Task 1: Understanding the data and problem

This is regression problem

Task 2: Data Visualization

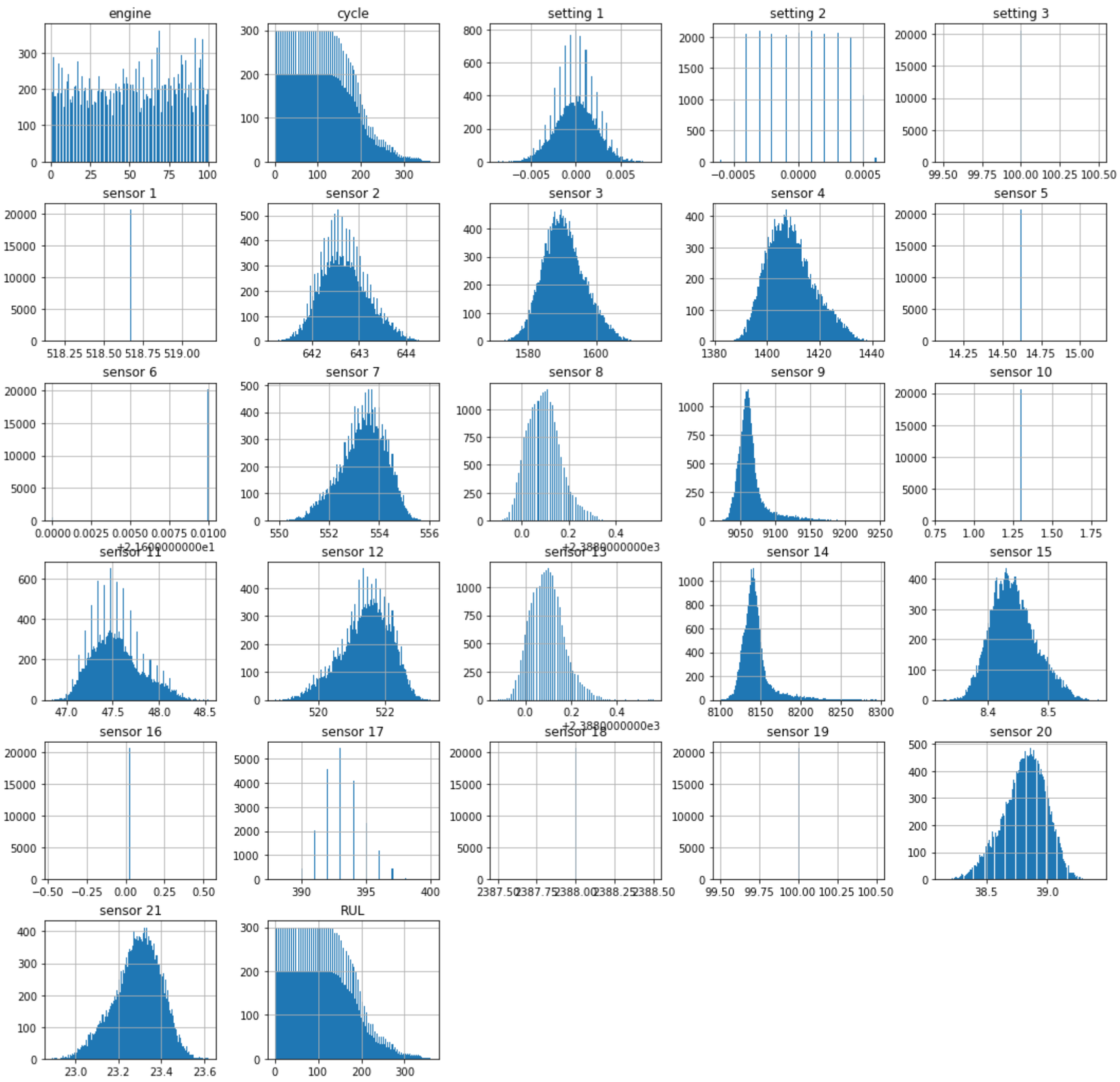
In [2]:
```
import pandas as pd
raw_train  = pd.read_csv('train_FD001.csv')
raw_train
```

Out[2]:

	engine	cycle	setting 1	setting 2	setting 3	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	...	sensor 13	sensor 14	sensor 15	sensor 16	sensor 17	sensor 18	sensor 19	sensor 20	sensor 21	RUL
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190	191.0
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236	190.0
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442	189.0
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739	188.0
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044	187.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735	4.0
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594	3.0
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333	2.0
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640	1.0
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522	0.0

20631 rows × 27 columns

In [4]:
```
#Histogram plots for each columns
import matplotlib.pyplot as plt
%matplotlib inline
#To calculate no of bins: https://datatofish.com/plot-histogram-python/
raw_train.hist(bins=144, figsize=(18,18))
plt.show()
```

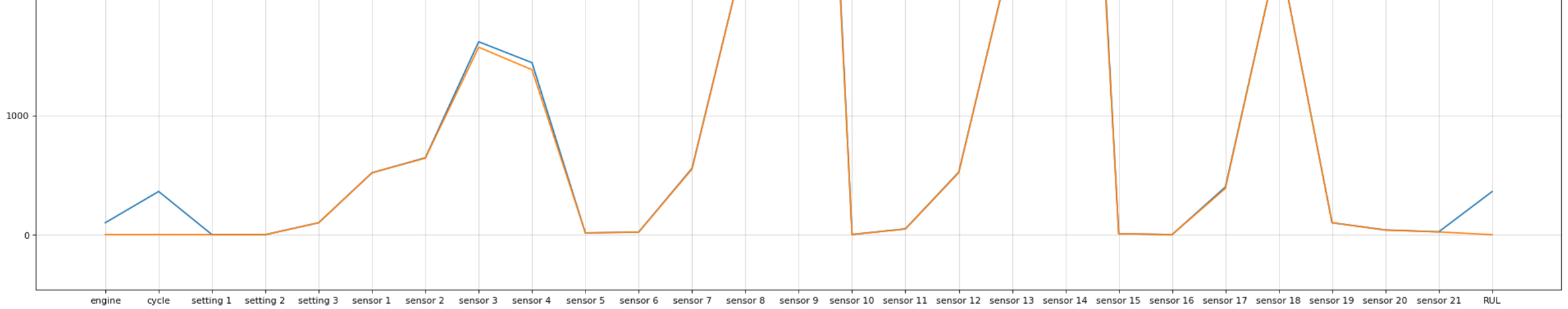


In [98]:
```
#Comparing the distribution of minimum and maximum values of each columns
import numpy as np

fig = plt.figure(figsize=(30, 30), dpi=80)
```

```
ax = fig.add_subplot(1, 1, 1)
maxValues = raw_train.max()
minValues = raw_train.min()
plt.plot(maxValues)
plt.plot(minValues)
major_ticks = np.arange(0, 13000, 1000) # Major ticks every 20, minor ticks every 5
ax.set_yticks(major_ticks)
ax.grid(which='both')# And a corresponding grid
ax.grid(which='major', alpha=0.5)
plt.legend(['Max','Min'])
plt.show()
```


Max
Min
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000



In [6]:
```
#Correlation coefficient with target 'RUL'
corr_matrix = raw_train.corr()
corr_matrix["RUL"].sort_values(ascending=False)
```

Out[6]:
```
RUL          1.000000
sensor 12    0.671983
sensor 7     0.657223
sensor 21    0.635662
sensor 20    0.629428
engine       0.078753
setting 2   -0.001948
setting 1   -0.003198
sensor 6    -0.128348
sensor 14   -0.306769
sensor 9    -0.390102
sensor 13   -0.562569
sensor 8    -0.563968
sensor 3    -0.584520
sensor 17   -0.606154
sensor 2    -0.606484
sensor 15   -0.642667
sensor 4    -0.678948
sensor 11   -0.696228
cycle       -0.736241
setting 3         NaN
sensor 1          NaN
sensor 5          NaN
sensor 10         NaN
sensor 16         NaN
sensor 18         NaN
sensor 19         NaN
Name: RUL, dtype: float64
```

