EAS509 Homework 6 (100 points). Key

Submit your answers as a single pdf attach all R code. Failure to do so will result in grade reduction.

Question 1 (100 points)

High-Performance Computing (HPC) resources (a.k.a. supercomputers) are complex systems. Slight changes in hardware or software can drastically affect their performance. For example, a corrupted lookup table in a network switch, an update of a linux kernel, a drop of hardware support in a new software version, and so on.

One way to ensure the top performance of HPC resources is to utilize continuous performance monitoring where the same application is executed with the same input on a regular basis (for example, daily). In a perfect world, the execution time will be exactly the same, but in reality, it varies due to system jitter (this is partially due to system processes taking resources to do their jobs).

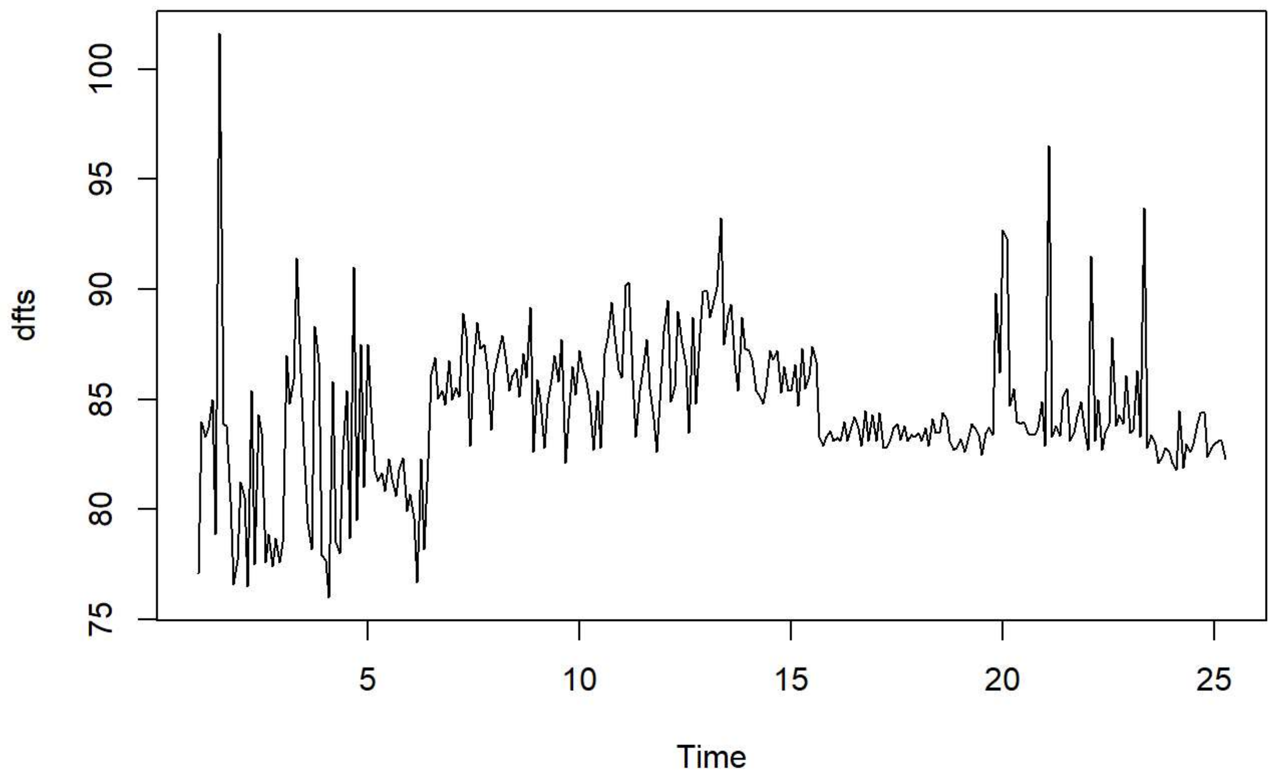
So normally, the execution time will be distributed around a certain value. If performance degradation occurs, the execution time will be distributed around different value.

An automated system that inform system administrators on performance change can be a very handy tool.

In this exercise, your task will be to identify the number and location of the change point where performance was changed. NWChem, an Quantum Chemistry application, was used to probe the performance of UB HPC cluster.

