PyCon ID 2019

Introduction to Change Point Analysis

Elvyna Tunggawan
Data Scientist at **Airy**

PYCON
INDONESIA 2019

airy

Outline

1

Introduction

What is change point analysis?

2

Methodology

How do we find the “change”?

3

Conclusions

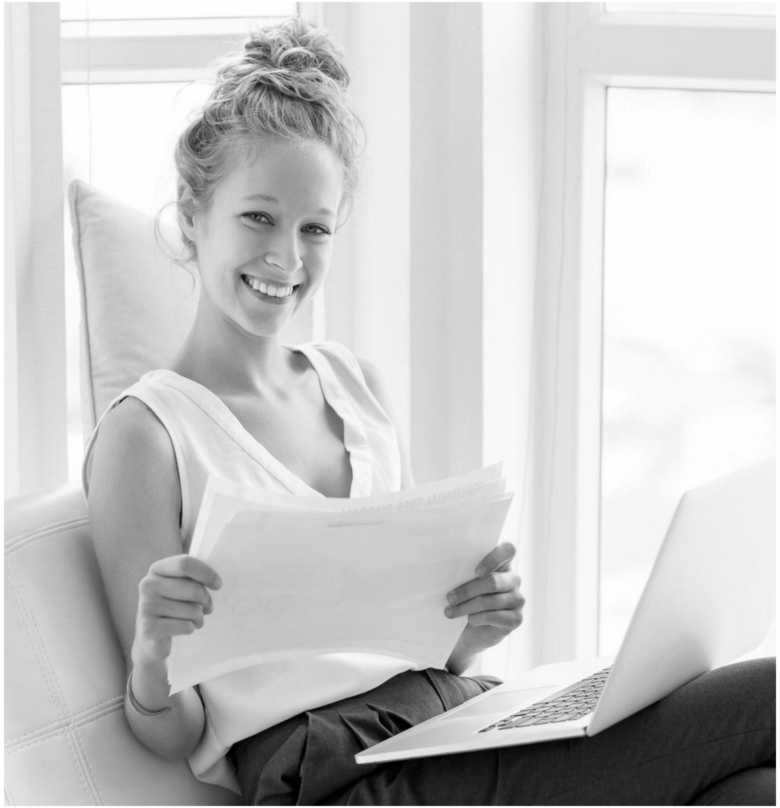
What have we learned?

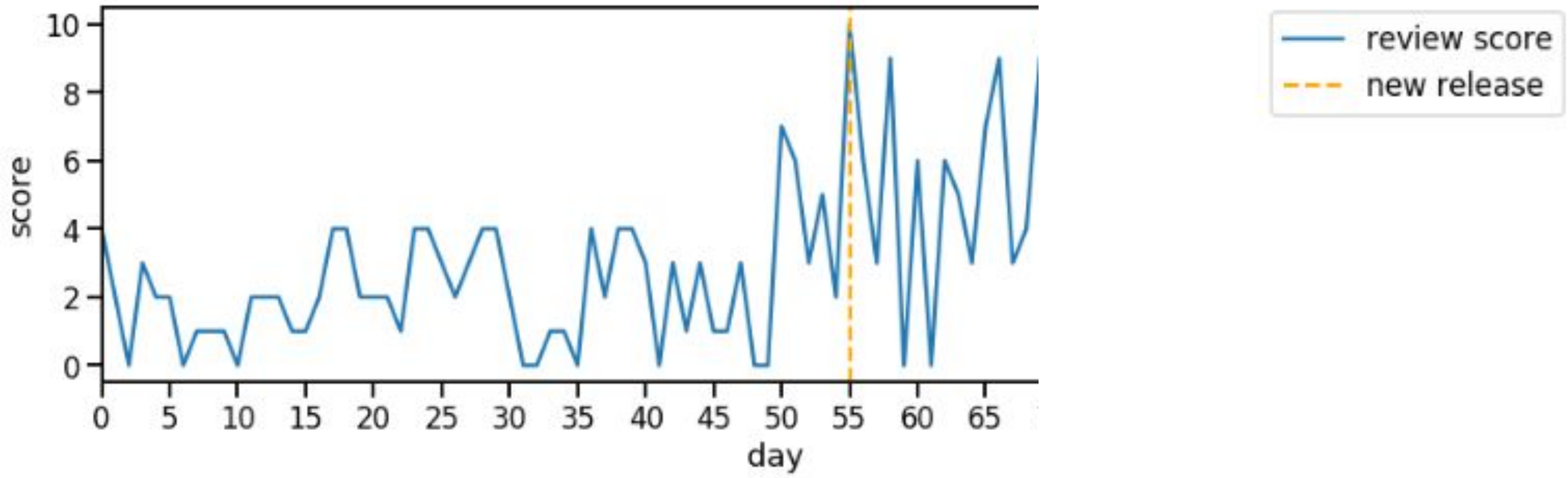
1. Introduction

What is change point analysis?

Whoa!

At the end of your peaceful Friday, a product manager came and asked a question...

