In [1]:
```
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
plt.rcParams['figure.figsize'] = (10, 8)
```

executed in 1.42s, finished 03:40:42 2021-01-05

In [2]:
```
df = pd.read_csv('train_FD001.txt', header=None, sep=' ').dropna(how='all', axis=1)

index_names = ['id', 'cycle']
setting_names = ['setting1', 'setting2', 'setting3']
sensor_names = ['s{}'.format(i+1) for i in range(0,21)]
col_names = index_names + setting_names + sensor_names

df.columns = col_names
df
```

executed in 148ms, finished 03:40:42 2021-01-05

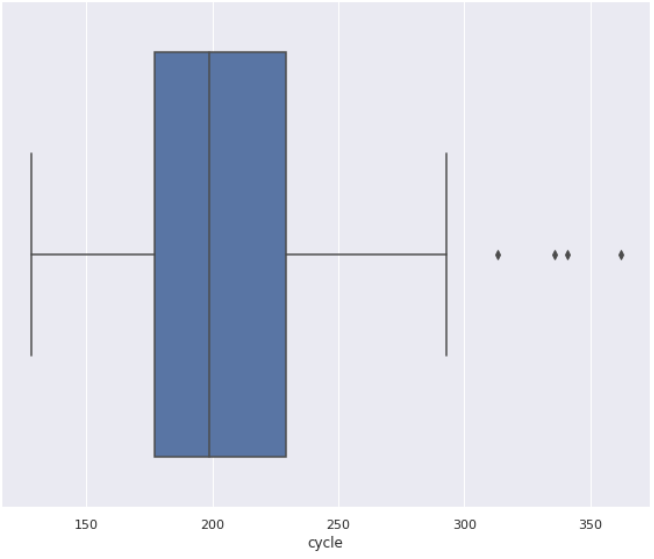
Out[2]:

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	519.49	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	519.68	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	520.01	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	519.67	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	519.30	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522

20631 rows × 26 columns

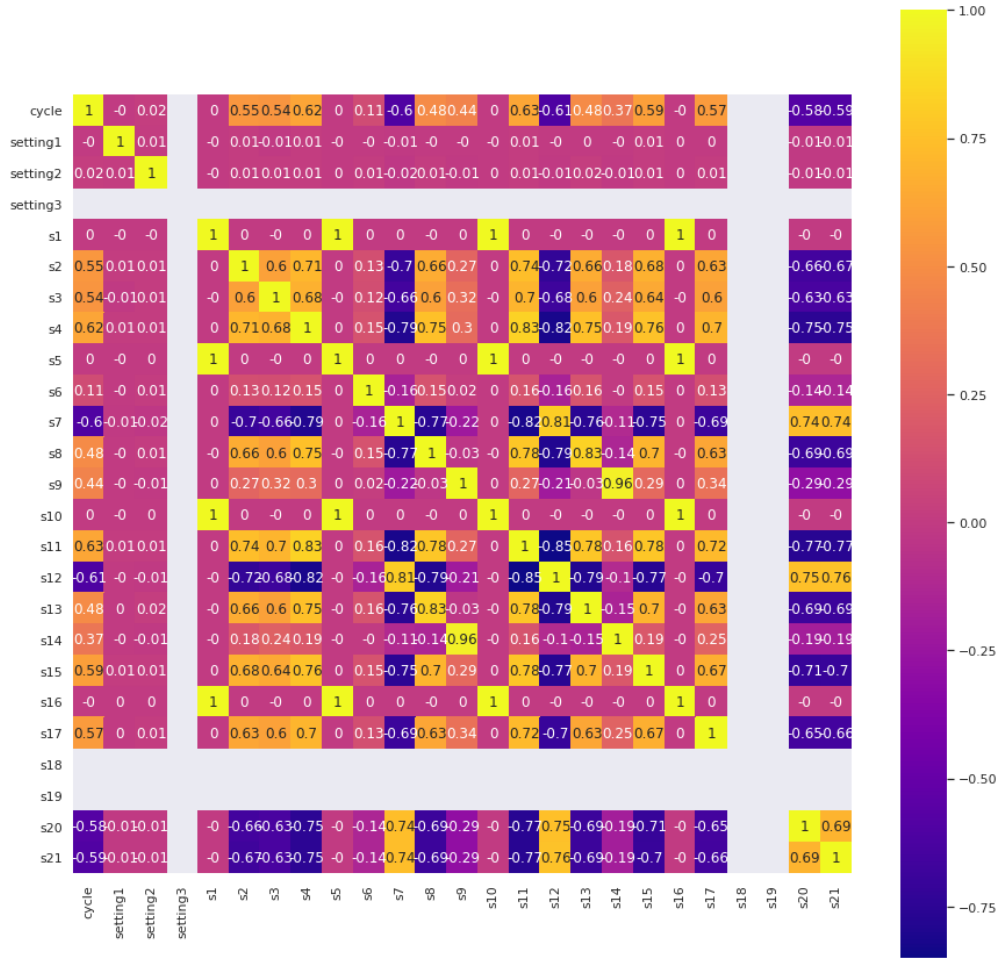
In [3]:
```
rul = df.groupby('id')['cycle'].max()
sns.boxplot(rul);
```

