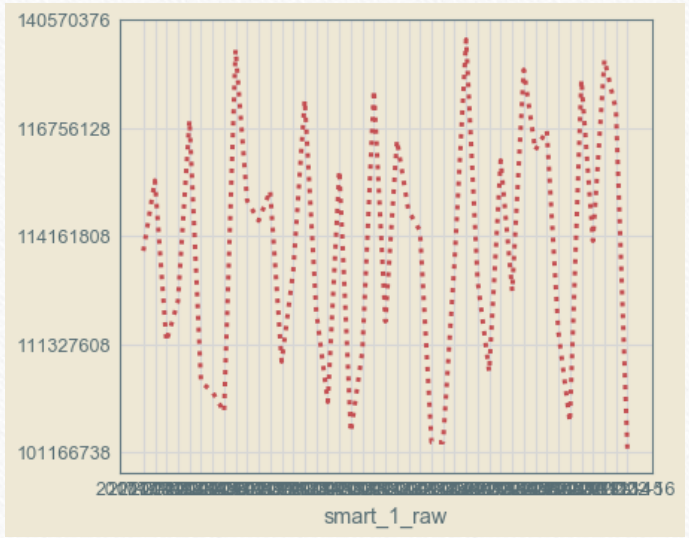
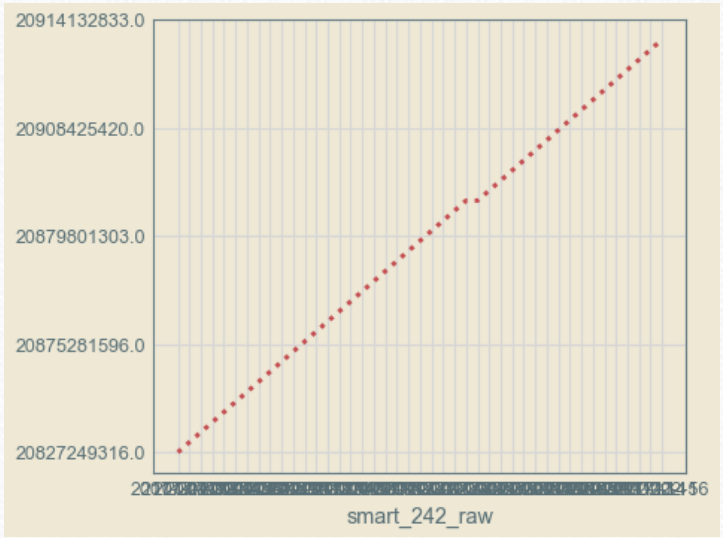
WEEK2 REPORT