1.1 `UBHPC_8cores_NWChem_Wall_Clock_Time.csv` file contains execution time (same as run time or wall time) of NWChem performing same reference calculation. Read the file and plot it run time on date. (10 points)

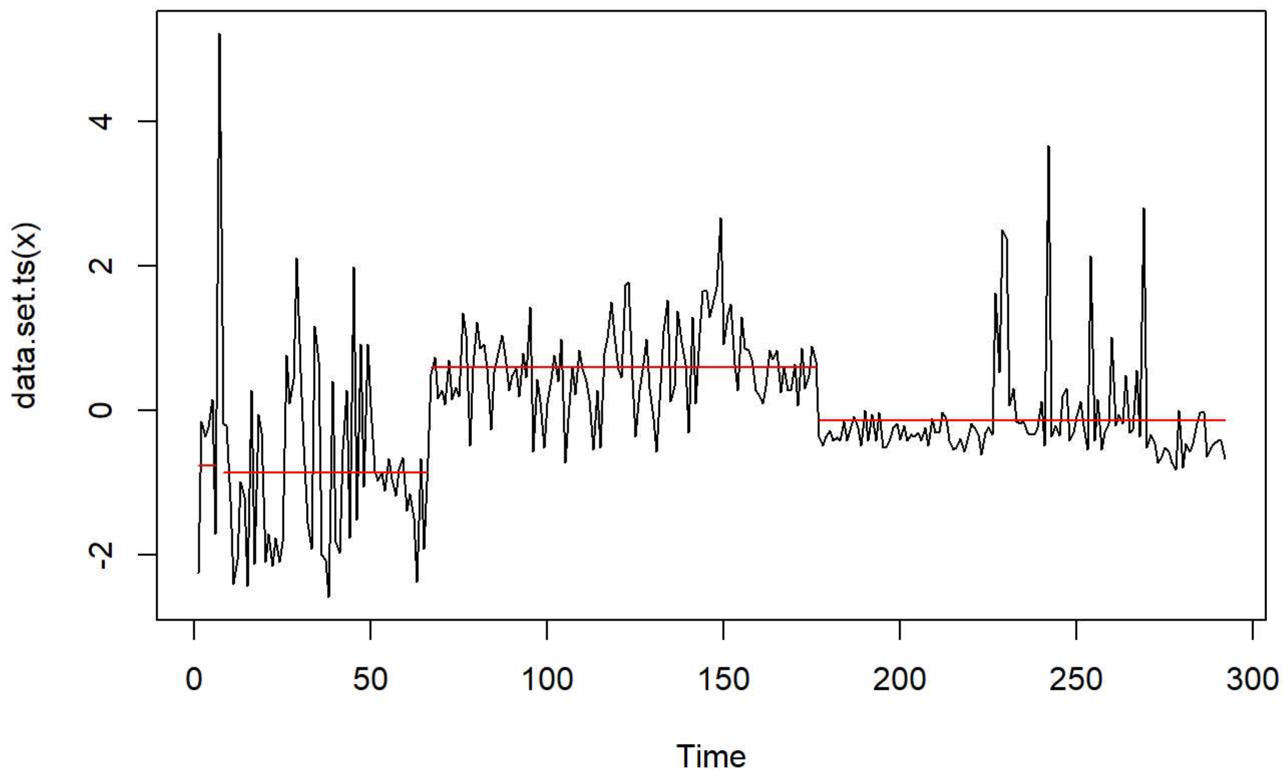
```
df <- read.csv('UBHPC_8cores_NWChem_Wall_Clock_Time.csv')
df$date <- as.POSIXct(df$date, format = "%d/%m/%Y %H:%M")
df$date <- as.Date(df$date)

dfts = ts(df$run_time, frequency = 12)
ts.plot(dfts)
```



```
#mean changepoint analysis
mvalue = cpt.mean(as.vector(scale(dfts)), method='PELT')
cpts(mvalue)
```

```
## [1]   6   7  66 176
```

```
plot(mvalue)
```