In [7]:
```
raw_train.describe()
```

Out[7]:

	engine	cycle	setting 1	setting 2	setting 3	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	...	sensor 13	sensor 14	sensor 15	sensor 16	sensor 17	sensor 18	sensor 19	
count	20631.000000	20631.000000	20631.000000	20631.000000	20631.0	2.063100e+04	20631.000000	20631.000000	20631.000000	2.063100e+04	...	20631.000000	20631.000000	20631.000000	2.063100e+04	20631.000000	20631.0	20631.0	206
mean	51.506568	108.807862	-0.000009	0.000002	100.0	5.186700e+02	642.680934	1590.523119	1408.933782	1.462000e+01	...	2388.096152	8143.752722	8.442146	3.000000e-02	393.210654	2388.0	100.0	
std	29.227633	68.880990	0.002187	0.000293	0.0	6.537152e-11	0.500053	6.131150	9.000605	3.394700e-12	...	0.071919	19.076176	0.037505	1.556432e-14	1.548763	0.0	0.0	
min	1.000000	1.000000	-0.008700	-0.000600	100.0	5.186700e+02	641.210000	1571.040000	1382.250000	1.462000e+01	...	2387.880000	8099.940000	8.324900	3.000000e-02	388.000000	2388.0	100.0	
25%	26.000000	52.000000	-0.001500	-0.000200	100.0	5.186700e+02	642.325000	1586.260000	1402.360000	1.462000e+01	...	2388.040000	8133.245000	8.414900	3.000000e-02	392.000000	2388.0	100.0	
50%	52.000000	104.000000	0.000000	0.000000	100.0	5.186700e+02	642.640000	1590.100000	1408.040000	1.462000e+01	...	2388.090000	8140.540000	8.438900	3.000000e-02	393.000000	2388.0	100.0	
75%	77.000000	156.000000	0.001500	0.000300	100.0	5.186700e+02	643.000000	1594.380000	1414.555000	1.462000e+01	...	2388.140000	8148.310000	8.465600	3.000000e-02	394.000000	2388.0	100.0	
max	100.000000	362.000000	0.008700	0.000600	100.0	5.186700e+02	644.530000	1616.910000	1441.490000	1.462000e+01	...	2388.560000	8293.720000	8.584800	3.000000e-02	400.000000	2388.0	100.0	

8 rows × 27 columns

Task 3: Preprocessing and feature selection

In [8]:
```
#Removing constant features and highly uncorrelated features
raw_train_dropped_1 = (raw_train.drop(columns=['setting 3', 'sensor 1' , 'sensor 5' , 'sensor 10' , 'sensor 16' , 'sensor 18' , 'sensor 19' , 'setting 2' , 'setting 1' , 'engine'])).copy()
raw_train_dropped_1
```

Out[8]:

	cycle	sensor 2	sensor 3	sensor 4	sensor 6	sensor 7	sensor 8	sensor 9	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL
0	1	641.82	1589.70	1400.60	21.61	554.36	2388.06	9046.19	47.47	521.66	2388.02	8138.62	8.4195	392	39.06	23.4190	191.0
1	2	642.15	1591.82	1403.14	21.61	553.75	2388.04	9044.07	47.49	522.28	2388.07	8131.49	8.4318	392	39.00	23.4236	190.0
2	3	642.35	1587.99	1404.20	21.61	554.26	2388.08	9052.94	47.27	522.42	2388.03	8133.23	8.4178	390	38.95	23.3442	189.0
3	4	642.35	1582.79	1401.87	21.61	554.45	2388.11	9049.48	47.13	522.86	2388.08	8133.83	8.3682	392	38.88	23.3739	188.0
4	5	642.37	1582.85	1406.22	21.61	554.00	2388.06	9055.15	47.28	522.19	2388.04	8133.80	8.4294	393	38.90	23.4044	187.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	196	643.49	1597.98	1428.63	21.61	551.43	2388.19	9065.52	48.07	519.49	2388.26	8137.60	8.4956	397	38.49	22.9735	4.0
20627	197	643.54	1604.50	1433.58	21.61	550.86	2388.23	9065.11	48.04	519.68	2388.22	8136.50	8.5139	395	38.30	23.1594	3.0
20628	198	643.42	1602.46	1428.18	21.61	550.94	2388.24	9065.90	48.09	520.01	2388.24	8141.05	8.5646	398	38.44	22.9333	2.0
20629	199	643.23	1605.26	1426.53	21.61	550.68	2388.25	9073.72	48.39	519.67	2388.23	8139.29	8.5389	395	38.29	23.0640	1.0
20630	200	643.85	1600.38	1432.14	21.61	550.79	2388.26	9061.48	48.20	519.30	2388.26	8137.33	8.5036	396	38.37	23.0522	0.0

20631 rows × 17 columns

In [9]:
```
#Analysing the correlation
corr_matrix_2 = raw_train_dropped_1.corr()
corr_matrix_2["RUL"].sort_values(ascending=False)
```

Out[9]:
```
RUL          1.000000
sensor 12    0.671983
sensor 7     0.657223
sensor 21    0.635662
sensor 20    0.629428
sensor 6    -0.128348
sensor 14   -0.306769
sensor 9    -0.390102
sensor 13   -0.562569
sensor 8    -0.563968
sensor 3    -0.584520
sensor 17   -0.606154
sensor 2    -0.606484
sensor 15   -0.642667
sensor 4    -0.678948
sensor 11   -0.696228
cycle       -0.736241
Name: RUL, dtype: float64
```

In [10]:
```
#Colouring correlated/non-correlated cells
def correlated_cells(cell_value):

    high_plus = 'background-color: green;'
    high_minus = 'background-color: red;'
    low_plus_minus = 'background-color: yellow;'
    default = ''

    if type(cell_value) in [float, int]:
        if cell_value >= 0.8:
            return high_plus
        elif cell_value <= -0.8:
            return high_minus
        elif cell_value <= 0.2 and cell_value >= -0.2:
            return low_plus_minus
        else:
            return default

corr_matrix_2.style.applymap(correlated_cells)
```

Out[10]:

	cycle	sensor 2	sensor 3	sensor 4	sensor 6	sensor 7	sensor 8	sensor 9	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL
cycle	1.000000	0.549898	0.543947	0.624577	0.105980	-0.595914	0.475977	0.443999	0.634385	-0.611354	0.477523	0.370324	0.588676	0.566995	-0.583597	-0.585923	-0.736241
sensor 2	0.549898	1.000000	0.602610	0.714949	0.132242	-0.702136	0.662325	0.273764	0.740020	-0.724578	0.661792	0.179752	0.675975	0.629886	-0.661841	-0.668050	-0.606484
sensor 3	0.543947	0.602610	1.000000	0.678413	0.116039	-0.664595	0.602481	0.322964	0.695900	-0.680307	0.600963	0.237137	0.639921	0.600017	-0.625941	-0.633901	-0.584520
sensor 4	0.624577	0.714949	0.678413	1.000000	0.150480	-0.793130	0.746852	0.297429	0.830136	-0.815591	0.745158	0.190748	0.758459	0.703499	-0.748067	-0.745193	-0.678948
sensor 6	0.105980	0.132242	0.116039	0.150480	1.000000	-0.155720	0.152161	0.019347	0.160014	-0.155884	0.158276	-0.002112	0.149042	0.130810	-0.141419	-0.137419	-0.128348
sensor 7	-0.595914	-0.702136	-0.664595	-0.793130	-0.155720	1.000000	-0.767132	-0.217835	-0.822805	0.812713	-0.764611	-0.110053	-0.747051	-0.692893	0.736163	0.737447	0.657223
sensor 8	0.475977	0.662325	0.602481	0.746852	0.152161	-0.767132	1.000000	-0.032091	0.782213	-0.786540	0.826084	-0.144787	0.700949	0.627785	-0.687030	-0.688840	-0.563968
sensor 9	0.443999	0.273764	0.322964	0.297429	0.019347	-0.217835	-0.032091	1.000000	0.274591	-0.210238	-0.034763	0.963157	0.293753	0.337110	-0.285280	-0.292795	-0.390102
sensor 11	0.634385	0.740020	0.695900	0.830136	0.160014	-0.822805	0.782213	0.274591	1.000000	-0.846884	0.780761	0.163408	0.780913	0.722296	-0.771510	-0.772554	-0.696228
sensor 12	-0.611354	-0.724578	-0.680307	-0.815591	-0.155884	0.812713	-0.786540	-0.210238	-0.846884	1.000000	-0.788441	-0.098141	-0.766052	-0.703485	0.751943	0.756263	0.671983
sensor 13	0.477523	0.661792	0.600963	0.745158	0.158276	-0.764611	0.826084	-0.034763	0.780761	-0.788441	1.000000	-0.147036	0.697662	0.627410	-0.686172	-0.688948	-0.562569
sensor 14	0.370324	0.179752	0.237137	0.190748	-0.002112	-0.110053	-0.144787	0.963157	0.163408	-0.098141	-0.147036	1.000000	0.193761	0.246621	-0.187364	-0.192415	-0.306769
sensor 15	0.588676	0.675975	0.639921	0.758459	0.149042	-0.747051	0.700949	0.293753	0.780913	-0.766052	0.697662	0.193761	1.000000	0.669205	-0.707671	-0.704043	-0.642667
sensor 17	0.566995	0.629886	0.600017	0.703499	0.130810	-0.692893	0.627785	0.337110	0.722296	-0.703485	0.627410	0.246621	0.669205	1.000000	-0.651021	-0.658710	-0.606154
sensor 20	-0.583597	-0.661841	-0.625941	-0.748067	-0.141419	0.736163	-0.687030	-0.285280	-0.771510	0.751943	-0.686172	-0.187364	-0.707671	-0.651021	1.000000	0.693480	0.629428
sensor 21	-0.585923	-0.668050	-0.633901	-0.745193	-0.137419	0.737447	-0.688840	-0.292795	-0.772554	0.756263	-0.688948	-0.192415	-0.704043	-0.658710	0.693480	1.000000	0.635662
RUL	-0.736241	-0.606484	-0.584520	-0.678948	-0.128348	0.657223	-0.563968	-0.390102	-0.696228	0.671983	-0.562569	-0.306769	-0.642667	-0.606154	0.629428	0.635662	1.000000

