**# Solution for the predictive maintenance tutorial**

This part reveals how the dataset was created and how the example solution was done.

**## How the dataset was created**

Failure was followed always when all these conditions were true:

- `avg\_curr` average current was over 7 or above.

- `temperature ` standard deviation from past 6 measurements was 4 or above.

- `pressure` mean from past 6 measurements was less than 15.

`avg\_vol` had no effect, it was totally random in relation to failures.

Each gadget was an individual unit, so measurements of the other gadgets had no impact.

**## The solution script**

Due to wicked mix of randomness and meaningless patterns in the created data, the modeling revealed to be surprisingly challenging. On the other hand, that is everyday stuff in real life.

Here is the `solution/run.py` script in a nutshell:

1. Read the csv files to two individual `pandas` `DataFrame`s

2. Merge the `DataFrames` so that each of measurement gets timestamp for the next failure for that gadget

3. If the next failure happens in less than 1 hour, tag the row as a failure

4. Calculate features

    \* For example mean of the sensor from the past 2 hours

    \* I created directly the features I knew should bring the best results

5. Split the data

    \* Gadgets 1-4 for training

    \* Gadgets 5-6 for testing

6. Train a few different machine learning models

7. Determine which model gave the best metrics against the test data:

    \* `Precision`: What percentage of the failure predictions were correct

    \* `Recall`: What percentage of the failures the model could predict

Here is some output from my `solution/run.py` script. The rate of correct predictions (precision) was from 30% to 34% and the rate of captured failures from 93% to 100% depending on the model.

```

Training data: (651, 43)

Training data: (327, 43)

-----------

random\_forest

Precision: 0.3106796116504854

Recall: 1.0

0    224

1    103

Name: random\_forest, dtype: int64

-----------

log\_regr

Precision: 0.30927835051546393

Recall: 0.9375

0    230

1     97

Name: log\_regr, dtype: int64

-----------

lin\_regr

Precision: 0.0

Recall: 0.0

0    327

Name: lin\_regr, dtype: int64

-----------

knn

Precision: 0.34285714285714286

Recall: 0.375

0    292

1     35

Name: knn, dtype: int64

-----------

nn

Precision: 0.3076923076923077

Recall: 0.125

0    314

1     13

Name: nn, dtype: int64

-----------

svm

Precision: 0.30612244897959184

Recall: 0.9375

0    229

1     98

Name: svm, dtype: int64

-----------

bayes

Precision: 0.30612244897959184

Recall: 0.9375

0    229

1     98

```

Obviously creating binary features from the exact tresholds used in the dataset creation would give 100% hit rate in all metrics.

A dimensionality reduction could be applied.