executed in 191ms, finished 03:40:43 2021-01-05



In [4]:
```
plt.figure(figsize=(15,15))
sns.heatmap((df.drop(columns='id').corr().round(decimals=2)), annot=True,
            square=True, cmap='plasma');
```

executed in 2.44s, finished 03:40:48 2021-01-05



In [5]:
```
df['rul'] = rul.loc[df.id].reset_index(drop=True) - df.cycle
```

executed in 7ms, finished 03:40:48 2021-01-05

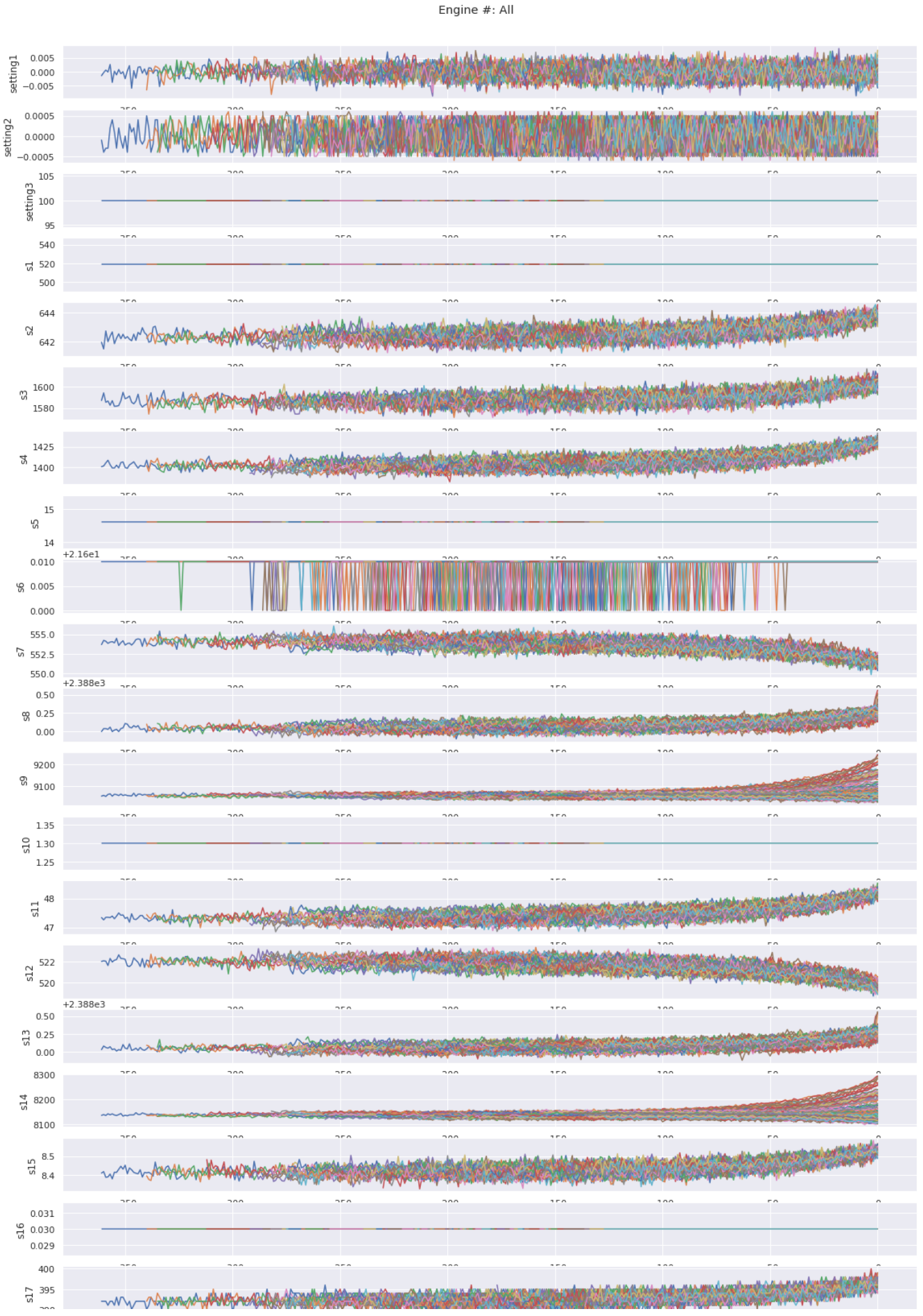
In [6]:

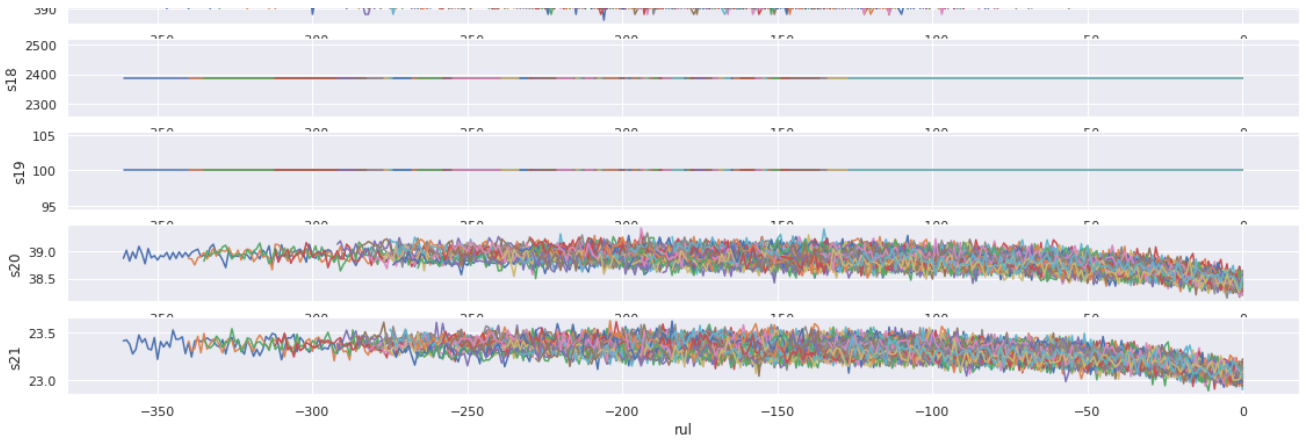
```
def plot_ts(df, engine_num='All'):
    # prepare the dataframe for plotting
    cols = df.columns[2:-1]
    fig, axes = plt.subplots(len(cols), 1, figsize=(19,35))

    if engine_num != 'All':
        for col, ax in zip(cols, axes):
            g = df[df.id==1][['cycle', col]]
            z = np.polyfit(g.cycle, g[col], 1)
            y_hat = np.poly1d(z)(g.cycle)
            sns.lineplot(x='cycle', y=col, data=g, ax=ax)
            sns.lineplot(x=g.cycle, y=y_hat, ax=ax)
            ax.set_xticks(np.arange(0,g.cycle.max(), g.cycle.max()//10))
            ax.set_xticklabels(abs(np.arange(-g.cycle.max(), 0,
                                             g.cycle.max()//10)))
            ax.set_xlabel('rul')
    else:
        g = (df.groupby(['rul', 'id']).mean().reset_index()
             .sort_values('rul', ascending=False).reset_index(drop=True))
        for col, ax in zip(cols, axes):
            for id_ in g.id.unique():
                sns.lineplot(x=-g['rul'], y=col, data=g[g.id==id_], ax=ax)

    # figure title
    fig.suptitle('Engine #: {}'.format(engine_num), y=.9);
```