In [11]:
```
#Strong relationships
#11 and 4 positive & take 11
#12 and 7 positive & take 12
#13 and 8 positive & take 13
#14 and 9 positive & take 14

#12 and 11 negative
#12 and 4 negative
#11 and 7 negative

#Removing other uncorrelated features
train_dropped = (raw_train_dropped_1.drop(columns=['sensor 4', 'sensor 6', 'sensor 7' , 'sensor 8' , 'sensor 9'])).copy()
train_dropped
```

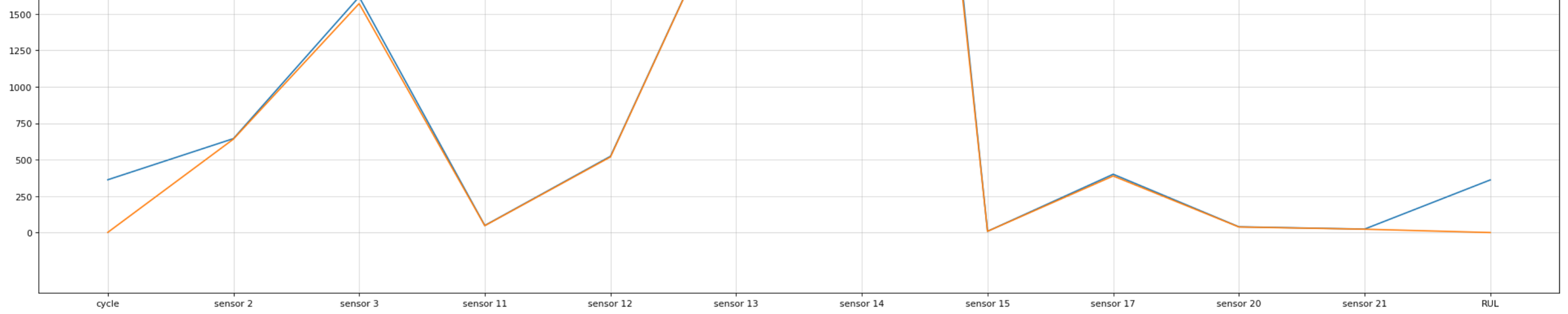
Out[11]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL
0	1	641.82	1589.70	47.47	521.66	2388.02	8138.62	8.4195	392	39.06	23.4190	191.0
1	2	642.15	1591.82	47.49	522.28	2388.07	8131.49	8.4318	392	39.00	23.4236	190.0
2	3	642.35	1587.99	47.27	522.42	2388.03	8133.23	8.4178	390	38.95	23.3442	189.0
3	4	642.35	1582.79	47.13	522.86	2388.08	8133.83	8.3682	392	38.88	23.3739	188.0
4	5	642.37	1582.85	47.28	522.19	2388.04	8133.80	8.4294	393	38.90	23.4044	187.0
...	...	...	...	...	...	...	...	...	...	...	...	...
20626	196	643.49	1597.98	48.07	519.49	2388.26	8137.60	8.4956	397	38.49	22.9735	4.0
20627	197	643.54	1604.50	48.04	519.68	2388.22	8136.50	8.5139	395	38.30	23.1594	3.0
20628	198	643.42	1602.46	48.09	520.01	2388.24	8141.05	8.5646	398	38.44	22.9333	2.0
20629	199	643.23	1605.26	48.39	519.67	2388.23	8139.29	8.5389	395	38.29	23.0640	1.0
20630	200	643.85	1600.38	48.20	519.30	2388.26	8137.33	8.5036	396	38.37	23.0522	0.0

20631 rows × 12 columns

In [12]:
```
fig = plt.figure(figsize=(30, 30), dpi=80)
ax = fig.add_subplot(1, 1, 1)
maxValues = train_dropped.max()
minValues = train_dropped.min()
plt.plot(maxValues)
plt.plot(minValues)
major_ticks = np.arange(0, 10000, 250) # Major ticks every 20, minor ticks every 5
ax.set_yticks(major_ticks)
ax.grid(which='both')# And a corresponding grid
ax.grid(which='major', alpha=0.5)
plt.show()
```


9750
9500
9250
9000
8750
8500
8250
8000
7750
7500
7250
7000
6750
6500
6250
6000
5750
5500
5250
5000
4750
4500
4250
4000
3750
3500
3250
3000
2750
2500
2250
2000
1750