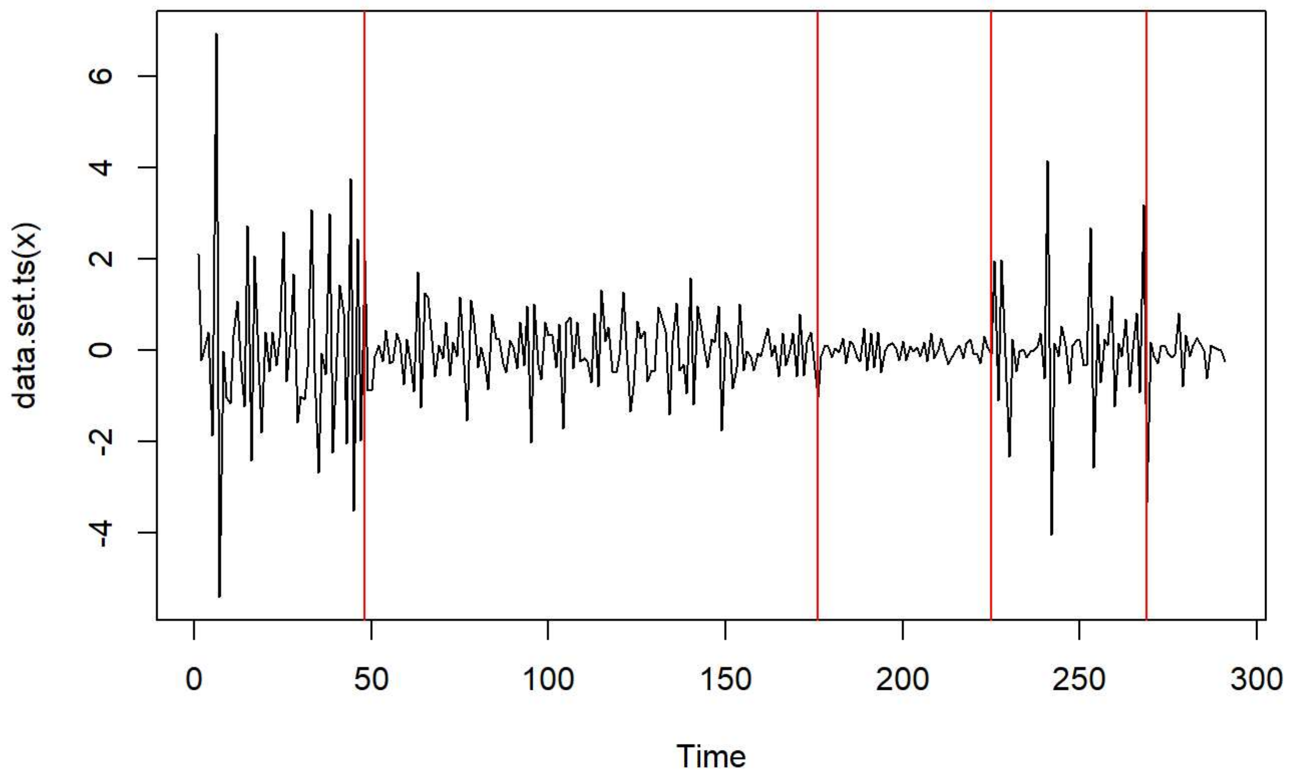
```
summary(mvalue)
```

```
## Created Using changepoint version 2.2.4
## Changepoint type      : Change in mean
## Method of analysis    : PELT
## Test Statistic  : Normal
## Type of penalty       : MBIC with value, 17.03026
## Minimum Segment Length : 1
## Maximum no. of cpts   : Inf
## Changepoint Locations : 6 7 66 176
```

```
# variance changepoint analysis
vnvalue = cpt.var(diff(as.vector(scale(dfts))), method='PELT')
cpts(vnvalue)
```

```
## [1]  48 176 225 269
```

```
plot(vnvalue)
```



```
summary(vnvalue)
```

```
## Created Using changepoint version 2.2.4
## Changepoint type      : Change in variance
## Method of analysis    : PELT
## Test Statistic  : Normal
## Type of penalty       : MBIC with value, 17.01997
## Minimum Segment Length : 2
## Maximum no. of cpts   : Inf
## Changepoint Locations : 48 176 225 269
```

1.2 How many segments/change points can you eyeball? What are they? (10 points)

Usinf cpt mean: 5 changepoints 6, 7, 66, 176 Usinf cpt var: 2 changepoints 48, 176, 225, 269