executed in 7ms, finished 03:40:48 2021-01-05

In [7]:
```
plot_ts(df)
```

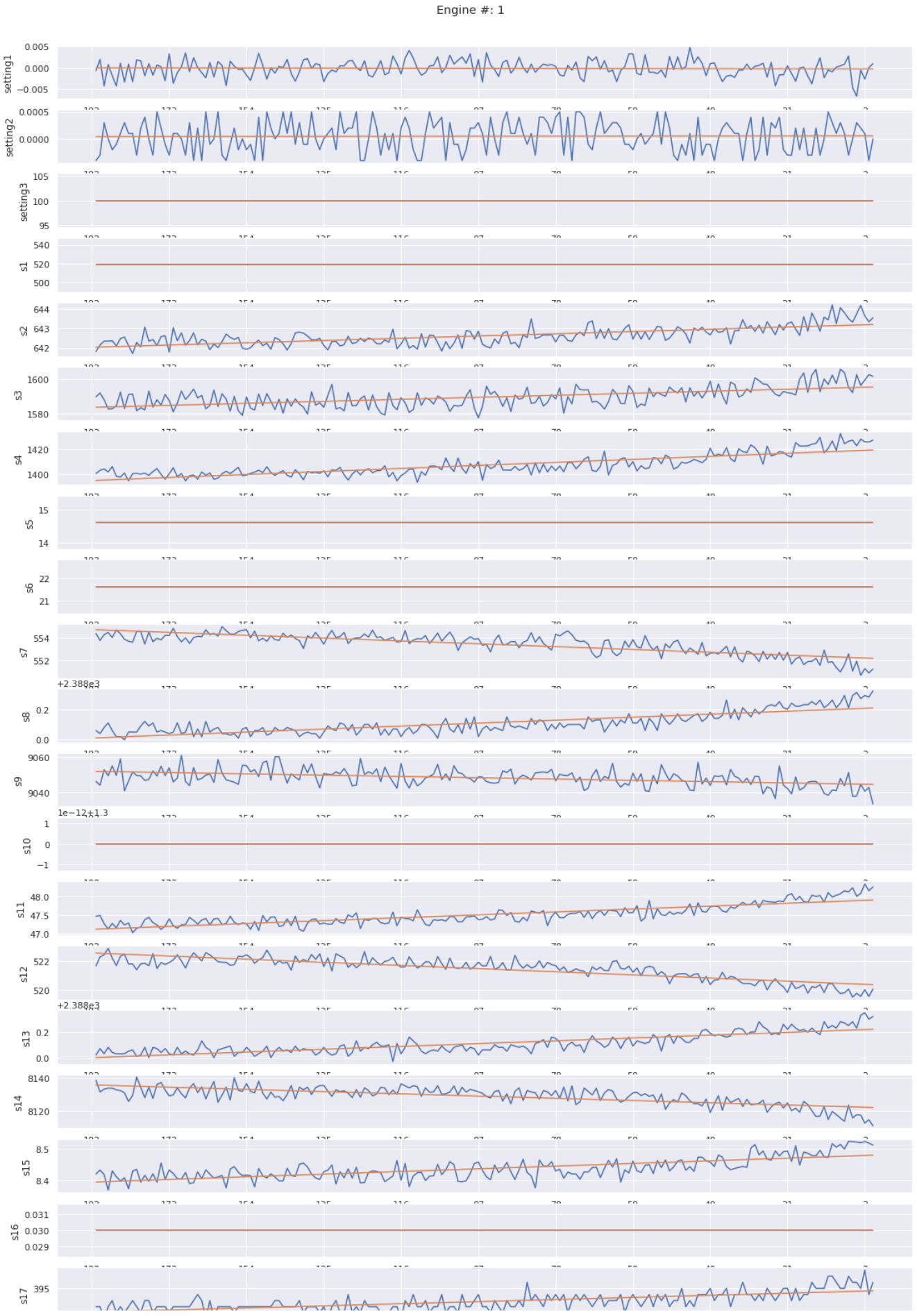
executed in 1m 13.5s, finished 02:40:42 2021-01-05