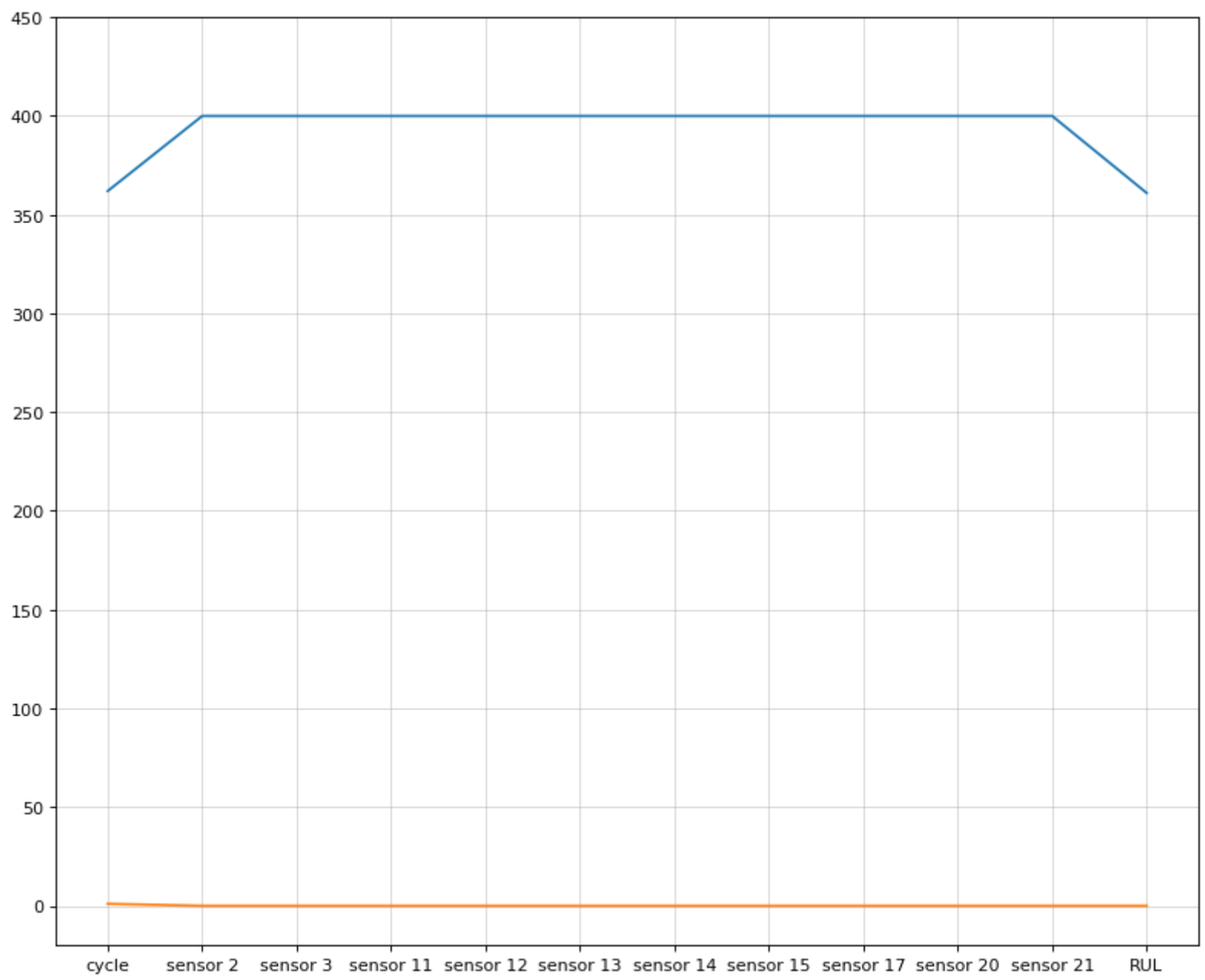
In [13]:
```
#Cycle features is not follows the normal distribution curve. Sensors 12,14,20 has skewness in their curve.
#Sensor values are subjected to the normalization to the range [0, 400]
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
train_norm = train_dropped.copy()
train_norm[["sensor 2", "sensor 3", "sensor 11", "sensor 13", "sensor 14", "sensor 15", "sensor 20", "sensor 21","sensor 12", "sensor 17"]] = min_max_scaler.fit_transform(train_norm[["sensor 2", "sensor 3
train_norm_to_range = train_norm.copy()
train_norm_to_range[["sensor 2", "sensor 3", "sensor 11", "sensor 13", "sensor 14", "sensor 15", "sensor 20", "sensor 21", "sensor 12", "sensor 17"]] = (train_norm_to_range[["sensor 2", "sensor 3", "senso
train_norm_to_range
```

Out[13]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL
0	1	73.493976	162.720733	147.619048	253.304904	82.352941	79.843121	145.594459	133.333333	285.271318	289.864678	191.0
1	2	113.253012	181.207761	152.380952	306.183369	111.764706	65.125400	164.524817	133.333333	266.666667	292.405413	190.0
2	3	137.349398	147.809026	100.000000	318.123667	88.235294	68.717102	142.978068	66.666667	251.162791	248.550124	189.0
3	4	137.349398	102.463484	66.666667	355.650320	117.647059	69.955620	66.641016	133.333333	229.457364	264.954432	188.0
4	5	139.759036	102.986702	102.380952	298.507463	94.117647	69.893694	160.831089	166.666667	235.658915	281.800608	187.0
...	...	...	...	...	...	...	...	...	...	...	...	...
20626	196	274.698795	234.924787	290.476190	68.230277	223.529412	77.737641	262.716429	300.000000	108.527132	43.800055	4.0
20627	197	280.722892	291.781121	283.333333	84.434968	200.000000	75.467024	290.881108	233.333333	49.612403	146.478873	3.0
20628	198	266.265060	273.991716	295.238095	112.579957	211.764706	84.859119	368.911120	333.333333	93.023256	21.596244	2.0
20629	199	243.373494	298.408546	366.666667	83.582090	205.882353	81.226133	329.357445	233.333333	46.511628	93.786247	1.0
20630	200	318.072289	255.853499	321.428571	52.025586	223.529412	77.180308	275.028857	266.666667	71.317829	87.268710	0.0

20631 rows × 12 columns

In [14]:
```
fig = plt.figure(figsize=(12, 10), dpi=80)
ax = fig.add_subplot(1, 1, 1)
maxValues = train_norm_to_range.max()
minValues = train_norm_to_range.min()
plt.plot(maxValues)
plt.plot(minValues)
major_ticks = np.arange(0, 500, 50) # Major ticks every 20, minor ticks every 5
ax.set_yticks(major_ticks)
ax.grid(which='both')# And a corresponding grid
ax.grid(which='major', alpha=0.5)
plt.show()
```



Task 4: Regression models

In [100…
```
#Linear regression in train data set
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()

#Assigning X and Y values in train data set
X = train_norm_to_range[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12',
                         'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']]
Y = train_norm_to_range['RUL']
lin_reg.fit(X, Y)
Y_lin_reg_predict_train = lin_reg.predict(X)
lin_reg.intercept_, lin_reg.coef_
```

Out[100]:
```
(188.02601493040865,
 array([-0.40592463, -0.05490579, -0.04178638, -0.13312342,  0.09455023,
        -0.07169688, -0.14353226, -0.06853918, -0.04287067,  0.04737151,
         0.05814561]))
```

In [16]:
```
#Polynomial regression in train data set
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, Y)
Y_poly_reg_predict_train = lin_reg.predict(X_poly)
```

In [17]:
```
#Regularized Linear Models in train data set

#Ridge Regression
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=5, solver="cholesky")
ridge_reg.fit(X, Y)
Y_ridge_reg_predict_train = ridge_reg.predict(X)

#Lasso Regression
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=1)
lasso_reg.fit(X, Y)
Y_lasso_reg_predict_train = lasso_reg.predict(X)

#Elastic Net
from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(alpha=1, l1_ratio=0.5)
elastic_net.fit(X, Y)
Y_elastic_net_reg_predict_train = elastic_net.predict(X)
```

In [18]:
```
#SVM Linear in train data set
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon=0.1,max_iter=10000)
svm_reg.fit(X, Y)
Y_SVM_lin_reg_predict_train = svm_reg.predict(X)
```

```
C:\Users\Niroshan\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

In [19]:
```
#SVM Poly in train data set
from sklearn.svm import SVR
svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, Y)
Y_SVM_poly_reg_predict_train = svm_poly_reg.predict(X)
```

In [20]:
```
#Predicted RUL for training dataset
train_rul = train_norm_to_range.copy()
train_rul.insert(12, "Linear Regression", Y_lin_reg_predict_train, True)
train_rul.insert(13, "Polynomial deg 3 Regression", Y_poly_reg_predict_train, True)

train_rul.insert(14, "Ridge Regression", Y_ridge_reg_predict_train, True)
train_rul.insert(15, "Lasso Regression", Y_lasso_reg_predict_train, True)
train_rul.insert(16, "Elastic Net Regression", Y_elastic_net_reg_predict_train, True)
train_rul.insert(17, "SVM Linear Regression", Y_SVM_lin_reg_predict_train, True)
train_rul.insert(18, "SVM Polynomial deg 2 Regression", Y_SVM_poly_reg_predict_train, True)
train_rul
```

Out[20]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL	Linear Regression	Polynomial deg 3 Regression	Ridge Regression	Lasso Regression	Elastic Net Regression	SVM Linear Regression	SVM Polynomial deg 2 Regression
0	1	73.493976	162.720733	147.619048	253.304904	82.352941	79.843121	145.594459	133.333333	285.271318	289.864678	191.0	178.392383	212.387939	178.392381	178.335467	178.355216	161.888959	196.159272
1	2	113.253012	181.207761	152.380952	306.183369	111.764706	65.125400	164.524817	133.333333	266.666667	292.405413	190.0	177.369368	220.947471	177.369366	177.347788	177.352369	167.923699	217.409920
2	3	137.349398	147.809026	100.000000	318.123667	88.235294	68.717102	142.978068	66.666667	251.162791	248.550124	189.0	187.359981	227.364486	187.359978	187.325970	187.338738	174.134764	202.244775
3	4	137.349398	102.463484	66.666667	355.650320	117.647059	69.955620	66.641016	133.333333	229.457364	264.954432	188.0	196.847637	222.813313	196.847636	196.893170	196.871998	182.199035	215.504548
4	5	139.759036	102.986702	102.380952	298.507463	94.117647	69.893694	160.831089	166.666667	235.658915	281.800608	187.0	181.214717	214.518965	181.214714	181.196272	181.201330	166.734633	201.270464
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	196	274.698795	234.924787	290.476190	68.230277	223.529412	77.737641	262.716429	300.000000	108.527132	43.800055	4.0	0.983665	0.837619	0.983665	1.022580	1.002745	-41.600415	6.901777
20627	197	280.722892	291.781121	283.333333	84.434968	200.000000	75.467024	290.881108	233.333333	49.612403	146.478873	3.0	6.474197	7.806645	6.474196	6.481603	6.475167	-34.275189	8.602582
20628	198	266.265060	273.991716	295.238095	112.579957	211.764706	84.859119	368.911120	333.333333	93.023256	21.596244	2.0	-8.349921	17.656845	-8.349924	-8.315618	-8.334009	-41.253068	1.073293
20629	199	243.373494	298.408546	366.666667	83.582090	205.882353	81.226133	329.357445	233.333333	46.511628	93.786247	1.0	-10.834382	-9.466395	-10.834382	-10.854969	-10.849700	-49.630570	-6.444320
20630	200	318.072289	255.853499	321.428571	52.025586	223.529412	77.180308	275.028857	266.666667	71.317829	87.268710	0.0	-8.118684	-1.505314	-8.118684	-8.100376	-8.112174	-51.927021	-1.328717

20631 rows × 19 columns

In [21]:

```
from sklearn.metrics import mean_squared_error

train_rul_array = train_rul['RUL'].copy().to_numpy()
data = {'Model':
            ['Linear Regression'
            , 'Polynomial Regression degree 3'
            , 'Ridge Regression'
            , 'Lasso Regression'
            , 'Elastic Net'
            , 'SVM Linear Regression'
            , 'SVM Polynomial deg 2 Regression'
            ],
        'RMSE (Train)':
            [np.sqrt(mean_squared_error(train_rul_array,Y_lin_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_poly_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_ridge_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_lasso_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_elastic_net_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_SVM_lin_reg_predict_train))
            , np.sqrt(mean_squared_error(train_rul_array,Y_SVM_poly_reg_predict_train))
            ],
        'Corr with RUL (Train)': [np.corrcoef(train_rul_array,Y_lin_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_poly_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_ridge_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_lasso_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_elastic_net_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_SVM_lin_reg_predict_train)[0,1]
            , np.corrcoef(train_rul_array,Y_SVM_poly_reg_predict_train)[0,1]
            ]}
summary = pd.DataFrame(data)
summary
```

Out[21]:

	Model	RMSE (Train)	Corr with RUL (Train)
0	Linear Regression	40.562272	0.808214
1	Polynomial Regression degree 3	35.986458	0.852666
2	Ridge Regression	40.562272	0.808214
3	Lasso Regression	40.562297	0.808214
4	Elastic Net	40.562280	0.808214
5	SVM Linear Regression	47.972160	0.805876
6	SVM Polynomial deg 2 Regression	37.769510	0.841626

In [22]:
```
#Model evaluation with test data
raw_test  = pd.read_csv('test_FD001.csv')
raw_test
```

Out[22]:

	engine	cycle	setting 1	setting 2	setting 3	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	...	sensor 13	sensor 14	sensor 15	sensor 16	sensor 17	sensor 18	sensor 19	sensor 20	sensor 21	RUL
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735	142.0
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916	141.0
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166	140.0
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	2388.05	8132.90	8.3917	0.03	391	2388	100.0	39.00	23.3737	139.0
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	2388.03	8129.54	8.4031	0.03	390	2388	100.0	38.99	23.4130	138.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13091	100	194	0.0049	0.0000	100.0	518.67	643.24	1599.45	1415.79	14.62	...	2388.00	8213.28	8.4715	0.03	394	2388	100.0	38.65	23.1974	24.0
13092	100	195	-0.0011	-0.0001	100.0	518.67	643.22	1595.69	1422.05	14.62	...	2388.09	8210.85	8.4512	0.03	395	2388	100.0	38.57	23.2771	23.0
13093	100	196	-0.0006	-0.0003	100.0	518.67	643.44	1593.15	1406.82	14.62	...	2388.04	8217.24	8.4569	0.03	395	2388	100.0	38.62	23.2051	22.0
13094	100	197	-0.0038	0.0001	100.0	518.67	643.26	1594.99	1419.36	14.62	...	2388.08	8220.48	8.4711	0.03	395	2388	100.0	38.66	23.2699	21.0
13095	100	198	0.0013	0.0003	100.0	518.67	642.95	1601.62	1424.99	14.62	...	2388.05	8214.64	8.4903	0.03	396	2388	100.0	38.70	23.1855	20.0

13096 rows × 27 columns

In [23]:
```
test_dropped = (raw_test.drop(columns=['setting 3', 'sensor 1' , 'sensor 5' , 'sensor 10' , 'sensor 16' , 'sensor 18' , 'sensor 19' , 'setting 2' , 'setting 1' , 'engine', 'sensor 4', 'sensor 6', 'sensor
test_dropped
```

Out[23]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21	RUL
0	1	643.02	1585.29	47.20	521.72	2388.03	8125.55	8.4052	392	38.86	23.3735	142.0
1	2	641.71	1588.45	47.50	522.16	2388.06	8139.62	8.3803	393	39.02	23.3916	141.0
2	3	642.46	1586.94	47.50	521.97	2388.03	8130.10	8.4441	393	39.08	23.4166	140.0
3	4	642.44	1584.12	47.28	521.38	2388.05	8132.90	8.3917	391	39.00	23.3737	139.0
4	5	642.51	1587.19	47.31	522.15	2388.03	8129.54	8.4031	390	38.99	23.4130	138.0
...	...	...	...	...	...	...	...	...	...	...	...	...
13091	194	643.24	1599.45	47.69	520.69	2388.00	8213.28	8.4715	394	38.65	23.1974	24.0
13092	195	643.22	1595.69	47.60	521.05	2388.09	8210.85	8.4512	395	38.57	23.2771	23.0
13093	196	643.44	1593.15	47.57	521.18	2388.04	8217.24	8.4569	395	38.62	23.2051	22.0
13094	197	643.26	1594.99	47.61	521.33	2388.08	8220.48	8.4711	395	38.66	23.2699	21.0
13095	198	642.95	1601.62	47.80	521.07	2388.05	8214.64	8.4903	396	38.70	23.1855	20.0

13096 rows × 12 columns

In [24]:
```
#Scaling test data according to the train data set parameters for [0,400]
maxValues = train_dropped.max()#Max values of train data set
minValues = train_dropped.min()#Min values of train data set
X_test_scaled = test_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
X_test_scaled[["sensor 2"]] = 400* (X_test_scaled[["sensor 2"]] - minValues["sensor 2"]) / (maxValues["sensor 2"] - minValues["sensor 2"])
X_test_scaled[["sensor 3"]] = 400* (X_test_scaled[["sensor 3"]] - minValues["sensor 3"]) / (maxValues["sensor 3"] - minValues["sensor 3"])
X_test_scaled[["sensor 11"]] = 400* (X_test_scaled[["sensor 11"]] - minValues["sensor 11"]) / (maxValues["sensor 11"] - minValues["sensor 11"])
X_test_scaled[["sensor 12"]] = 400* (X_test_scaled[["sensor 12"]] - minValues["sensor 12"]) / (maxValues["sensor 12"] - minValues["sensor 12"])
X_test_scaled[["sensor 13"]] = 400* (X_test_scaled[["sensor 13"]] - minValues["sensor 13"]) / (maxValues["sensor 13"] - minValues["sensor 13"])
X_test_scaled[["sensor 14"]] = 400* (X_test_scaled[["sensor 14"]] - minValues["sensor 14"]) / (maxValues["sensor 14"] - minValues["sensor 14"])
X_test_scaled[["sensor 15"]] = 400* (X_test_scaled[["sensor 15"]] - minValues["sensor 15"]) / (maxValues["sensor 15"] - minValues["sensor 15"])
X_test_scaled[["sensor 17"]] = 400* (X_test_scaled[["sensor 17"]] - minValues["sensor 17"]) / (maxValues["sensor 17"] - minValues["sensor 17"])
X_test_scaled[["sensor 20"]] = 400* (X_test_scaled[["sensor 20"]] - minValues["sensor 20"]) / (maxValues["sensor 20"] - minValues["sensor 20"])
X_test_scaled[["sensor 21"]] = 400* (X_test_scaled[["sensor 21"]] - minValues["sensor 21"]) / (maxValues["sensor 21"] - minValues["sensor 21"])
X_test_scaled
```

Out[24]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21
0	1	218.072289	124.264225	83.333333	258.422175	88.235294	52.864073	123.585995	133.333333	223.255814	264.733499
1	2	60.240964	151.820362	154.761905	295.948827	105.882353	81.907318	85.263563	166.666667	272.868217	274.730737
2	3	150.602410	138.652714	154.761905	279.744136	88.235294	62.256167	183.455175	166.666667	291.472868	288.539078
3	4	148.192771	114.061478	102.380952	229.424307	100.000000	68.035917	102.808773	100.000000	266.666667	264.843966
4	5	156.626506	140.832788	109.523810	295.095949	88.235294	61.100217	120.353982	66.666667	263.565891	286.550677
...	...	...	...	...	...	...	...	...	...	...	...
13091	194	244.578313	247.743623	200.000000	170.575693	70.588235	233.956033	225.625240	200.000000	158.139535	167.467550
13092	195	242.168675	214.955308	178.571429	201.279318	123.529412	228.940035	194.382455	233.333333	133.333333	211.488539
13093	196	268.674699	192.805755	171.428571	212.366738	94.117647	242.130251	203.155060	233.333333	148.837209	171.720519
13094	197	246.987952	208.851101	180.952381	225.159915	117.647059	248.818247	225.009619	233.333333	161.240310	207.511737
13095	198	209.638554	266.666667	226.190476	202.985075	100.000000	236.763340	254.559446	266.666667	173.643411	160.894780

13096 rows × 11 columns

In [25]:
```
#Test data with linear regression
lin_reg.fit(X, Y)
```