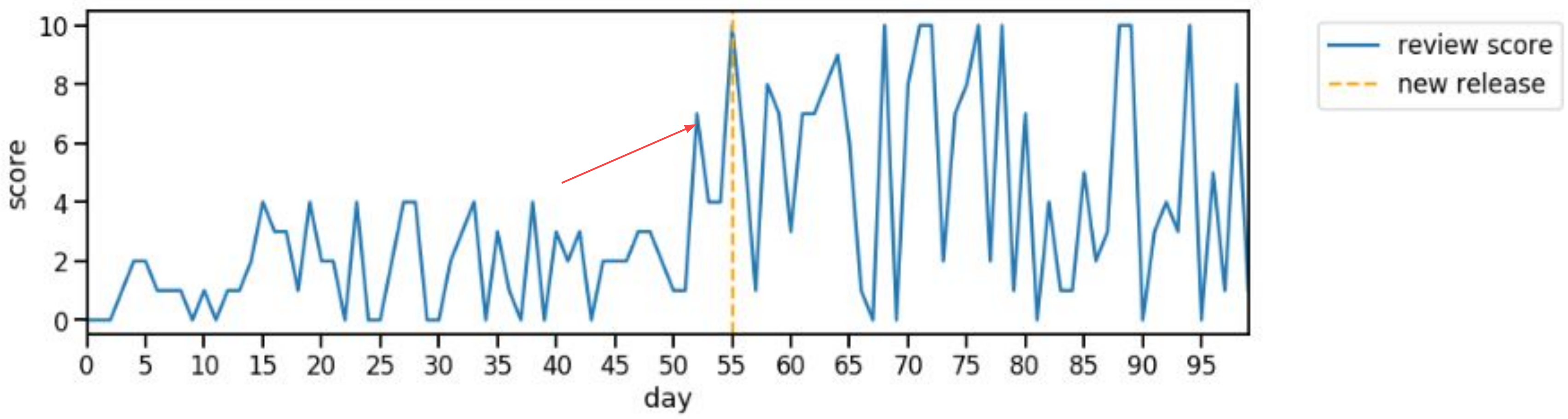




Our reviews are getting better!

It's because of our new feature release, *isn't it*?

Is it *really* getting better?



What is change point analysis?

Given a series of data, change point analysis involves ***detecting the number and*** *location of change points,* ***locations in the data where some feature****, for example the mean,* ***changes****.*

There are two types of change point analysis ...

Offline

- all data are **processed in one go**
- main goal: **accurate** detection of changes

Online

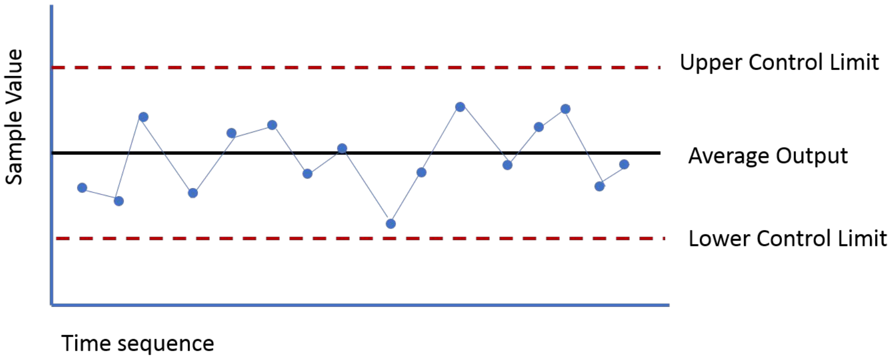
- data must be **processed quickly** “on the fly” before new data arrives
- main goal: the **quickest** detection of a change after it has occured

2. Methodology

How do we find the “change”?

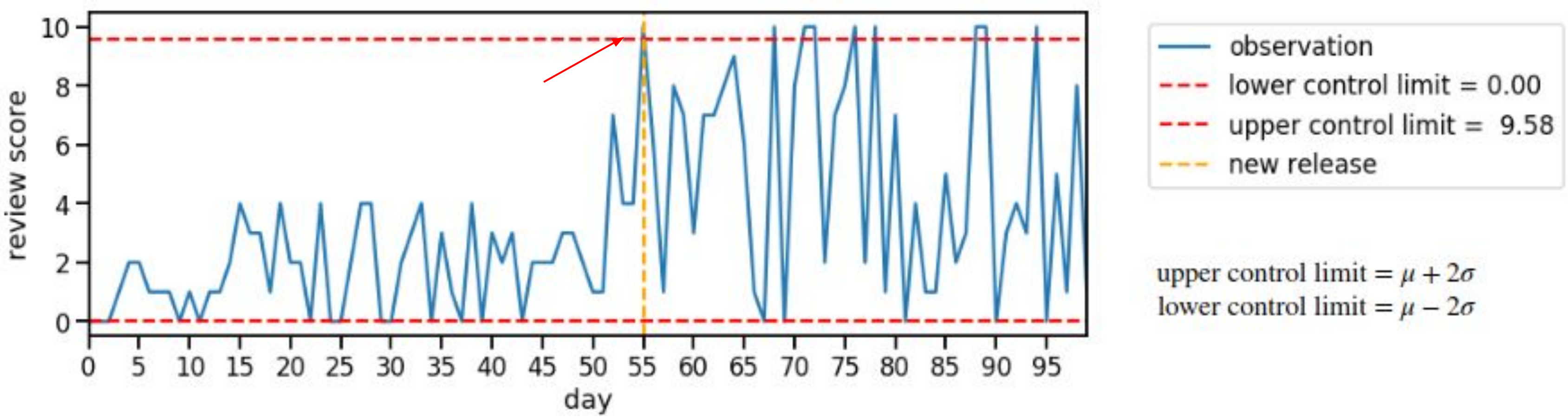
A. Control charts

- Common in **process control**
- Use average, lower & upper **control limit**
- Focus on **point-wise** error rate
- Lower & upper limit is determined based on **standard deviation**