1.3 Create another column `seg` and assign segment number to it based on previous question. (10 points)

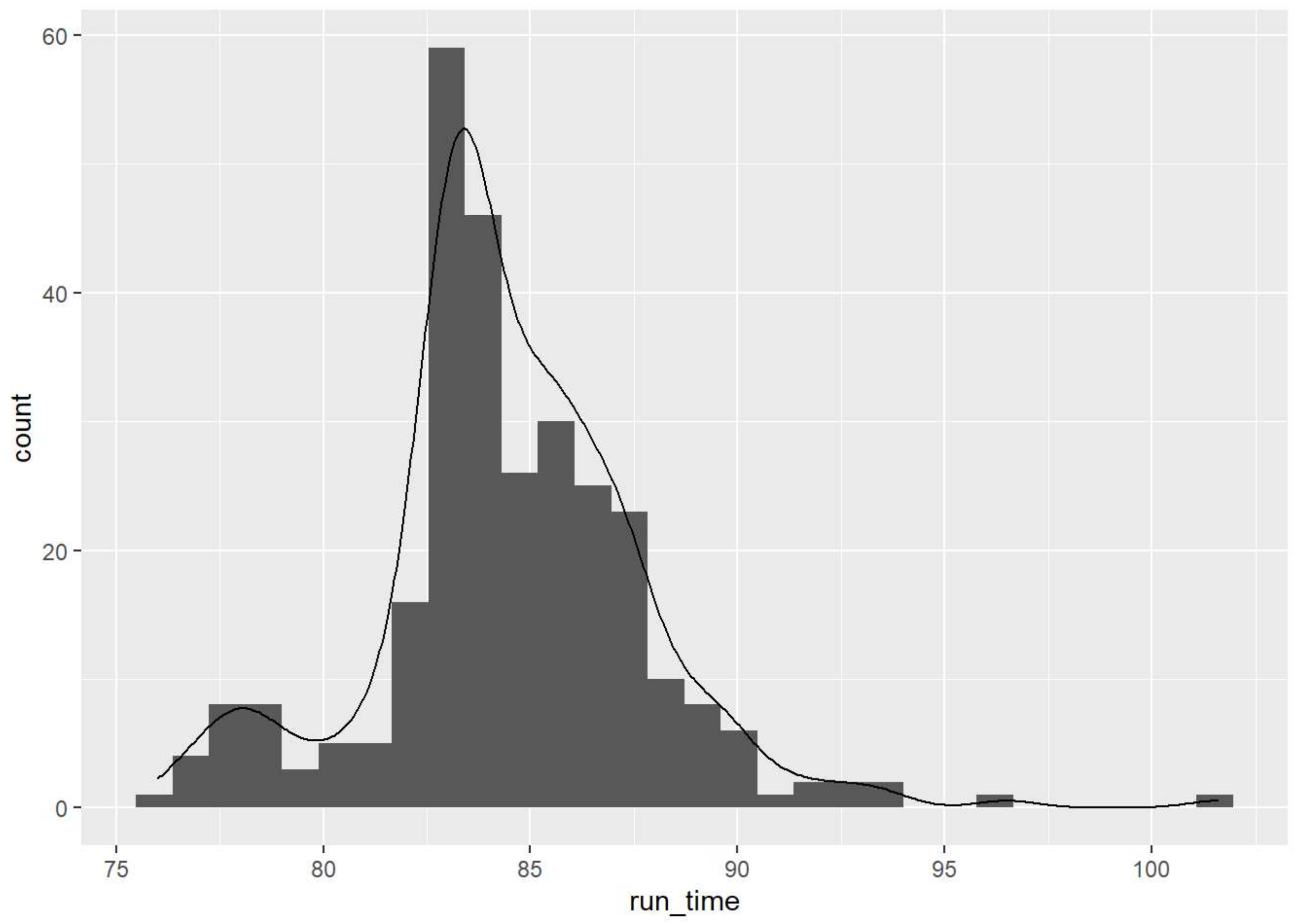
```
seg_data = df %>% mutate(segement = row_number()) %>% mutate(seg = ifelse(row_number() < 6, 1,
                                                                        ifelse(row_number() %in%
6:7, 2,
                                                                            ifelse(row_number()
%in% 7:66, 3,
                                                                                ifelse(row_numbe
r() %in% 66:176, 4,
                                                                                       5)))))

head(seg_data)
```

```
##         date run_time segement seg
## 1 2017-09-11     77.1        1   1
## 2 2017-10-11     84.0        2   1
## 3 2017-11-11     83.3        3   1
## 4 2017-12-11     83.7        4   1
## 5       <NA>     85.0        5   1
## 6       <NA>     78.9        6   2
```