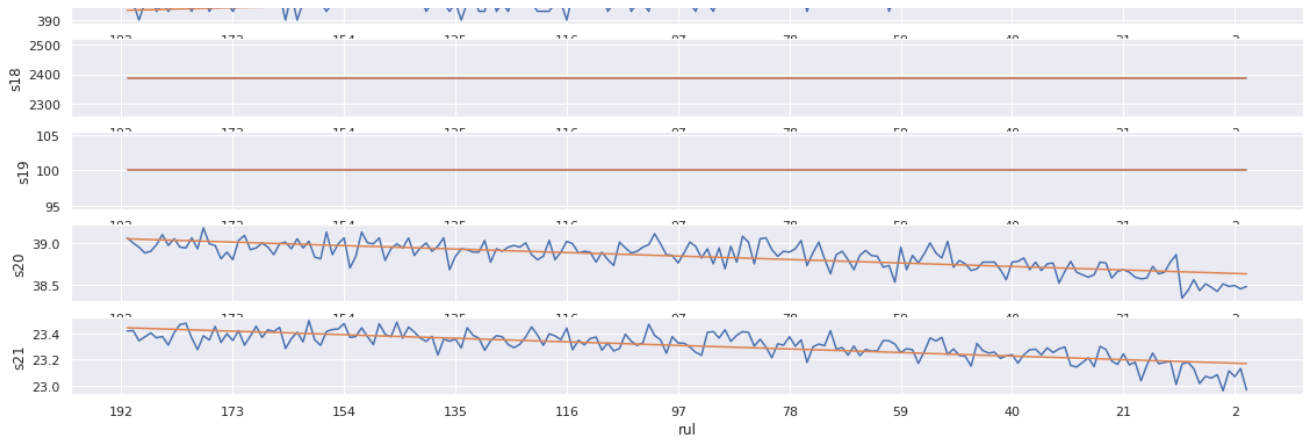



390
2500
2400
2300
s18
105
100
95
s19
39.0
38.5
s20
23.5
23.0
s21
−350
−300
−250
−200
−150
−100
−50
0
rul

In [8]:
```
plot_ts(df, 1)
```

executed in 4.66s, finished 02:41:38 2021-01-05





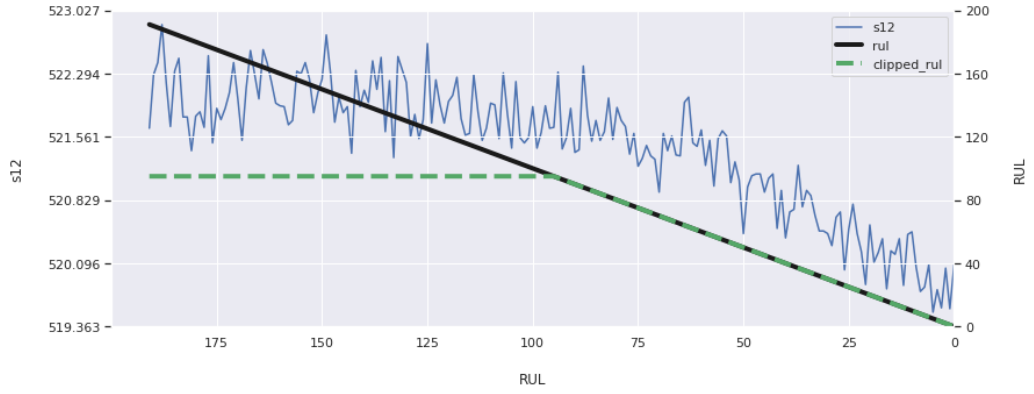
In [7]:
```
order = abs(df.drop(columns='id').corrwith(df.rul)).sort_values(ascending=False)[2:]
order2 = list(order[np.where(order>0.5)[0]].index)
order
```

executed in 46ms, finished 03:40:55 2021-01-05

Out[7]:
```
s11          6.962281e-01
s4           6.789482e-01
s12          6.719831e-01
s7           6.572227e-01
s15          6.426670e-01
s21          6.356620e-01
s20          6.294285e-01
s2           6.064840e-01
s17          6.061536e-01
s3           5.845204e-01
s8           5.639684e-01
s13          5.625688e-01
s9           3.901016e-01
s14          3.067689e-01
s6           1.283484e-01
setting1     3.198458e-03
setting2     1.947628e-03
s5           3.969701e-16
s16          3.969701e-16
setting3              NaN
s1                    NaN
s10                   NaN
s18                   NaN
s19                   NaN
dtype: float64
```