Example of control chart (source)

Sample control chart



$$\text{upper control limit} = \mu + 2\sigma$$
$$\text{lower control limit} = \mu - 2\sigma$$

The first observation which lies above upper control limit: **day 55**.

B. Change point analysis

- Can detect **subtle changes** frequently missed by control charts
- Can be conducted **once all observations are collected**, to identify **change-wise** error rate
- Based on **mean** or **variance**

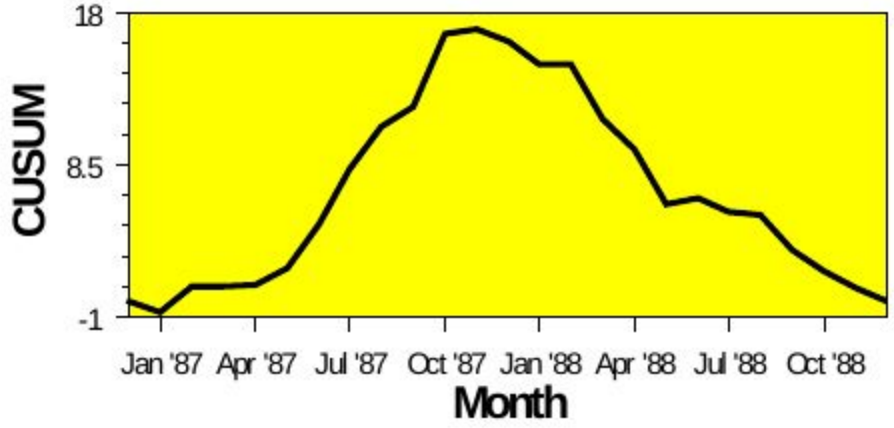
Single change point analysis

Method 1: Cumulative Sum (CUSUM)

Cumulative Sum (CUSUM)

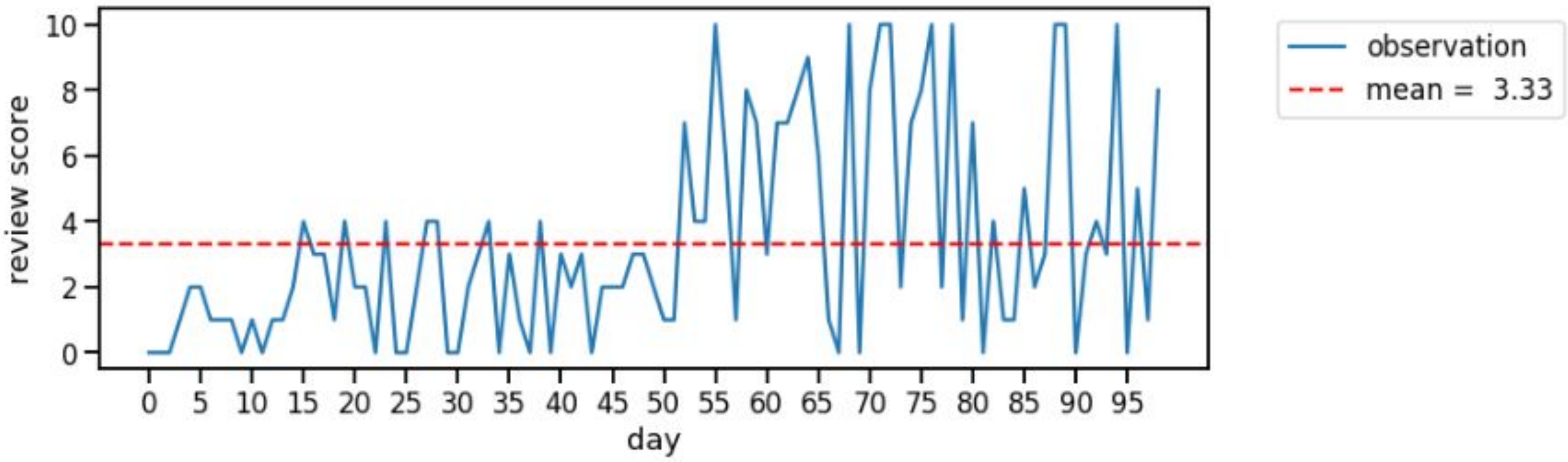
Steps:

1. Calculate **mean** value of **all observations** ($\bar{y}$)
2. Calculate **residuals**: difference between y_i and $\bar{y}$
3. Set cumulative sum of residuals at 0: $S_0 = 0$
4. Calculate **cumulative sum of residuals**: $S_i = S_{i-1} + \varepsilon_i$