```
Y_lin_reg_predict_test = lin_reg.predict(X_test_scaled)
Y_lin_reg_predict_test
```

Out[25]:
```
array([181.66295106, 181.50594067, 174.19678597, ...,  34.521257  ,
        33.10117279,  21.62930354])
```

In [26]:
```
#Test data with polynomial regression
X_poly = poly_features.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, Y)
X_test_scaled_poly = poly_features.fit_transform(X_test_scaled)
Y_poly_reg_predict_test = lin_reg.predict(X_test_scaled_poly)
Y_poly_reg_predict_test
```

Out[26]:
```
array([218.74157232, 223.11371513, 214.27077592, ...,  13.67781051,
         9.75257392,   8.51440295])
```

In [27]:
```
#Test with regularized Linear Models

#Ridge Regression
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=5, solver="cholesky")
ridge_reg.fit(X, Y)
Y_ridge_reg_predict_test = ridge_reg.predict(X_test_scaled)

#Lasso Regression
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=1)
lasso_reg.fit(X, Y)
Y_lasso_reg_predict_test = lasso_reg.predict(X_test_scaled)

#Elastic Net
from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(alpha=1, l1_ratio=0.5)
elastic_net.fit(X, Y)
Y_elastic_net_reg_predict_test = elastic_net.predict(X_test_scaled)
```

In [28]:
```
#Test with SVM Linear
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon=0.1,max_iter=10000)
svm_reg.fit(X, Y)
Y_SVM_lin_reg_predict_test = svm_reg.predict(X_test_scaled)
```

```
C:\Users\Niroshan\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

In [29]:
```
#Test with SVM Poly
from sklearn.svm import SVR
svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, Y)
Y_SVM_poly_reg_predict_test = svm_poly_reg.predict(X_test_scaled)
```

In [31]:
```
test_rul_dataframe = (test_dropped.drop(columns=['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21'])).copy()
test_rul_dataframe.rename(columns = {'RUL':'RUL - Test'}, inplace = True)
test_rul_dataframe ['Linear Regression - Test'] = Y_lin_reg_predict_test.tolist()
test_rul_dataframe ['Polynomial Regression deg 3 - Test'] = Y_poly_reg_predict_test.tolist()
test_rul_dataframe ['Ridge Regression - Test'] = Y_ridge_reg_predict_test.tolist()
test_rul_dataframe ['Lasso Regression - Test'] = Y_lasso_reg_predict_test.tolist()
test_rul_dataframe ['Elastic Net Regression - Test'] = Y_elastic_net_reg_predict_test.tolist()
test_rul_dataframe ['SVM linear Regression - Test'] = Y_SVM_lin_reg_predict_test.tolist()
test_rul_dataframe ['SVM Polynomial deg 2 Regression - Test'] = Y_SVM_poly_reg_predict_test.tolist()
test_rul_dataframe
```

Out[31]:

	RUL - Test	Linear Regression - Test	Polynomial Regression deg 3 - Test	Ridge Regression - Test	Lasso Regression - Test	Elastic Net Regression - Test	SVM linear Regression - Test	SVM Polynomial deg 2 Regression - Test
0	142.0	181.662951	218.741572	181.662947	181.613551	181.631055	190.005562	194.189444
1	141.0	181.505941	223.113715	181.505940	181.496459	181.495939	191.596097	203.294057
2	140.0	174.196786	214.270776	174.196783	174.127542	174.153267	197.510942	212.006895
3	139.0	181.325659	216.048290	181.325657	181.295834	181.305249	183.168450	184.769456
4	138.0	187.777124	224.702620	187.777122	187.726131	187.746099	193.262730	204.678742
...	...	...	...	...	...	...	...	...
13091	24.0	29.548162	3.658843	29.548163	29.579722	29.564569	61.819384	24.925131
13092	23.0	35.421417	15.942830	35.421420	35.553662	35.495070	74.670676	20.791689
13093	22.0	34.521257	13.677811	34.521259	34.622645	34.578465	70.726171	21.642975
13094	21.0	33.101173	9.752574	33.101176	33.246274	33.182468	77.990608	8.932829
13095	20.0	21.629304	8.514403	21.629305	21.707982	21.671390	68.124292	5.070769

13096 rows × 8 columns

In [32]:
```
#Total summary
total_summary = summary.copy()
test_rul_array = test_dropped['RUL'].copy().to_numpy()
total_summary.insert(2, "RMSE (Test)",
                                    [np.sqrt(mean_squared_error(test_rul_array,Y_lin_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_poly_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_ridge_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_lasso_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_elastic_net_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_SVM_lin_reg_predict_test))
                                    ,np.sqrt(mean_squared_error(test_rul_array,Y_SVM_poly_reg_predict_test))
                                    ], True)
total_summary.insert(4, "Corr with RUL (Test)",
                                     [np.corrcoef(test_rul_array,Y_lin_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_poly_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_ridge_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_lasso_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_elastic_net_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_SVM_lin_reg_predict_test)[0,1]
                                    ,np.corrcoef(test_rul_array,Y_SVM_poly_reg_predict_test)[0,1]
                                     ], True)
total_summary
```

Out[32]:

	Model	RMSE (Train)	RMSE (Test)	Corr with RUL (Train)	Corr with RUL (Test)
0	Linear Regression	40.562272	42.983388	0.808214	0.690076
1	Polynomial Regression degree 3	35.986458	40.800747	0.852666	0.725349
2	Ridge Regression	40.562272	42.983388	0.808214	0.690076
3	Lasso Regression	40.562297	42.985801	0.808214	0.690059
4	Elastic Net	40.562280	42.985193	0.808214	0.690059
5	SVM Linear Regression	47.972160	45.889857	0.805876	0.685516
6	SVM Polynomial deg 2 Regression	37.769510	42.363157	0.841626	0.718104

Task 5: Regression models with extended features

In [33]:
```
#Feature lag for train data
train_X_t0 = train_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
```

```
train_X_t1 = train_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
train_X_t2 = train_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
train_X_t1 =  train_X_t1.shift(1)
train_X_t1.columns = ['cycle(t-1)', 'sensor 2(t-1)', 'sensor 3(t-1)', 'sensor 11(t-1)', 'sensor 12(t-1)', 'sensor 13(t-1)', 'sensor 14(t-1)', 'sensor 15(t-1)', 'sensor 17(t-1)', 'sensor 20(t-1)', 'sensor
train_X_t2 =  train_X_t2.shift(2)
train_X_t2.columns = ['cycle(t-2)', 'sensor 2(t-2)', 'sensor 3(t-2)', 'sensor 11(t-2)', 'sensor 12(t-2)', 'sensor 13(t-2)', 'sensor 14(t-2)', 'sensor 15(t-2)', 'sensor 17(t-2)', 'sensor 20(t-2)', 'sensor
train_X_t0
```