In [10]:
```
clipped_rul = df.loc[df['id']==1].copy()
clipped_rul['rul'].clip(upper=95, inplace=True)

fig, ax1 = plt.subplots(1,1, figsize=(13,5))

signal = ax1.plot('rul', 's12', 'b',
                  data=df.loc[df['id']==1])
plt.xlim(200, 0)  # reverse the x-axis so RUL counts down to zero
plt.xticks(np.arange(0, 200, 25))
ax1.set_ylabel('s12', labelpad=20)
ax1.set_xlabel('RUL', labelpad=20)

ax2 = ax1.twinx()
rul_line = ax2.plot('rul', 'rul', 'k', linewidth=4,
                    data=df.loc[df['id']==1])
rul = df.loc[df['id']==1, 'rul']
rul_line2 = ax2.plot(rul, rul.where(rul <= 95, 95), '--g', linewidth=4, label='clipped_rul')  # SET LABEL MANUALLY?
ax2.set_ylabel('RUL', labelpad=20)

# code to have equal spacing of y ticks for both axes, so the gridlines allign
ax2.set_ylim(0, 200)  # set limits of axis you want to display neatly
ax2.set_yticks(
    np.linspace(ax2.get_ybound()[0], ax2.get_ybound()[1], 6))  # choose integer to neatly divide your axis, in our case 6
ax1.set_yticks(
    np.linspace(ax1.get_ybound()[0], ax1.get_ybound()[1], 6))  # apply same spacing to other axis


# code to have a unified legend
lines = signal+rul_line+rul_line2
labels = [line.get_label() for line in lines]
ax1.legend(lines, labels, loc=0)

plt.show()
```

executed in 297ms, finished 02:41:53 2021-01-05



In [8]:
```
test = pd.read_csv('test_FD001.txt', header=None, sep=' ').dropna(how='all', axis=1)
test.columns = col_names
```

executed in 76ms, finished 03:40:59 2021-01-05

In [9]:
```
X_train = df[['id', 'cycle']+order2]
y_train = df.rul

X_test = test.groupby('id').last().reset_index()[['id', 'cycle']+order2]
```

executed in 23ms, finished 03:41:00 2021-01-05

In [10]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = MinMaxScaler()
sensors_scaled = scaler.fit_transform(X_train.drop(columns=['id', 'cycle']))
```

executed in 796ms, finished 03:41:01 2021-01-05

In [11]:
```
X_train_scaled = pd.concat([X_train[['id', 'cycle']],
                            pd.DataFrame(sensors_scaled,
                                         columns=X_train.columns[2:])], 1)
X_test_scaled = pd.concat([X_test[['id', 'cycle']],
                           pd.DataFrame(scaler.transform(
                                        X_test.drop(columns=['id', 'cycle'])),
                                        columns=X_test.columns[2:])], 1)
```

executed in 10ms, finished 03:41:01 2021-01-05

In [12]:
```
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.linear_model import LinearRegression

GBM = GradientBoostingRegressor()
GBM.fit(X_train_scaled,y_train)

y_pred_GBM = GBM.predict(X_test_scaled)

RF = RandomForestRegressor()
RF.fit(X_train_scaled,y_train)

y_pred_RF = RF.predict(X_test_scaled)

LR = LinearRegression()
LR.fit(X_train_scaled,y_train)

y_pred_LR = LR.predict(X_test_scaled)
```

executed in 20.5s, finished 03:41:22 2021-01-05

In [140]:
```
truth = pd.read_fwf('RUL_FD001.txt', sep=' ', header=None)
```

executed in 13ms, finished 04:30:26 2021-01-05

In [14]:
```
from sklearn.metrics import r2_score, mean_squared_error

def evaluate(y_true, y_hat, label='test'):
    mse = mean_squared_error(y_true, y_hat)
    rmse = np.sqrt(mse)
    variance = r2_score(y_true, y_hat)
    print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))
```

executed in 4ms, finished 03:41:23 2021-01-05

In [15]:
```
print('GBM')
evaluate(df.rul, GBM.predict(X_train_scaled), 'train')
evaluate(truth[0].values, GBM.predict(X_test_scaled))

print('\nRF')
evaluate(df.rul, RF.predict(X_train_scaled), 'train')
evaluate(truth[0].values, RF.predict(X_test_scaled))

print('\nLR')
evaluate(df.rul, LR.predict(X_train_scaled), 'train')
evaluate(truth[0].values, LR.predict(X_test_scaled))
```

executed in 460ms, finished 03:41:24 2021-01-05

```
GBM
train set RMSE:23.26120790703591, R2:0.8859522257614105
test set RMSE:32.976327482927374, R2:0.3702835615026916

RF
train set RMSE:5.587904247923458, R2:0.9934185710200077
test set RMSE:37.3666886812305, R2:0.19144478367098483

LR
train set RMSE:39.76776517274139, R2:0.6666621274831255
test set RMSE:32.33996686227092, R2:0.39435296292622124
```

In [16]:
```
y_train_clipped = y_train.clip(upper=125)

GBM = GradientBoostingRegressor()
GBM.fit(X_train_scaled,y_train_clipped)

y_pred_GBM = GBM.predict(X_test_scaled)

RF = RandomForestRegressor()
RF.fit(X_train_scaled,y_train_clipped)

y_pred_RF = RF.predict(X_test_scaled)

LR = LinearRegression()
LR.fit(X_train_scaled,y_train_clipped)

y_pred_LR = LR.predict(X_test_scaled)
```

executed in 20.1s, finished 03:41:44 2021-01-05

In [17]:
```
print('GBM')
evaluate(df.rul, GBM.predict(X_train_scaled), 'train')
evaluate(truth[0].values, GBM.predict(X_test_scaled))

print('\nRF')
evaluate(df.rul, RF.predict(X_train_scaled), 'train')
evaluate(truth[0].values, RF.predict(X_test_scaled))

print('\nLR')
evaluate(df.rul, LR.predict(X_train_scaled), 'train')
evaluate(truth[0].values, LR.predict(X_test_scaled))
```

executed in 427ms, finished 03:41:45 2021-01-05

```
GBM
train set RMSE:46.404713828803644, R2:0.5461143153081366
test set RMSE:21.603339389320976, R2:0.729739713542825

RF
train set RMSE:43.79988729728767, R2:0.5956399198105808
test set RMSE:23.2729984746272, R2:0.6863501112070071

LR
train set RMSE:49.570833118562405, R2:0.4820656391699114
test set RMSE:23.53847639939435, R2:0.679153613448957
```

In [99]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import (LSTM, Dense, Dropout, Masking,
                                     TimeDistributed, Flatten)
from keras.optimizers import RMSprop
```

executed in 6ms, finished 04:19:44 2021-01-05

In [84]:
```
from rul_codes import *
```

executed in 5ms, finished 04:10:12 2021-01-05

In [25]:
```
X_train['rul'] = y_train_clipped

remaining_sensors = ['s'+i[1:] for i in order2]
drop_sensors = [element for element in sensor_names if element not in
                remaining_sensors]
```

executed in 8ms, finished 03:43:34 2021-01-05

In [82]:
```
X_train = exponential_smoothing(X_train, remaining_sensors, 0, 0.4)
X_test = exponential_smoothing(X_test, remaining_sensors, 0, 0.4)
```

executed in 438ms, finished 04:09:48 2021-01-05

In [129]:
```
from sklearn.model_selection import GroupShuffleSplit

sequence_length = 20

gss = GroupShuffleSplit(n_splits=1, train_size=0.8, random_state=42)
for train_unit, val_unit in gss.split(X_train['id'].unique(),
                                      groups=X_train['id'].unique()):
    train_unit = X_train['id'].unique()[train_unit]
    val_unit = X_train['id'].unique()[val_unit]

    train_split_array = gen_data_wrapper(X_train, sequence_length,
                                         remaining_sensors, train_unit)
    train_split_label = gen_label_wrapper(X_train, sequence_length, ['rul'],
                                          train_unit)

    val_split_array = gen_data_wrapper(X_train, sequence_length,
                                       remaining_sensors, val_unit)
    val_split_label = gen_label_wrapper(X_train, sequence_length, ['rul'],
                                        val_unit)

# create sequences train, test
train_array = gen_data_wrapper(X_train, sequence_length, remaining_sensors)
label_array = gen_label_wrapper(X_train, sequence_length, ['rul'])

test_gen = (list(gen_test_data(X_train[X_train['id']==unit_nr],
                               sequence_length, remaining_sensors, -30.))
           for unit_nr in X_train['id'].unique())
test_array = np.concatenate(list(test_gen)).astype(np.float32)
```

executed in 677ms, finished 04:26:15 2021-01-05

In [146]:
```
del model
model = Sequential()
model.add(Masking(mask_value=-99., input_shape=(sequence_length,
                                                train_array.shape[2])))
# model.add(Flatten(input_shape=(sequence_length,
#                                                 train_array.shape[2])))
model.add(LSTM(64, activation='relu'))
model.add(Dense(32))
model.add(Dense(1))


rmse = tf.keras.metrics.RootMeanSquaredError()
model.compile(loss='mean_squared_error', optimizer=RMSprop(),
              metrics=[rmse])
```

executed in 111ms, finished 04:32:26 2021-01-05

In [147]:
```
history = model.fit(train_split_array, train_split_label,
                    validation_data=(val_split_array, val_split_label),
                    epochs=5,
                   batch_size=32)
```