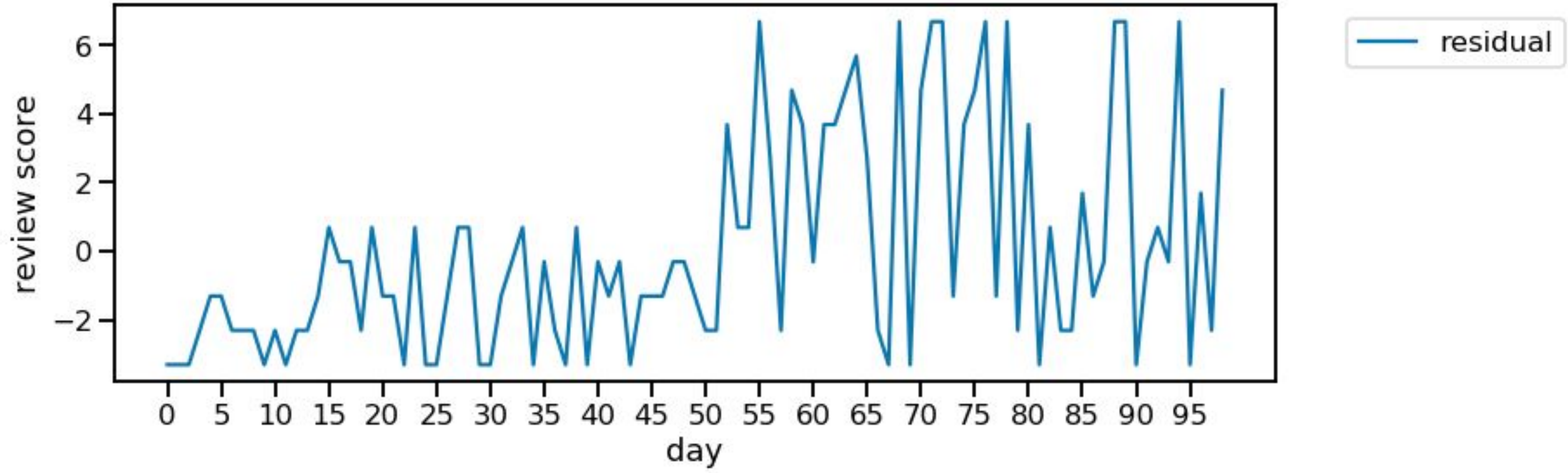


Example of CUSUM plot (source)

CUSUM: calculate mean



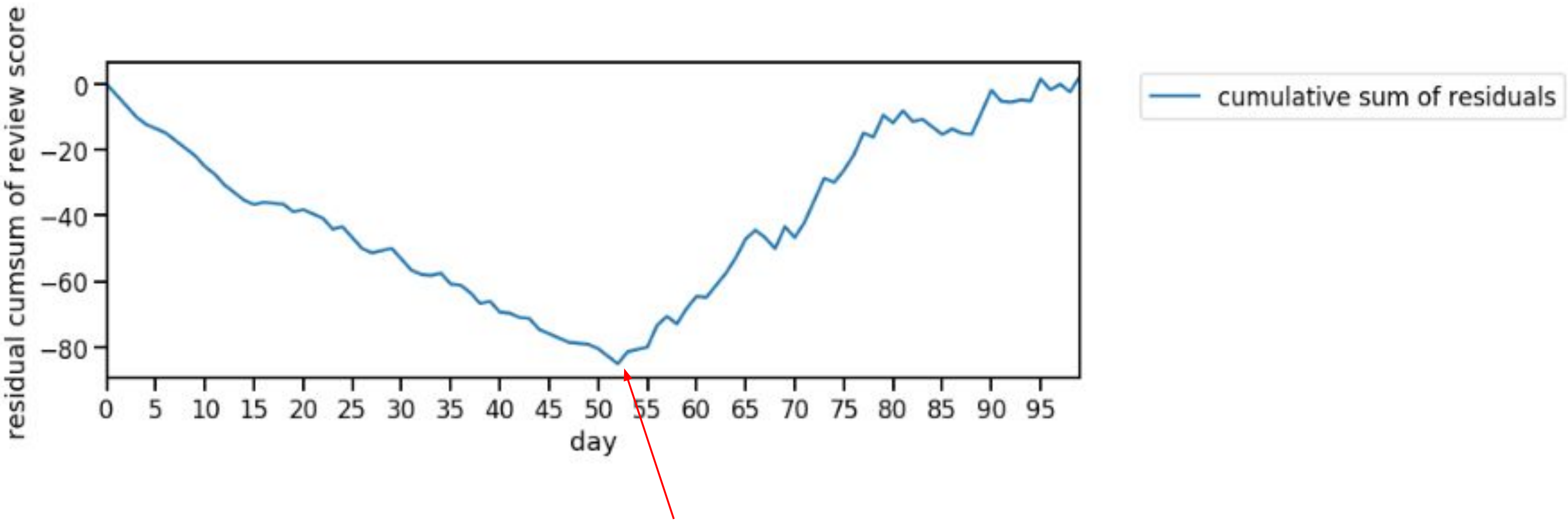
CUSUM: calculate residuals



CUSUM: calculate cumulative sum of residuals

```
def calculate_cusum_residuals(df, observation_column=0):
    mu = df[observation_column].mean()
    df = df.shift(1)
    df['residual'] = df[observation_column] - mu
    df.loc[(df.index == 0), 'residual'] = 0
    df['residual_cumsum'] = df['residual'].cumsum()
    return df
```

CUSUM: calculate cumulative sum of residuals



Sudden change is observed at **day 52**.