1.4 Make a histagramm plot of all run times. (10 points)

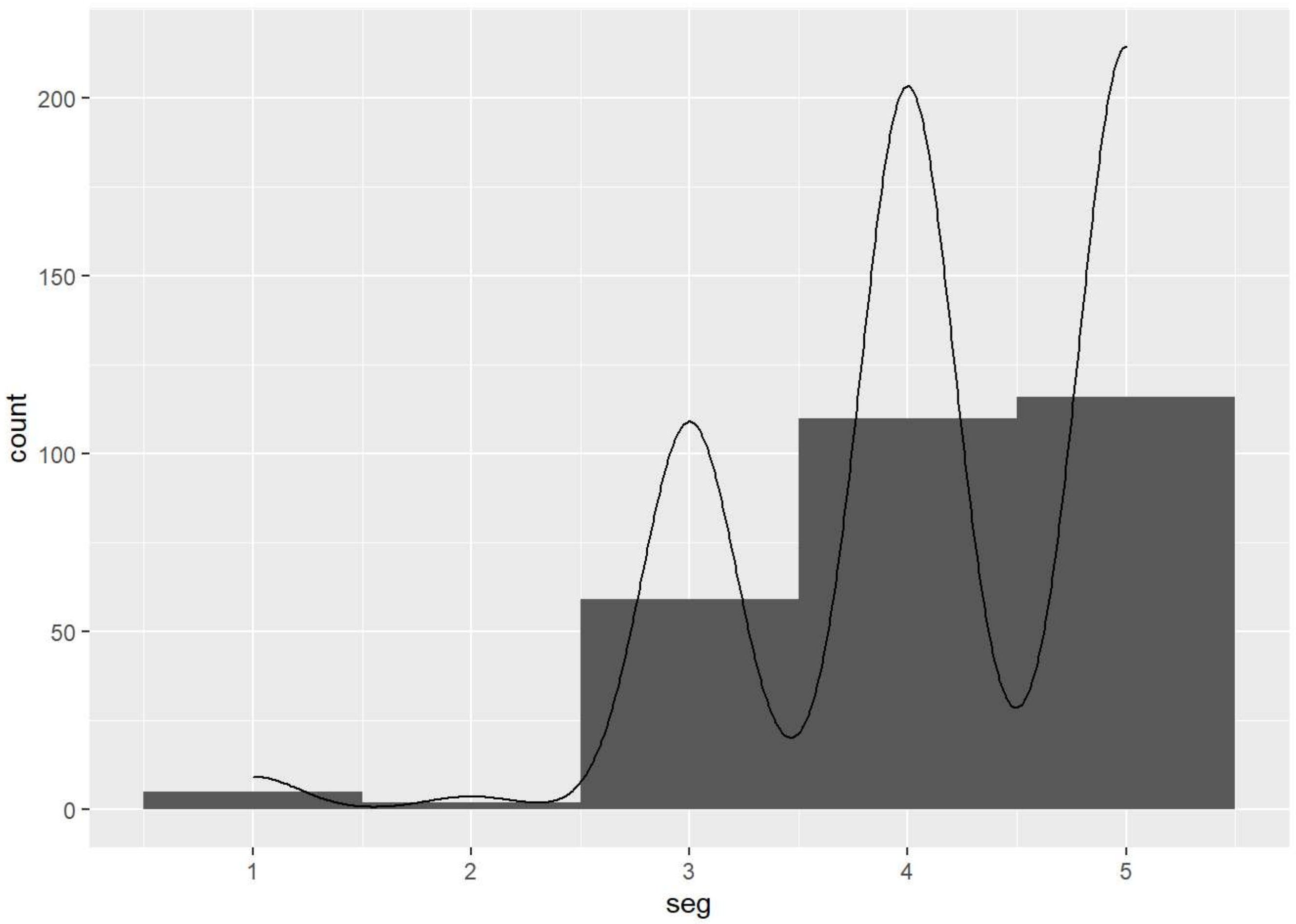
```
ggplot(seg_data, aes(x =run_time)) + geom_histogram()+ geom_density(aes(y=..count..))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



1.5 Make a histogram plot of for each segments. (10 points)