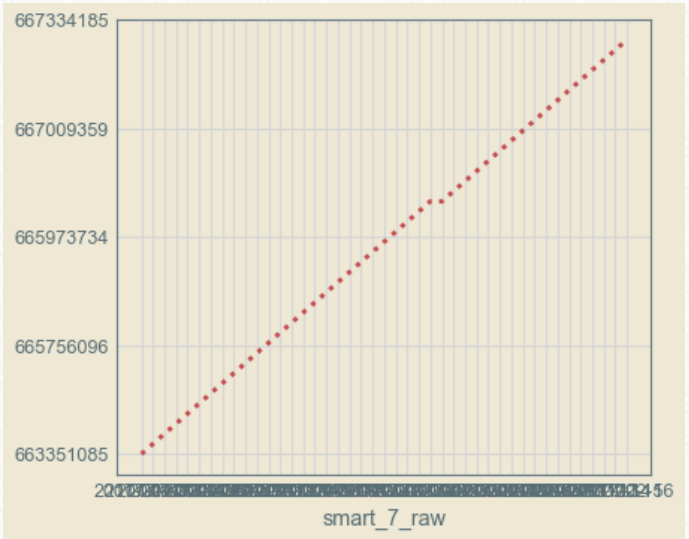
YANG zhi lin

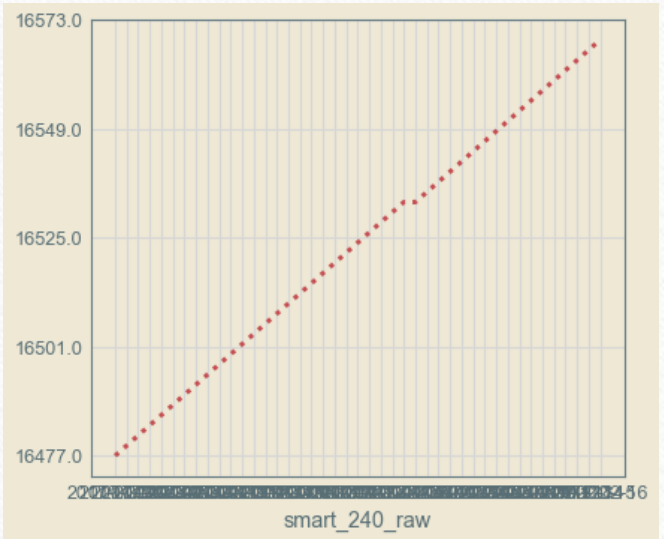
1.data preprocessing

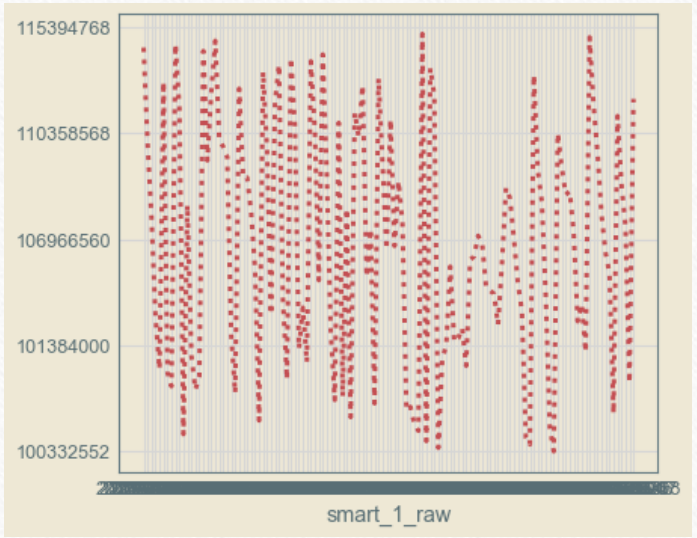
- 1) Firstly, I noticed I had made some mistakes in my last week's work. So I had to redo the total data preprocessing stage. This time I chose Seagate A model for the whole year of 2017.
- 2) Before doing change point detection, I had a doubt about one assumption in the paper: it assumes 'a significant change in attribute value' will take place before the disk failure happens. However, I think some attributes may just change gradually by time. So I chose two samples and tried to plot some of their attributes' values to see its trend.

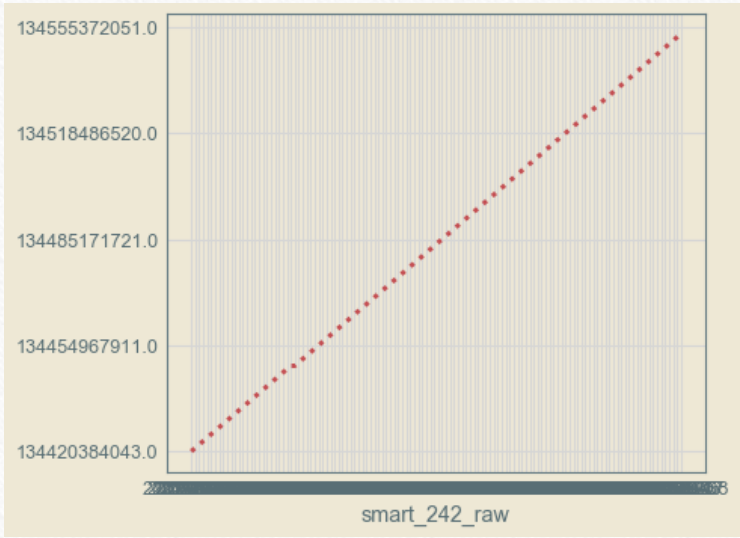

140570376
116756128
114161808
111327608
101166738
smart_1_raw

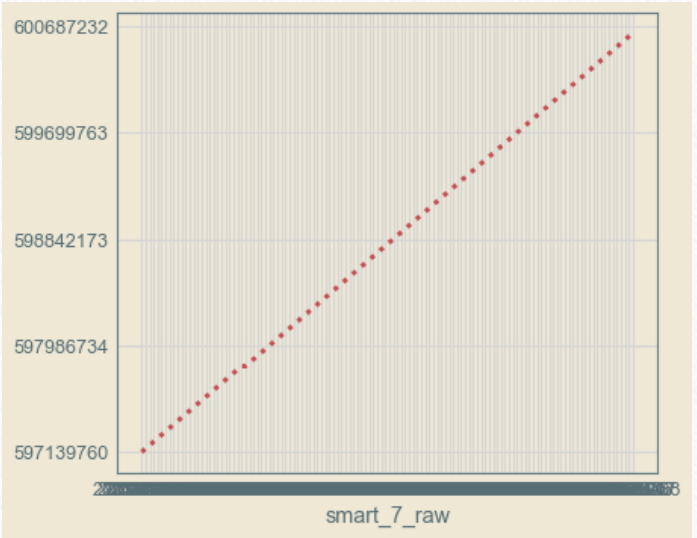

20914132833.0
20908425420.0
20879801303.0
20875281596.0
20827249316.0
smart_242_raw

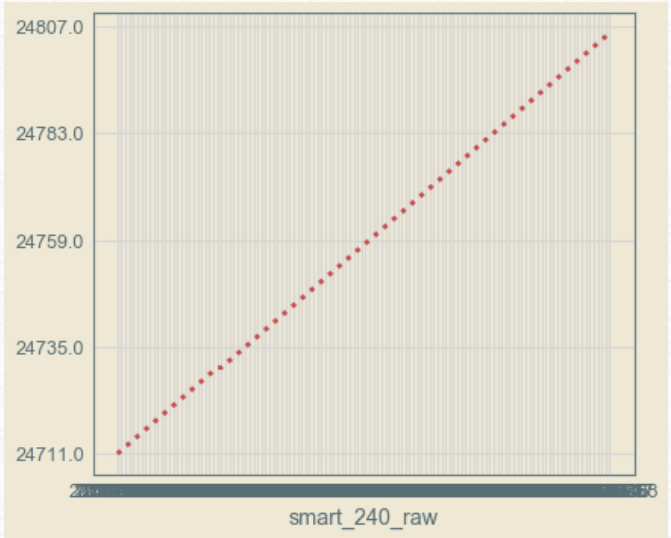

667334185
667009359
665973734
665756096
663351085
smart_7_raw


16573.0
16549.0
16525.0
16501.0
16477.0
smart_240_raw


115394768
110358568
106966560
101384000
100332552
smart_1_raw


134555372051.0
134518486520.0
134485171721.0
134454967911.0
134420384043.0
smart_242_raw


600687232
599699763
598842173
597986734
597139760
smart_7_raw


24807.0
24783.0
24759.0
24735.0
24711.0
smart_240_raw

1.data preprocessing

- These two sample: 'Z3029FAS' and 'Z3015SZN' correspond to largely different time scale: the first one failed in Feb but the second failed in Dec. Despite their largely different lifespan, **we can see that their attribute values are increasing gradually instead of making a significant shift.**
- In this case, the change point detection does not make sense. When I tried several tools to do the change point detection, they all return a long list of change points to me...So I have to skip change point detection.

2.compact time series representation

- 1. The difficulty in change point detection actually influenced this stage of work, because according to the paper, when choosing the parameter: 'window width' in the smoothing function, I should use the 'the median of the distribution of the time stamps of their corresponding significant change'.
- 2. Thus, I used the **median value of their life span** as the **parameter 'span'** in 'ewma' function.

```
In [7]: sample=df[df.iloc[:,1].str.match('Z3029FAS')]
        for i in range(1,len(sample)):
            arr=(sample['smart_7_normalized'].ewm(span=i).mean()).va
            print(arr[-1])
```

```
88.0
88.0
87.9999999999
87.9999999716
87.9999989967
87.9999897647
87.9999477379
87.9998256879
87.9995609664
87.9990900449
87.9983614947
87.9973430135
87.9960226061
87.9944059577
87.9925121118
87.9903689257
87.9880090998
87.9854670789
87.9827768406
87.9799704495
```

About the choice of parameter 'span':

- I actually tried all possible values of this parameter and find that the final smoothing results differ quite little. So I think it is reasonable to just choose the median value of this parameter.
- I will refine this step in the future...

3.downsampling

- 1) In total, there are over 30k samples. I compressed time series for 15031 samples in total. Among them are 14550 healthy samples and 1072 failed samples.
- 2) Next I handled the skewed class problem by downsampling via k-means algorithm. I ran k-means on the healthy disk set with parameter **k=100**. I chose the **top 10 closest samples to the center** for each class.
- 3)Finally the training set contains 1000 healthy samples and 480 healthy samples.

3.downsampling

- Here are a few things for further improvement:
- 1. Tuning hyperparameter k for K-Means algorithm: why use 100?
- 2.Try different clustering algorithm
- 3.Try other approaches to handle skewed classes, e.g. up-sample minority classes...

4.classification

- 1. Issue 1: How to feed 'raw values' into ML classifier? Most of them requires feature scaling beforehand. Normalisation is one way to do feature scaling. But why they still use raw values?
- 2. The point I don't understand in the paper: "We also note that it's mostly the raw values of SMART indicators that correlate with impending replacements. This is expected, since, the normalized values are computed based on generous thresholds, where a replacement can also occur before the normalized value changes at all. " ???

I first tried running Logistic Regression on the whole training set, x_train_norm and x_train_raw separately, and found a little improvement on x_train_norm.

```
In [5]: scores=cross_val_score(estimator=lr,X=X_train,y=y_train,cv=100,n_jobs=2)
        print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))

        CV accuracy: 0.662 +/- 0.096
```

```
In [7]: scores=cross_val_score(estimator=lr,X=X_train_raw,y=y_train,cv=100,n_jobs=2)
        print('x_train_raw CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))

        x_train_raw CV accuracy: 0.662 +/- 0.096
```

```
In [8]: scores=cross_val_score(estimator=lr,X=X_train_norm,y=y_train,cv=100,n_jobs=2)
        print('x_train_norm CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))

        x_train_norm CV accuracy: 0.671 +/- 0.041
```

Low precision problem

- The precision score for LR seems unreasonably low and I got a warning as shown below:
- I tried Grid Search to tune the regularization parameter for LR, no improvement
- I tried different ML algorithms, but all have this problem

```
/Users/zhilinyang/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:113
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted
samples.
  'precision', 'predicted', average, warn_for)
/Users/zhilinyang/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:113
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted
samples.
  'precision', 'predicted', average, warn_for)
```

```
CV precision: 0.045 +/- 0.201
```

ML algorithm performance

- LR CV precision: 0.045 +/- 0.201
- RF CV precision: 0.490 +/- 0.377
- DT CV precision: 0.472 +/- 0.314
- GBDT CV precision: 0.650 +/- 0.378
- SVM CV precision: 0.543 +/- 0.372
- RGF CV precision: 0.591 +/- 0.382

- I suspected the low precision across several algorithm is not from classifiers themselves but from dataset.
- I shuffled the training set to make the train test set split more balanced, but the problem persists.

Diagnose problem via learning-curve

To diagnose the problem, I plotted the learning curve for RGF classifier: the accuracy score with respect to the size of training set. There is **a huge gap** between the training accuracy and test accuracy, which suggest a **high variance** problem.