CUSUM: how **confident** are we that the change exists?

- **Frequentist** method
- **Sampling without replacement**: randomly reorder the observations

1. determine number of iteration N
2. for each iteration:
   - take random sample without replacement from the observations $X_1^0, X_2^0, \ldots, X_n^0$, where n = number of observations
   - calculate the sample's cumulative sum of residuals (S^0): $S_1^0, S_2^0, \ldots, S_n^0$
   - calculate the difference between maximum and minimum residuals in each bootstrap
     $$S_{\text{diff}}^0 = S_{max}^0 - S_{min}^0$$
   - If $S_{\text{diff}}^0 < S_{\text{diff}}$, it means the result from the sample is consistent with actual observations

$$\text{confidence level}(\%) = 100 * \frac{X}{N}$$

where X represents number of iteration which has $S_{\text{diff}}^0 < S_{\text{diff}}$

CUSUM: how confident are we that the change exists?

```python
def calculate_residual_difference(df):
    ## calculate difference between maximum and minimum cumsum residuals
    resid_max = df['residual_cumsum'].max()
    resid_min = df['residual_cumsum'].min()

    resid_diff = resid_max - resid_min
    return resid_diff
```

CUSUM: how confident are we that the change exists?

```
N = 1000 ## determine number of iteration
X = 0 ## occurrence when sample residual difference < observed residual difference

for i in np.arange(0,N):
    _sample = pd.DataFrame(
        np.random.choice(df[0], size = df.shape[0], replace = False)
    )
    _sample = calculate_cusum_residuals(_sample)
    _sample_resid_diff = calculate_residual_difference(_sample)

    if _sample_resid_diff < resid_diff:
        X += 1

confidence_level = 100 * X / N
print("Confidence level: {:.2f}%".format(confidence_level))
```

Single change point analysis

Method 2: Structure change model - MSE Estimator

Structure Change Model: MSE Estimator

Steps:

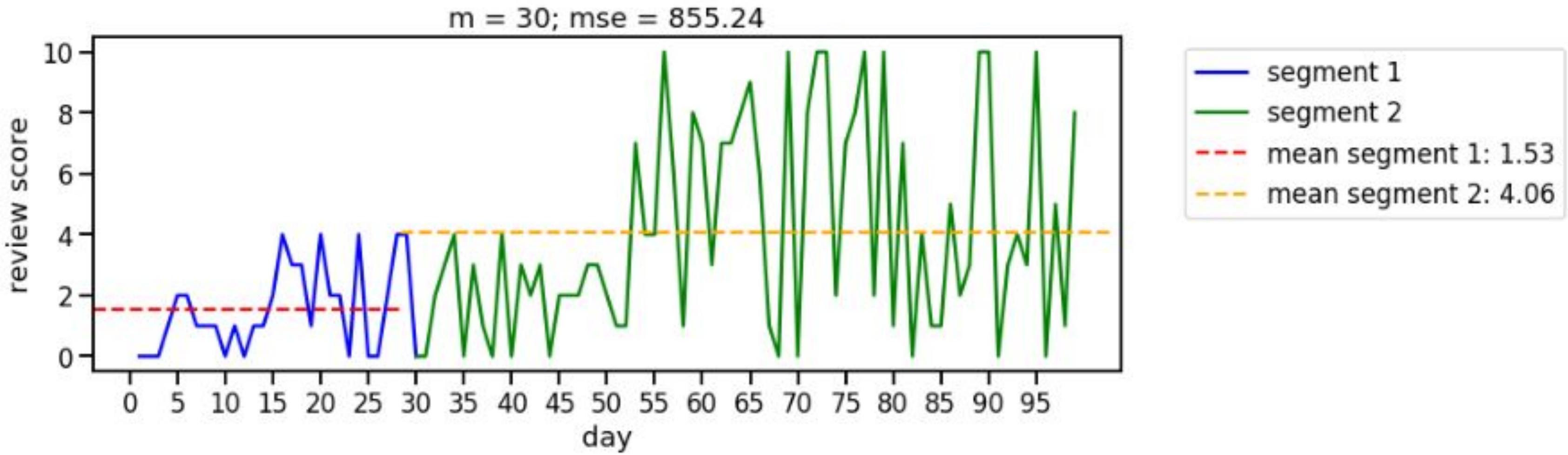
1. Split the data into **2 segments**
   - segment 1 = {1, …, m}
   - segment 2 = {m+1, …, n}
2. Calculate **average** value of each segment: $\bar{X}_1$ and $\bar{X}_2$
3. Calculate **mean squared error** of observation in **each segment**

$$MSE(m) = \sum_{i=1}^{m} (X_i - \bar{X}_1)^2 + \sum_{i=m+1}^{n} (X_i - \bar{X}_2)^2$$

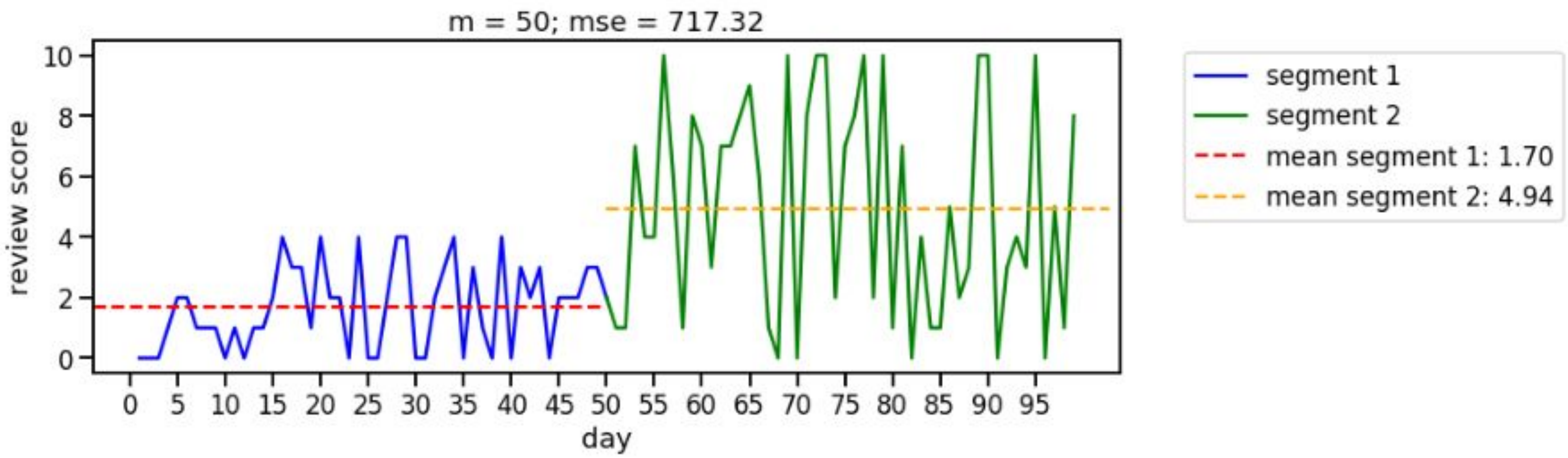
$$\text{where} \quad \bar{X}_1 = \frac{\sum_{i=1}^{m} X_i}{m} \quad \text{and} \quad \bar{X}_2 = \frac{\sum_{i=m+1}^{n} X_i}{n - m}$$