```
ggplot(seg_data, aes(x=seg))+geom_histogram(binwidth=1)+geom_density(aes(y=..count..))
```



1.6

Does it look reasonably normal? (10 points)

From figure 1.4, we can tell that it is a right skewed but a reasonably normal curve.

1.7 Identify change points with `cpt.meanvar` function. Use `PELT` method and `Normal` for `test.stat`. Plot your data with identified segments mean. (10 points)

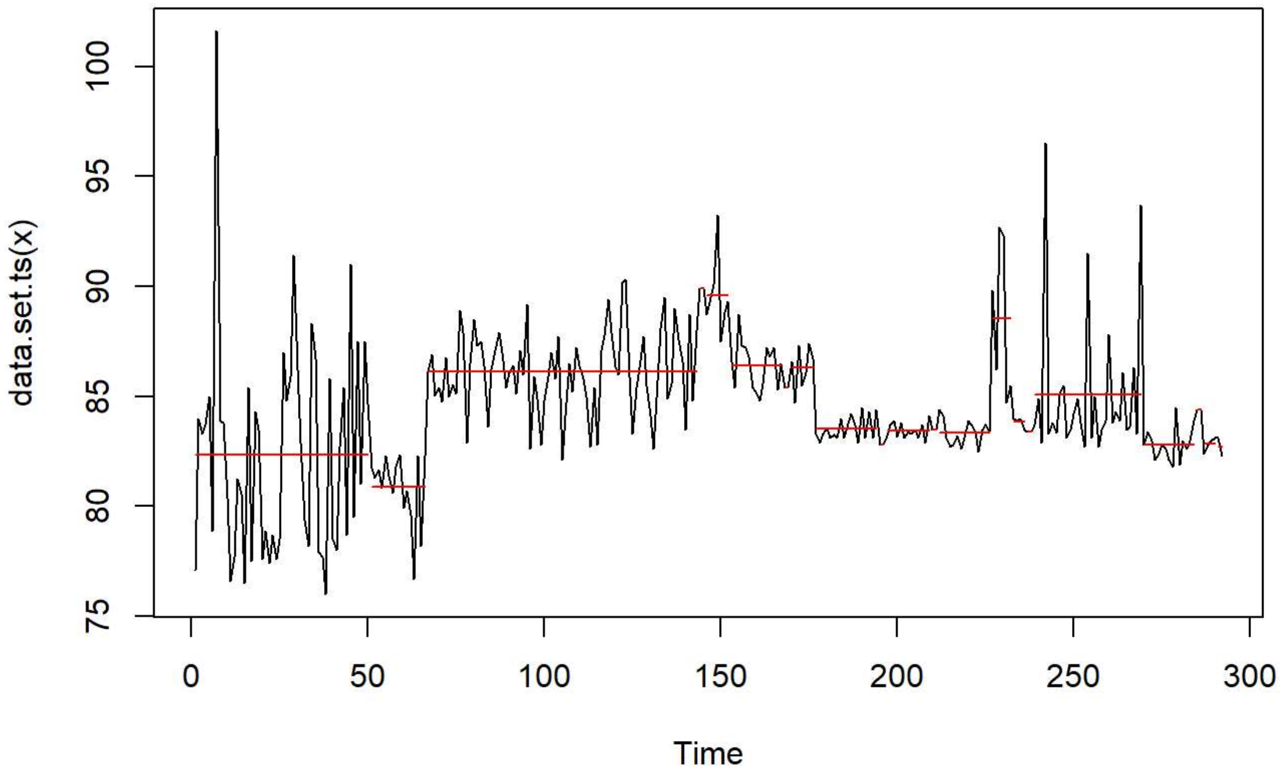
```
set.seed(49)
df_meanvar = cpt.meanvar(df$run_time, test.stat='Normal', method='PELT',penalty = 'SIC')

cpts(df_meanvar)
```

```
##  [1]  50  66 143 145 152 167 169 176 194 196 209 211 226 232 236 238 269 284 286
## [20] 290
```

```
pen.value(df_meanvar)
```

```
## [1] 17.03026
```

```
plot(df_meanvar)
```



hints: run `cpt.meanvar` on the `run_time` column (i.e. `df$run_time`)

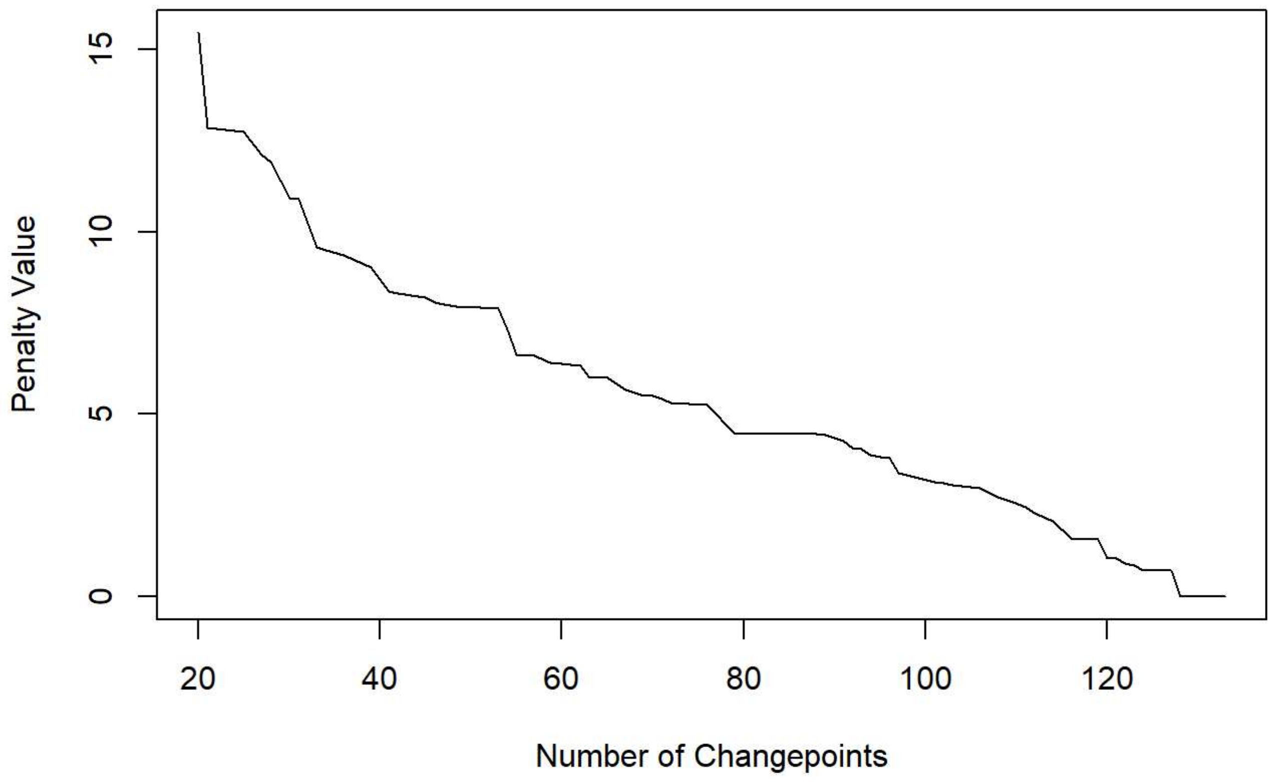
use `pen.value` funtion to see current value of penalty (MBIC value), use that value as guide for your penalty range in next question.

1.8 Using CROPS procedure find optimal number of points. Plot data with optimal number of segments. (10 points)

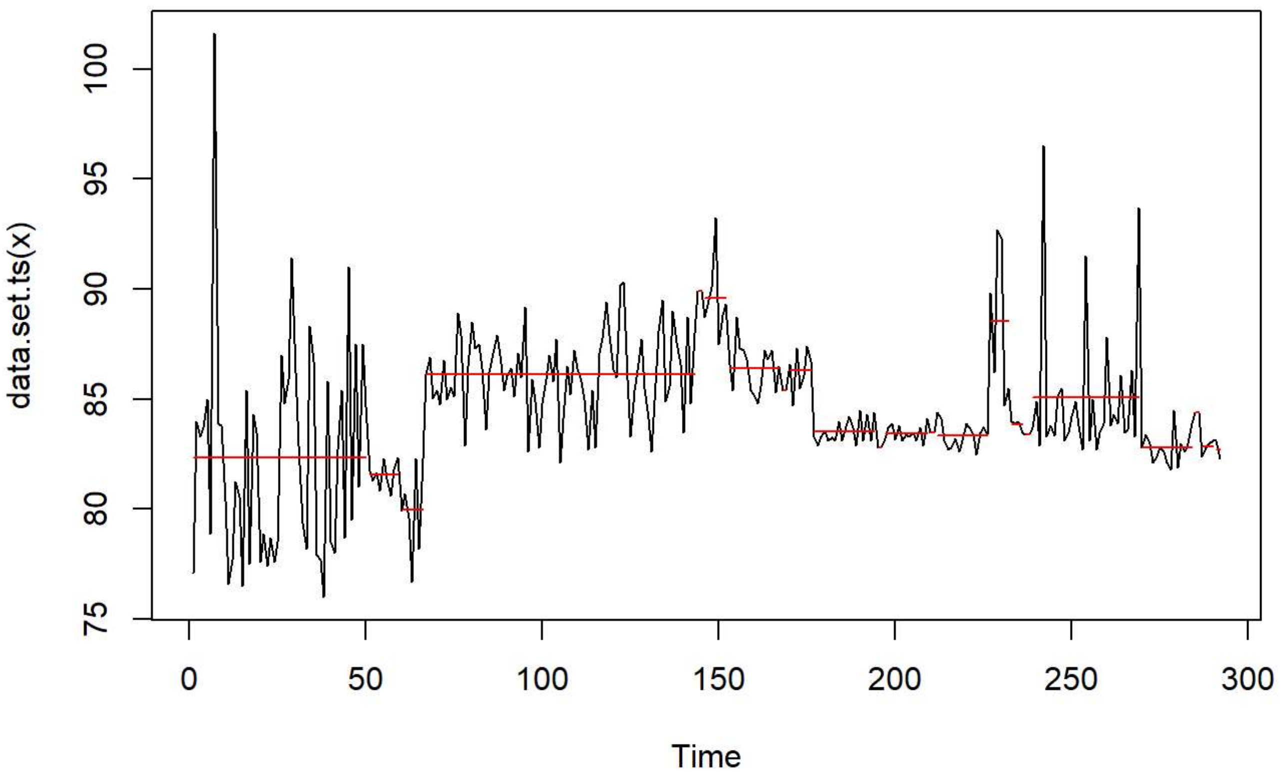
```
df_meanvar_crops = cpt.meanvar(df$run_time, method="PELT", penalty="CROPS",
                 pen.value=c(0, pen.value(df_meanvar)))
```

```
## [1] "Maximum number of runs of algorithm = 115"
## [1] "Completed runs = 2"
## [1] "Completed runs = 3"
## [1] "Completed runs = 5"
## [1] "Completed runs = 9"
## [1] "Completed runs = 17"
## [1] "Completed runs = 33"
## [1] "Completed runs = 59"
## [1] "Completed runs = 85"
## [1] "Completed runs = 93"
## [1] "Completed runs = 95"
```

```
plot(df_meanvar_crops, diagnostic=TRUE)
```



```
plot(df_meanvar_crops,ncpts=21)
```



1.9 Does your initial segment guess matches with optimized by CROPS? (10 points)

Our guess was 5 with mean and 20 using mean var. Our guess matches with the one optimized with CROPS

1.10 The run-time in this example does not really follow normal distribution. What to do you think can we still use this method to identify changepoints? (10 points)

PS. Just in case if you wounder. On 2018-02-21 system got a critical linux kernel update to alleviate Meltdown-Spectre vulnerabilities. On 2018-06-28 system got another kernel update which is more robust and hit the performance less

Mean, var and meanvar assumes that the data is normalized. We need to normalize the data by differencing it such that the mean fits over x axis(mean =0) and the variance is 1.