Out[33]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	sensor 21
0	1	641.82	1589.70	47.47	521.66	2388.02	8138.62	8.4195	392	39.06	23.4190
1	2	642.15	1591.82	47.49	522.28	2388.07	8131.49	8.4318	392	39.00	23.4236
2	3	642.35	1587.99	47.27	522.42	2388.03	8133.23	8.4178	390	38.95	23.3442
3	4	642.35	1582.79	47.13	522.86	2388.08	8133.83	8.3682	392	38.88	23.3739
4	5	642.37	1582.85	47.28	522.19	2388.04	8133.80	8.4294	393	38.90	23.4044
...	...	...	...	...	...	...	...	...	...	...	...
20626	196	643.49	1597.98	48.07	519.49	2388.26	8137.60	8.4956	397	38.49	22.9735
20627	197	643.54	1604.50	48.04	519.68	2388.22	8136.50	8.5139	395	38.30	23.1594
20628	198	643.42	1602.46	48.09	520.01	2388.24	8141.05	8.5646	398	38.44	22.9333
20629	199	643.23	1605.26	48.39	519.67	2388.23	8139.29	8.5389	395	38.29	23.0640
20630	200	643.85	1600.38	48.20	519.30	2388.26	8137.33	8.5036	396	38.37	23.0522

20631 rows × 11 columns

In [34]:
```
#Combination features / target of training dataset
train_lag_features = pd.concat([train_X_t0, train_X_t1, train_X_t2], axis=1)
train_lag_target = train_dropped[['RUL']].copy()
#Top two rows are not important since there are NAN values
train_lag_features.drop([0,1], axis=0, inplace=True)
train_lag_target.drop([0,1], axis=0, inplace=True)#Similarly delete first 2 rows of target 'RUL'
```

In [35]:
```
#Feature lag for test data
test_X_t0 = test_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
test_X_t1 = test_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
test_X_t2 = test_dropped[['cycle', 'sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']].copy()
test_X_t1 =  test_X_t1.shift(1)
test_X_t1.columns = ['cycle(t-1)', 'sensor 2(t-1)', 'sensor 3(t-1)', 'sensor 11(t-1)', 'sensor 12(t-1)', 'sensor 13(t-1)', 'sensor 14(t-1)', 'sensor 15(t-1)', 'sensor 17(t-1)', 'sensor 20(t-1)', 'sensor 2
test_X_t2 =  test_X_t2.shift(2)
test_X_t2.columns = ['cycle(t-2)', 'sensor 2(t-2)', 'sensor 3(t-2)', 'sensor 11(t-2)', 'sensor 12(t-2)', 'sensor 13(t-2)', 'sensor 14(t-2)', 'sensor 15(t-2)', 'sensor 17(t-2)', 'sensor 20(t-2)', 'sensor 2
test_X_t2
```

Out[35]:

	cycle(t-2)	sensor 2(t-2)	sensor 3(t-2)	sensor 11(t-2)	sensor 12(t-2)	sensor 13(t-2)	sensor 14(t-2)	sensor 15(t-2)	sensor 17(t-2)	sensor 20(t-2)	sensor 21(t-2)
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1.0	643.02	1585.29	47.20	521.72	2388.03	8125.55	8.4052	392.0	38.86	23.3735
3	2.0	641.71	1588.45	47.50	522.16	2388.06	8139.62	8.3803	393.0	39.02	23.3916
4	3.0	642.46	1586.94	47.50	521.97	2388.03	8130.10	8.4441	393.0	39.08	23.4166
...	...	...	...	...	...	...	...	...	...	...	...
13091	192.0	643.05	1598.18	47.57	520.99	2388.07	8210.24	8.4171	395.0	38.77	23.2148
13092	193.0	643.10	1595.60	47.58	521.37	2388.05	8213.57	8.4429	395.0	38.63	23.2952
13093	194.0	643.24	1599.45	47.69	520.69	2388.00	8213.28	8.4715	394.0	38.65	23.1974
13094	195.0	643.22	1595.69	47.60	521.05	2388.09	8210.85	8.4512	395.0	38.57	23.2771
13095	196.0	643.44	1593.15	47.57	521.18	2388.04	8217.24	8.4569	395.0	38.62	23.2051

13096 rows × 11 columns

In [36]:
```
#Combination features / target of test dataset
test_lag_features = pd.concat([test_X_t0, test_X_t1, test_X_t2], axis=1)
test_lag_target = test_dropped[['RUL']].copy()
#Top two rows are not important since there are NAN values
test_lag_features.drop([0,1], axis=0, inplace=True)
test_lag_target.drop([0,1], axis=0, inplace=True)#Similarly delete first 2 rows of target 'RUL'
```

In [37]:
```
#Scaling training dataset by bringing into [0,400] range
scaled_train_lag_features = train_lag_features.copy()
scaled_train_lag_features[["sensor 2", "sensor 3", "sensor 11", "sensor 13", "sensor 14", "sensor 15", "sensor 20", "sensor 21","sensor 12", "sensor 17"]] = min_max_scaler.fit_transform(scaled_train_lag_f
scaled_train_lag_features[["sensor 2(t-1)", "sensor 3(t-1)", "sensor 11(t-1)", "sensor 13(t-1)", "sensor 14(t-1)", "sensor 15(t-1)", "sensor 20(t-1)", "sensor 21(t-1)","sensor 12(t-1)", "sensor 17(t-1)"]]
scaled_train_lag_features[["sensor 2(t-2)", "sensor 3(t-2)", "sensor 11(t-2)", "sensor 13(t-2)", "sensor 14(t-2)", "sensor 15(t-2)", "sensor 20(t-2)", "sensor 21(t-2)","sensor 12(t-2)", "sensor 17(t-2)"]]
scaled_train_lag_features
```

Out[37]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	...	sensor 2(t-2)	sensor 3(t-2)	sensor 11(t-2)	sensor 12(t-2)	sensor 13(t-2)	sensor 14(t-2)	sensor 15(t-2)	sensor 17(t-2)	sensor 20(t-2)	sens 21(t-
2	3	137.349398	147.809026	100.000000	318.123667	88.235294	68.717102	142.978068	66.666667	251.162791	...	73.493976	162.720733	147.619048	253.304904	82.352941	79.843121	145.594459	133.333333	285.271318	289.8646
3	4	137.349398	102.463484	66.666667	355.650320	117.647059	69.955620	66.641016	133.333333	229.457364	...	113.253012	181.207761	152.380952	306.183369	111.764706	65.125400	164.524817	133.333333	266.666667	292.4054
4	5	139.759036	102.986702	102.380952	298.507463	94.117647	69.893694	160.831089	166.666667	235.658915	...	137.349398	147.809026	100.000000	318.123667	88.235294	68.717102	142.978068	66.666667	251.162791	248.5501
5	6	107.228916	117.113582	73.809524	255.010661	88.235294	67.932707	132.204694	100.000000	260.465116	...	137.349398	102.463484	66.666667	355.650320	117.647059	69.955620	66.641016	133.333333	229.457364	264.9544
6	7	153.012048	185.567909	121.428571	309.594883	88.235294	66.838683	111.581377	133.333333	297.674419	...	139.759036	102.986702	102.380952	298.507463	94.117647	69.893694	160.831089	166.666667	235.658915	281.8006
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
20626	196	274.698795	234.924787	290.476190	68.230277	223.529412	77.737641	262.716429	300.000000	108.527132	...	302.409639	228.907783	302.380952	73.347548	200.000000	82.010527	302.577915	200.000000	74.418605	131.5658
20627	197	280.722892	291.781121	283.333333	84.434968	200.000000	75.467024	290.881108	233.333333	49.612403	...	265.060241	252.888598	326.190476	86.993603	235.294118	88.677882	349.365140	200.000000	0.000000	164.6506
20628	198	266.265060	273.991716	295.238095	112.579957	211.764706	84.859119	368.911120	333.333333	93.023256	...	274.698795	234.924787	290.476190	68.230277	223.529412	77.737641	262.716429	300.000000	108.527132	43.8000
20629	199	243.373494	298.408546	366.666667	83.582090	205.882353	81.226133	329.357445	233.333333	46.511628	...	280.722892	291.781121	283.333333	84.434968	200.000000	75.467024	290.881108	233.333333	49.612403	146.4788
20630	200	318.072289	255.853499	321.428571	52.025586	223.529412	77.180308	275.028857	266.666667	71.317829	...	266.265060	273.991716	295.238095	112.579957	211.764706	84.859119	368.911120	333.333333	93.023256	21.5962