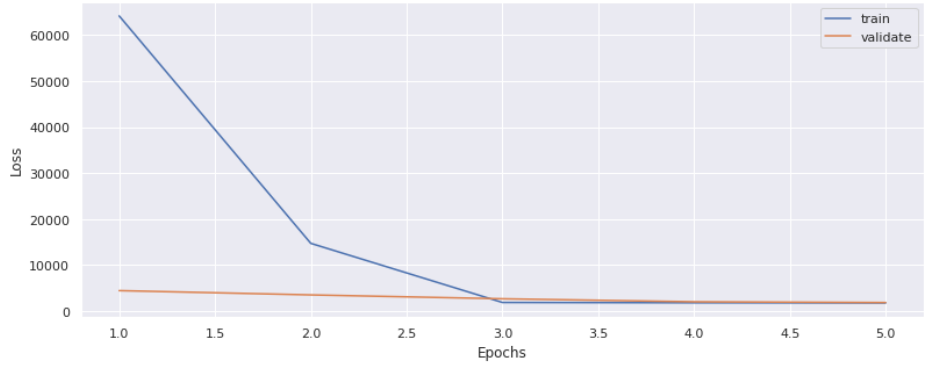
executed in 2m 20s, finished 04:34:52 2021-01-05

```
Epoch 1/5
471/471 [==============================] - 30s 61ms/step - loss: 187249.0674 - root_mean_squared_error: 382.4785 - val_loss: 4452.8901 - val_root_mea
n_squared_error: 66.7300
Epoch 2/5
471/471 [==============================] - 28s 60ms/step - loss: 34938.8769 - root_mean_squared_error: 177.3010 - val_loss: 3521.8838 - val_root_mean
_squared_error: 59.3455
Epoch 3/5
471/471 [==============================] - 27s 58ms/step - loss: 1912.7019 - root_mean_squared_error: 43.7332 - val_loss: 2677.6362 - val_root_mean_s
quared_error: 51.7459
Epoch 4/5
471/471 [==============================] - 27s 58ms/step - loss: 1815.1530 - root_mean_squared_error: 42.6018 - val_loss: 2026.2130 - val_root_mean_s
quared_error: 45.0135
Epoch 5/5
471/471 [==============================] - 27s 57ms/step - loss: 1782.2017 - root_mean_squared_error: 42.2150 - val_loss: 1861.0273 - val_root_mean_s
quared_error: 43.1396
```

In [ ]:
```
history = model.fit(train_split_array, train_split_label,
                    validation_data=(val_split_array, val_split_label),
                    epochs=5,
                   batch_size=32)
```

In [148]:
```
# plot history
def plot_loss(fit_history):
    plt.figure(figsize=(13,5))
    plt.plot(range(1, len(fit_history.history['loss'])+1), fit_history.history['loss'], label='train')
    plt.plot(range(1, len(fit_history.history['val_loss'])+1), fit_history.history['val_loss'], label='validate')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

plot_loss(history)
```

executed in 220ms, finished 04:34:54 2021-01-05



In [149]:
```
# predict and evaluate
y_hat_train = model.predict(train_array)
evaluate(label_array, y_hat_train, 'train')

y_hat_test = model.predict(test_array)
evaluate(truth, y_hat_test)
```

executed in 4.73s, finished 04:35:00 2021-01-05

```
train set RMSE:42.993988037109375, R2:-0.05571037087489472
test set RMSE:45.000595707791035, R2:-0.17267315033865982
```

In [150]:
```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(sensors_scaled)
100*pca.explained_variance_ratio_
```

executed in 27ms, finished 04:35:21 2021-01-05

Out[150]:
```
array([74.87714026,  3.40879361,  3.29685633,  2.85733537,  2.77984779,
        2.56149157,  2.47501371,  1.97511338,  1.76706345,  1.59930534,
        1.56499601,  0.83704317])
```

In [151]:
```
pca = PCA(n_components=3, whiten=True)
sensors_pca = pca.fit_transform(sensors_scaled[:, :6])
100*pca.explained_variance_ratio_
```

executed in 233ms, finished 04:35:27 2021-01-05

Out[151]:
```
array([81.91677609,  4.81393047,  4.16760062])
```

In [ ]: