in the same way, it is true that their behavior is practically identical, and they can be studied at once. Their one-by-one analysis can be found again well explained in the notebook.

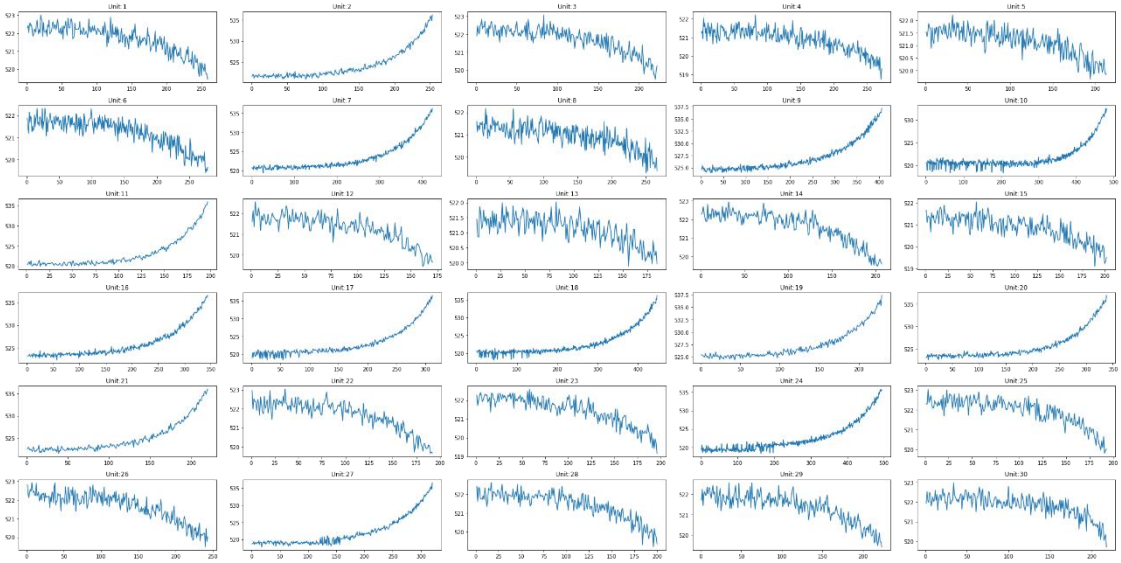


Figure 56 Sensor measurement 12 sample of engines

As commented at the beginning of this chapter, for these sensors the key distinction is the trend of the trajectories. Each sensor has different engines with increasing or decreasing sensor functions, thus, a good way to split each sensor distribution is to classify the slopes by positive or negative values. After so, following groups were obtained:

- Sensor 12

```
In total, there are separated groups of positive and negative trend Engines; which are
 Trend
Negative    56
Positive    44
```

- Sensor 14

```
In total, there are separated groups of positive and negative trend Engines; which are
 Trend
Negative    24
Positive    76
```

- Sensor 15

```
In total, there are separated groups of positive and negative trend Engines; which are
 Trend
Negative    44
Positive    56
```

- Sensor 20

```
In total, there are separated groups of positive and negative trend Engines; which are
 Trend
Negative    56
Positive    44
```

- Sensor 21

```
In total, there are separated groups of positive and negative trend Engines; which are
 Trend
Negative    56
Positive    44
```

All these sensors, excepting sensor 14, coupled with sensor 10 have the same distribution. It is time now if these distributions keep the same engines each and not only the number:

```
set(sensor_12_G1 == sensor_10_G1), set(sensor_12_G2 == sensor_10_G2)
```

```
({True}, {True})
```

```
set(sensor_15_G1 == sensor_10_G2), set(sensor_15_G2 == sensor_10_G1)
```

```
({True}, {True})
```

```
set(sensor_20_G1 == sensor_10_G1)
```

```
{True}
```

```
set(sensor_21_G1 == sensor_10_G1)
```

```
{True}
```

It is confirmed that distributions from sensors 10, 12, 15, 20 and 21 are identical.

In the notebook of the FD003 classification is extensively explained the assumed theory behind the different sensors studied behavior and how it is affected by the dataset conditions and fault modes nature.

In addition, due to the final distribution of the train set of engines in 2 groups: group 1 for Fan degradation fault and group 2 for HPC degradation fault, it was confirmed the difference in terms of maximum life reached by engines:

```
Average Max Cycle reached when Fan Fault Class 304.65909090909093

Average Max Cycle reached when HPC Fault Class 202.05357142857142
```

6.2. Classification Model

Once the potential distribution of the dataset has been assumed. The last to do is to perform a basic classification model with the train set of FD003 and test it in the test set.

- Fan Fault Mode = 1
- HPC Fault Mode = -1

Train-test split using the train set of engines was used to train the model, with last 30 engines as test size. 2 different models were basically used for this, with following accuracies:

- Logistic Regressor: 0.94
- Random Forest Classifier: 1.00

The model obviously presents high accuracy, as it is being tested in the dataset where its distribution was extracted. However, this is assumed as a good signal of a good choice of engines group split.

Last, this trained model is tested on the test set, with no actual objective (as no real classification is provided in the data) but to test which kind of distribution is predicted.

```
list_engines = results_test.groupby("ID")["Fault_Class"].mean().index
list_faults = results_test.groupby("ID")["Fault_Class"].mean().values

fault_Fan_engines = []
fault_HPC_engines = []
for engine, fault in zip(list_engines, list_faults):
    if fault > 0:
        fault_Fan_engines.append(engine)
    else:
        fault_HPC_engines.append(engine)
```

```
len(fault_Fan_engines), len(fault_HPC_engines)
```

```
(46, 54)
```

Classification model predicts that engines from test set will fail into the two fault categories in the following distribution: 46 in the Fan Degradation and 54 in the HPC degradation.

This is quite similar to the 44 / 56 engines distribution from train set, so it could be assumed that, as both train and test sets work in the same flight condition and fall into same fault categories, it is not a bad result.

7. Conclusions and Next steps

During this project several models and featuring steps have been developed in order to find the best approach to forecast the remaining useful life of different sets of engines working in different flight conditions and falling into different fault modes.

It has been intended that, in each notebook study, different methods could be used, however, some of them were constantly applied due to the good results taken.

It has been observed that, between reducing and augmenting the dimensionality of the test set, the best approach has been to, initially, increase the number of features by applying featuring such lagged variables or polynomial features, and afterwards moving away from overfitting and try to increase the test score by filtering somehow the information (applying slope value filtering and limiting the RUL for example).

Overall scores above 0.75 have been achieved in this project. It is true that much project is left behind and there is much more work that could be applied, many more variable's loops iteration, further featuring, many more models to be applied… But the main intention from this project, objectively, was to get a first contact with Data science projects from scratch and start building a new way of research and working with these types of ventures.

For further steps, apart from increasing the range of *grisearch* for the Random Forest Regressor, which has demonstrated to be the best applicable model in this study, it could be applied more precisely the "art" of deep learning model development. In this study, it was aimed to have a slight vision of its scope, but the enormous range of possibilities could not be handled due to time limitations. Hence, for future works continuing this study, these NN could be applied as well in FD001 and FD003, as well as trying different NN architectures apart from MLP.

As a last comment, and from a vision of someone who works in aerospace sector, it is for sure that the potential of these predictive tools has just began to be something more than things such as this project (academia simulations), but to be converted into an industrialized solution that each year more actors are involved an aiming to improve this technology to manage a non-stop progressing in flight safety and airspace companies' wealth.

8. Bibliography

[1] "AirsportsworldAero," 2019. [Online]. Available: https://www.airportsworld.aero/importance-of-aircraft-maintenance/#:~:text=Extending%20the%20life%20of%20the,through%20an%20aircraft%20maintenance%20schedule..

[2] S. Ackert, "Engine Maintenance Concepts for Financiers," Aircraft Monitor, 2011.

[3] I. Martínez, "Aircraft Envitronmental Control," in *Applied Propulsion Thermodynamics*, ETSIAE, 1995.

[4] C. a. D. o. a. m. The A, "QANTAS Newsroom," QANTAS airline, 18 July 2016. [Online]. Available: https://www.qantasnewsroom.com.au/roo-tales/the-a-c-and-d-of-aircraft-maintenance/.

[5] K. G. D. S. a. N. E. A. Saxena, "Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation, in the Proceedings of the Ist International Conference on Prognostics and Health Management (PHM08)," 2008.

[6] P. Koversis, "Machine Learning for Predictive Maintenance in Aviation," *Université PARIS-SACLAY,* 2019.

[7] S. -. parameters, "sklearn.model_selection.RandomizedSearchCV," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html.

[8] J. Avila, "Support Vector Regressor," JacobSoft - Consultoria de capacitacion, [Online]. Available: https://www.jacobsoft.com.mx/es_mx/support-vector-regression/.

[9] A. Smith, "Kaggle questions and answers," 2017. [Online]. Available: https://www.kaggle.com/questions-and-answers/33807.

[10] J. Amat, "PCA con Python," Cienciadedatos.net, 2020. [Online]. Available: https://www.cienciadedatos.net/documentos/py19-pca-python.html.

[11] A. Sharma, "Principal Component Analysis (PCA) in Python," Datacamp, [Online]. Available: https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python.

[12] Anonymous, "Inclusion of lagged dependent variable in regression," StackExchange, 2021. [Online]. Available: https://stats.stackexchange.com/questions/52458/inclusion-of-lagged-dependent-variable-in-regression.

[13] J. Brownlee, "How to Use Polynomial Feature Transforms for Machine Learning," Machine Learning Mastery, 29 May 2020. [Online]. Available: https://machinelearningmastery.com/polynomial-features-transforms-for-machine-learning/.

[14] Scikit-learn.org, " Neural network models (supervised)," [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[15] A. Kim, "How to Compare Two Distributions in Practice," Towards Data Science, 25 11 2019. [Online]. Available: https://towardsdatascience.com/how-to-compare-two-distributions-in-practice-8c676904a285.

[16] J. Browniee, "How to Get Reproducible Results with Keras," Maching Learning Mastery, 14 06 2017. [Online]. Available: https://machinelearningmastery.com/reproducible-results-neural-networks-keras/.

[17] D. Effrosynidis, "Normalization and Standardization in 2 Minutes," Towards Data Science, 13 06 2020. [Online]. Available: https://towardsdatascience.com/normalization-and-standardization-in-2-minutes-e0609a01e76.

[18] J. McHugh, "Smoothing Time Series in Python: A Walkthrough with Covid-19 Data," Medium, 18 08 2020. [Online]. Available: https://medium.com/@teamastertoast/smoothing-time-series-in-python-a-walkthrough-with-covid-19-data-b4ccfb980a61.

[19] J. Browniee, "How to Configure the Number of Layers and Nodes in a Neural Network," Machine Learning Mastery, 27 07 2018. [Online]. Available: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/.