4. Value of **m** which **minimizes the MSE** is the best estimator of the **last point before the change occured**

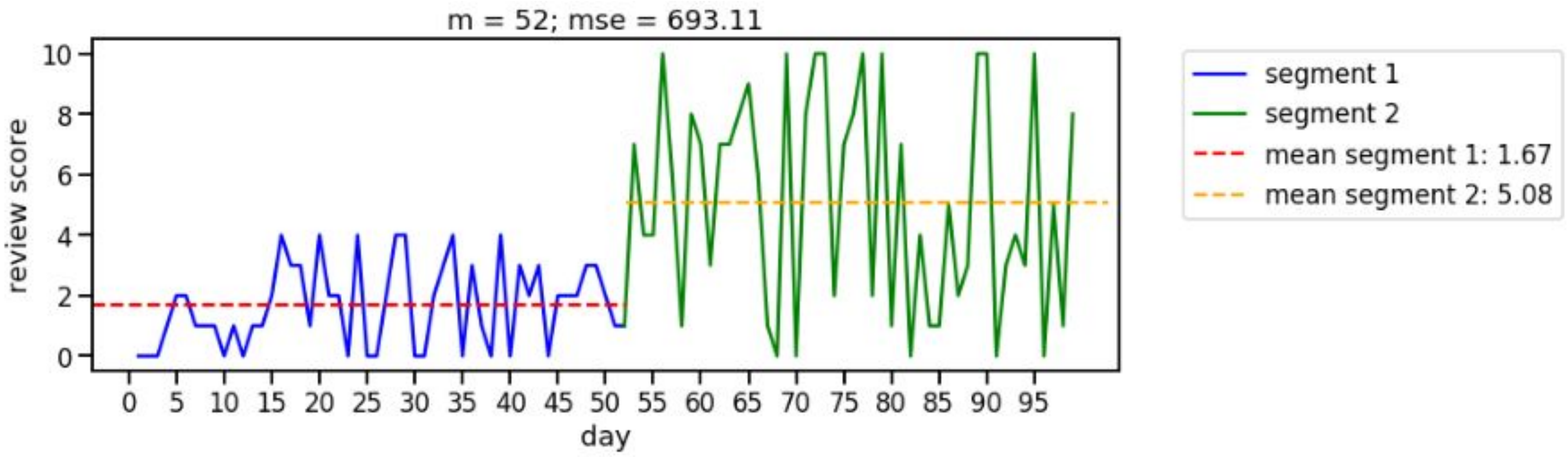
MSE estimator: intuition



MSE estimator: intuition



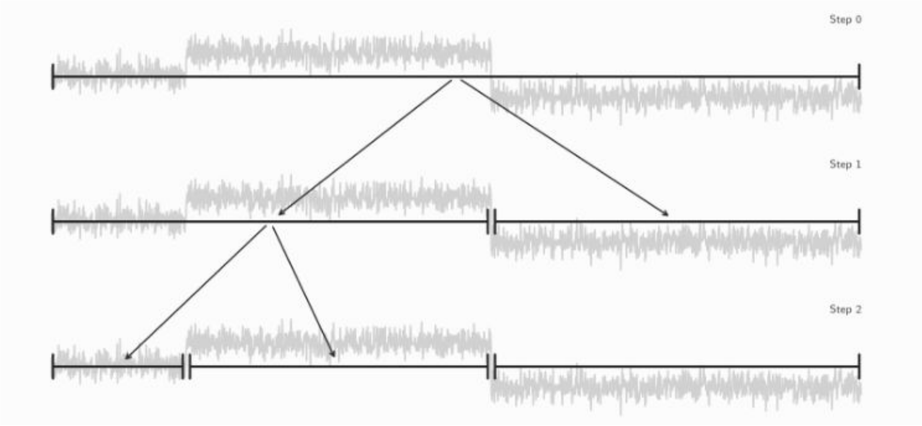
MSE estimator: intuition



Value of **m** which **minimizes the MSE** is the best estimator of the **last point before the change occured** → **day 52**

What if we’re looking for
more than one change point?

Multiple Change Point: Binary Segmentation



Schematic view of the binary segmentation algorithm (source)

Libraries

python

ruptures
bayesloop
fbProphet

changepoint
bcp
strucchange
cpm

Confused?
You can apply *Bayesian* approach too!

—Anonymous

Bayesian Approach

1. Set **prior** distribution of μ_1, μ_2, and overall σ
2. The **changepoint** could occur in $\tau \in \{1,\ldots,n\}$
3. Assign:

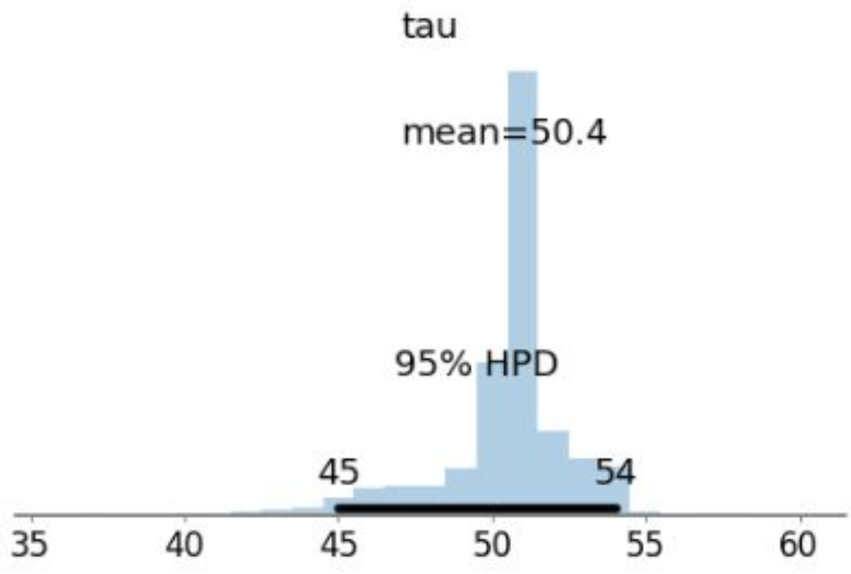
$$\mu = \begin{cases} \mu_1 & \text{if } \tau \geq t \\ \mu_2 & \text{if } \tau < t \end{cases}$$

4. Produce the **sample**!

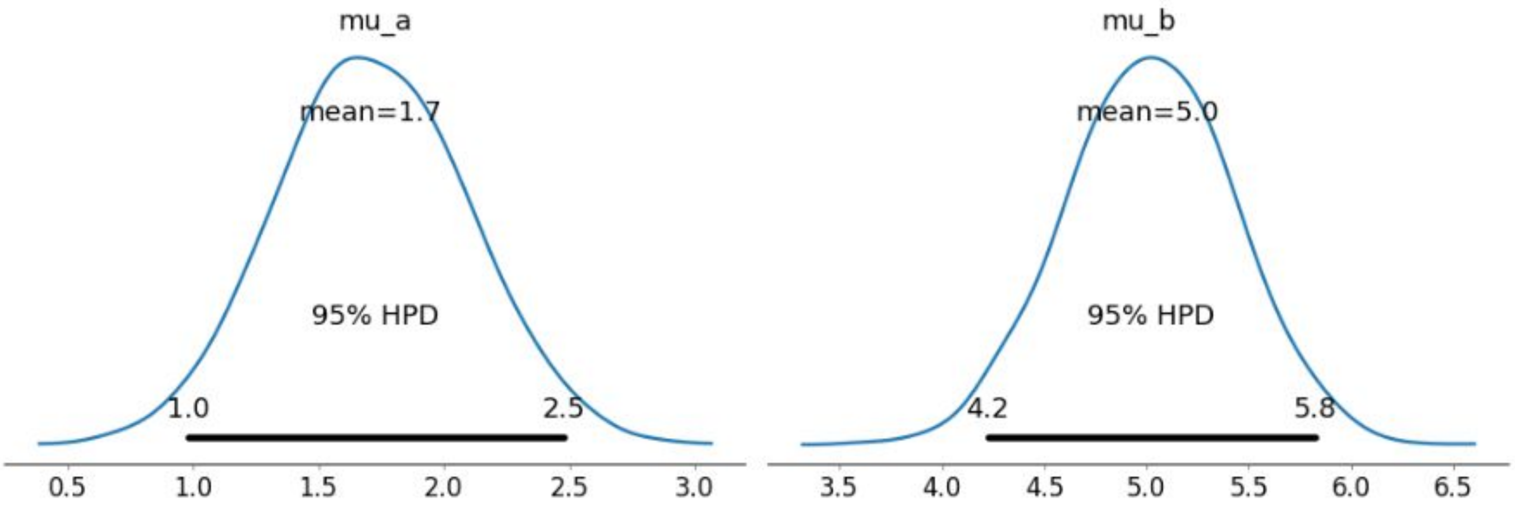
Bayesian Approach: PyMC3 - Example

```
import pymc3 as pm

## set number of sample
## set t = time, from 0 to length of observations
samples = 5000 ## number of iteration
t = np.arange(0, len(z)) ## array of observation positions (time)

with pm.Model() as model:
    ## define uniform priors for the mean values
     mu_a = pm.Uniform('mu_a', 0, 10)
     mu_b = pm.Uniform('mu_b', 0, 10)
     sigma = pm.HalfCauchy('sigma', np.std(z))
     tau = pm.DiscreteUniform('tau', t.min(), t.max())
    ## define stochastic variable mu
     mu = pm.math.switch(tau >= t, mu_a, mu_b)
     observation = pm.Normal('observation', mu, sigma, observed = z)

     trace = pm.sample(samples, step = pm.NUTS())
     burned_trace = trace[1000:]
```

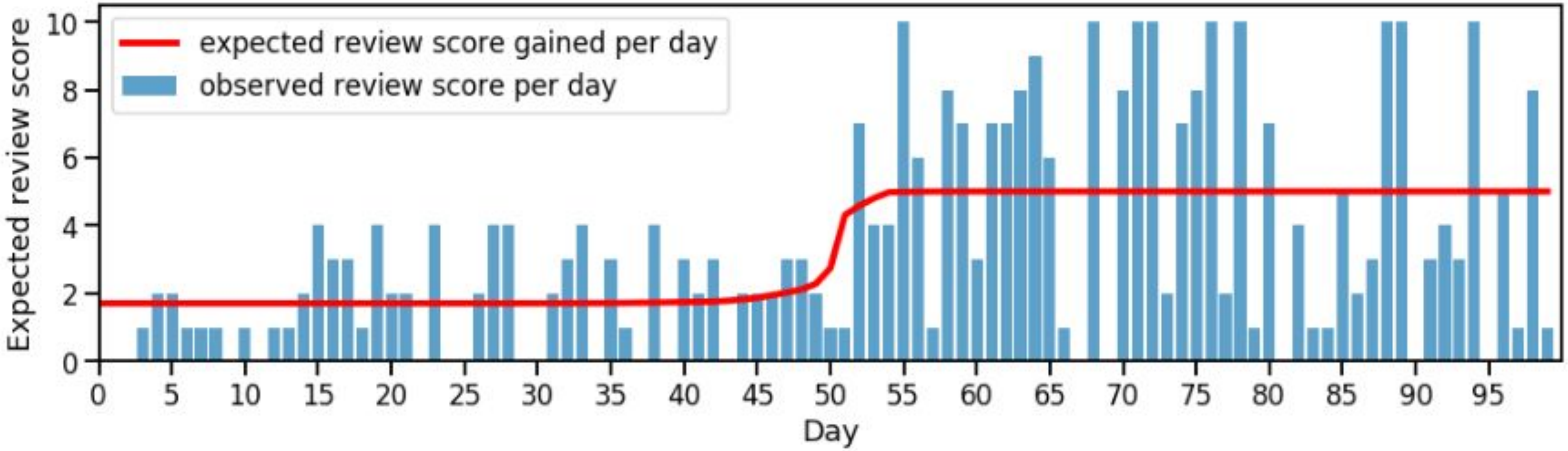
Bayesian Approach: PyMC3 - Changepoint distribution



Bayesian Approach: PyMC3 - Mean distribution



Bayesian Approach: Estimate the change point



3. Conclusions

What have we learned?

Thanks!

Materials: https://github.com/elvyna/pycon-id-2019
Find me on Twitter: @vexenta

Want to learn more?

Killick, R. (2017). Introduction to optimal changepoint detection algorithms. useR! Tutorial 2017

Kass-Hout, T. (2010). Change point analysis. Slideshare.

Bellei, C. (2016). Changepoint Detection. Part I - A Frequentist Approach. [Blog]

Bellei, C. (2017). Changepoint Detection. Part II - A Bayesian Approach. [Blog]

Davidson-Pilon, C. (2015). Chapter 1 - Introduction - PyMC3. Probabilistic Programming and Bayesian Methods for Hackers.