20629 rows × 33 columns

In [57]:
```
#Scaling test dataset by using the parameters of training data set and bringing into [0,400] range
maxValues_train_lag = train_lag_features.max()#Max values of train data set with lag features
minValues_train_lag  = train_lag_features.min()#Min values of train data set with lag features
```

```
test_lag_features_scaled = test_lag_features.copy()
features_list = ['sensor 2', 'sensor 3', 'sensor 11', 'sensor 12', 'sensor 13', 'sensor 14', 'sensor 15', 'sensor 17', 'sensor 20', 'sensor 21']
time_lag = ['' , '(t-1)' , '(t-2)']
for i in range(3):
     for feature_name in features_list:
          test_lag_features_scaled[[feature_name+time_lag[i]]] = 400* (test_lag_features_scaled[[feature_name+time_lag[i]]] - minValues_train_lag[feature_name+time_lag[i]]) / (maxValues_train_lag[featur
test_lag_features_scaled
```

Out[57]:

	cycle	sensor 2	sensor 3	sensor 11	sensor 12	sensor 13	sensor 14	sensor 15	sensor 17	sensor 20	...	sensor 2(t-2)	sensor 3(t-2)	sensor 11(t-2)	sensor 12(t-2)	sensor 13(t-2)	sensor 14(t-2)	sensor 15(t-2)	sensor 17(t-2)	sensor 20(t-2)	se 21
2	3	150.602410	138.652714	154.761905	279.744136	88.235294	62.256167	183.455175	166.666667	291.472868	...	218.072289	124.264225	83.333333	258.422175	88.235294	52.864073	123.585995	133.333333	223.255814	264.73
3	4	148.192771	114.061478	102.380952	229.424307	100.000000	68.035917	102.808773	100.000000	266.666667	...	60.240964	151.820362	154.761905	295.948827	105.882353	81.907318	85.263563	166.666667	272.868217	274.73
4	5	156.626506	140.832788	109.523810	295.095949	88.235294	61.100217	120.353982	66.666667	263.565891	...	150.602410	138.652714	154.761905	279.744136	88.235294	62.256167	183.455175	166.666667	291.472868	288.53
5	6	108.433735	70.459996	97.619048	275.479744	117.647059	56.806688	152.212389	133.333333	238.759690	...	148.192771	114.061478	102.380952	229.424307	100.000000	68.035917	102.808773	100.000000	266.666667	264.84
6	7	108.433735	107.259647	109.523810	283.155650	105.882353	72.308804	102.347057	100.000000	220.155039	...	156.626506	140.832788	109.523810	295.095949	88.235294	61.100217	120.353982	66.666667	263.565891	286.55
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
13091	194	244.578313	247.743623	200.000000	170.575693	70.588235	233.956033	225.625240	200.000000	158.139535	...	221.686747	236.668847	171.428571	196.162047	111.764706	227.680875	141.900731	233.333333	195.348837	177.07
13092	195	242.168675	214.955308	178.571429	201.279318	123.529412	228.940035	194.382455	233.333333	133.333333	...	227.710843	214.170482	173.809524	228.571429	100.000000	234.554650	181.608311	233.333333	151.937984	221.48
13093	196	268.674699	192.805755	171.428571	212.366738	94.117647	242.130251	203.155060	233.333333	148.837209	...	244.578313	247.743623	200.000000	170.575693	70.588235	233.956033	225.625240	200.000000	158.139535	167.46
13094	197	246.987952	208.851101	180.952381	225.159915	117.647059	248.818247	225.009619	233.333333	161.240310	...	242.168675	214.955308	178.571429	201.279318	123.529412	228.940035	194.382455	233.333333	133.333333	211.48
13095	198	209.638554	266.666667	226.190476	202.985075	100.000000	236.763340	254.559446	266.666667	173.643411	...	268.674699	192.805755	171.428571	212.366738	94.117647	242.130251	203.155060	233.333333	148.837209	171.72

13094 rows × 33 columns

In [58]:
```
#Linear regression with feature lag
lin_reg.fit(scaled_train_lag_features, train_lag_target)
Y_lin_reg_predict_train_lag = lin_reg.predict(scaled_train_lag_features)
Y_lin_reg_predict_test_lag = lin_reg.predict(test_lag_features_scaled)
```

In [65]:
```
rul_train_lag_model = train_lag_target.copy()
rul_train_lag_model.columns = ['RUL in Train Set']
rul_train_lag_model.insert(1, "Predicted RUL for training set with feature lag linear reg", Y_lin_reg_predict_train_lag, True)
rul_train_lag_model
```

Out[65]:

	RUL in Train Set	Predicted RUL by linear regression with feature lag
2	189.0	180.837972
3	188.0	189.871088
4	187.0	186.512300
5	186.0	187.570307
6	185.0	182.137809
...	...	...
20626	4.0	-2.095181
20627	3.0	2.079712
20628	2.0	-5.454949
20629	1.0	-11.393039
20630	0.0	-13.901772

20629 rows × 2 columns

In [66]:
```
rul_test_lag_model = test_lag_target.copy()
rul_test_lag_model.columns = ['RUL in Test Set']
rul_test_lag_model.insert(1, "Predicted RUL for test set with feature lag linear reg", Y_lin_reg_predict_test_lag, True)
rul_test_lag_model
```

Out[66]:

	RUL in Test Set	Predicted RUL for test set with feature lag linear reg
2	140.0	175.405254
3	139.0	177.876133
4	138.0	180.357253
5	137.0	183.366691
6	136.0	183.465845
...	...	...
13091	24.0	35.490773
13092	23.0	33.856045
13093	22.0	31.809773
13094	21.0	33.902024
13095	20.0	25.972119

13094 rows × 2 columns

In [94]:
```
#Fianl summary table for all regression models
rul_test_feature_lag = test_lag_target.copy().to_numpy()
rul_train_feature_lag = train_lag_target.copy().to_numpy()
feature_lag_data = {'Model':
            ['Linear Reg with feature lag (Two previous)'],
        'RMSE (Train)':
            [np.sqrt(mean_squared_error(rul_train_feature_lag,Y_lin_reg_predict_train_lag))],
        'RMSE (Test)':
            [np.sqrt(mean_squared_error(rul_test_feature_lag,Y_lin_reg_predict_test_lag))],
        'Corr with RUL (Train)':
            [np.corrcoef(rul_train_feature_lag[:,0],Y_lin_reg_predict_train_lag[:,0])[0,1]],
        'Corr with RUL (Test)':
            [np.corrcoef(rul_test_feature_lag[:,0],Y_lin_reg_predict_test_lag[:,0])[0,1]]
           }
total_regression_summary = pd.concat((total_summary, pd.DataFrame(feature_lag_data, index=['7'])), axis = 0)
total_regression_summary
```

Out[94]:

	Model	RMSE (Train)	RMSE (Test)	Corr with RUL (Train)	Corr with RUL (Test)
0	Linear Regression	40.562272	42.983388	0.808214	0.690076
1	Polynomial Regression degree 3	35.986458	40.800747	0.852666	0.725349
2	Ridge Regression	40.562272	42.983388	0.808214	0.690076
3	Lasso Regression	40.562297	42.985801	0.808214	0.690059
4	Elastic Net	40.562280	42.985193	0.808214	0.690059
5	SVM Linear Regression	47.972160	45.889857	0.805876	0.685516
6	SVM Polynomial deg 2 Regression	37.769510	42.363157	0.841626	0.718104
7	Linear Reg with feature lag (Two previous)	40.251548	42.808218	0.811473	0.692642

In [ ]: