EAS509 Homework 5 (40 points). Key

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

Question 1 (40 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. (4 points)

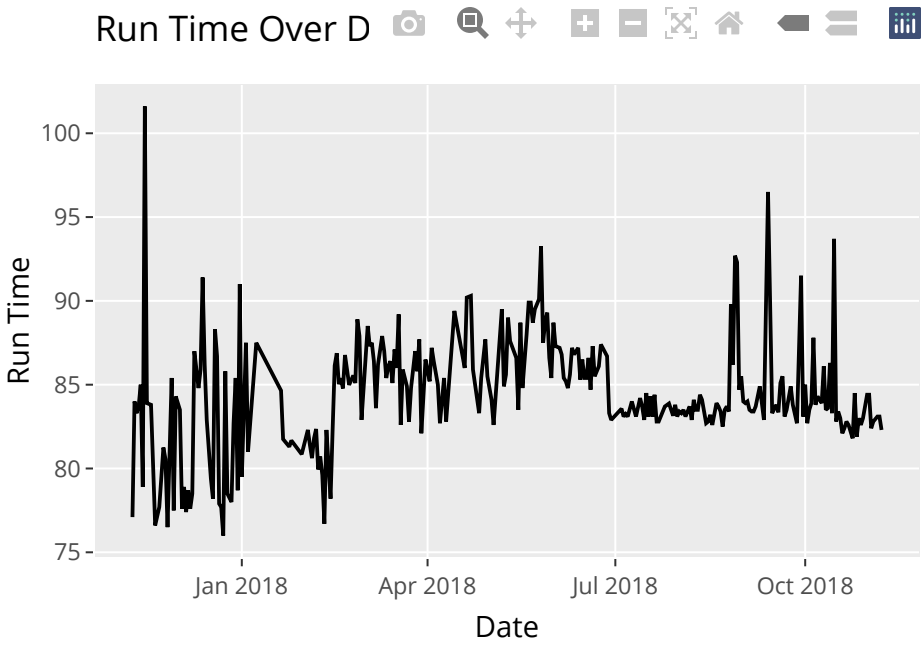
```
data <- read_csv("UBHPC_8cores_NWChem_Wall_Clock_Time.csv")
```

```
## Rows: 292 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): date
## dbl (1): run_time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data <- data %>%
  mutate(date = as.POSIXct(date, format="%m/%d/%Y %H:%M", tz="UTC"))
```

```
# Your existing ggplot code
p <- ggplot(data, aes(x = date, y = run_time)) +
  geom_line() +
  labs(title = "Run Time Over Date", x = "Date", y = "Run Time")

# Convert it to a plotly interactive plot
ggplotly(p)
```



1.2 How many segments/change points can you eyeball? What are they? (4 points)

I can 5 segments and 4 change points in timeseries plot.

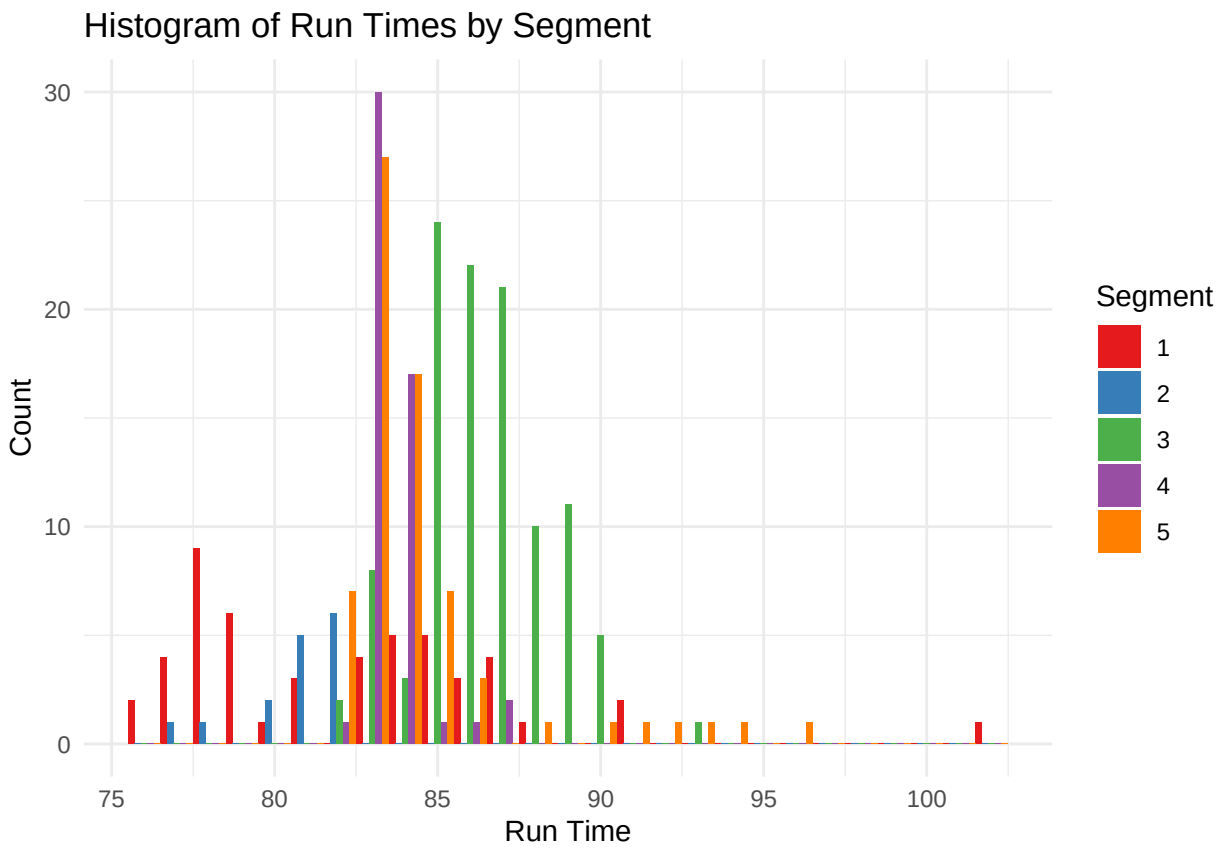
- 21-01-2018 there is change in variance
- 14-02-2018 there is a change in mean.
- 21-06-2018 there is a change in both mean and variance
- 24-08-2018 there is a change in variance

1.3 Create another column `seg` and assign segment number to it based on previous question. (4 points)

```
data <- data %>%
  mutate(seg = case_when(
    date < as.Date('2018-01-21') ~ 1,
    date >= as.Date('2018-01-21') & date < as.Date('2018-02-14') ~ 2,
    date >= as.Date('2018-02-14') & date < as.Date('2018-06-21') ~ 3,
    date >= as.Date('2018-06-21') & date < as.Date('2018-08-24') ~ 4,
    date >= as.Date('2018-08-24') ~ 5
  ))
```

1.4 Make a histagramm plot of all run times. (4 points)

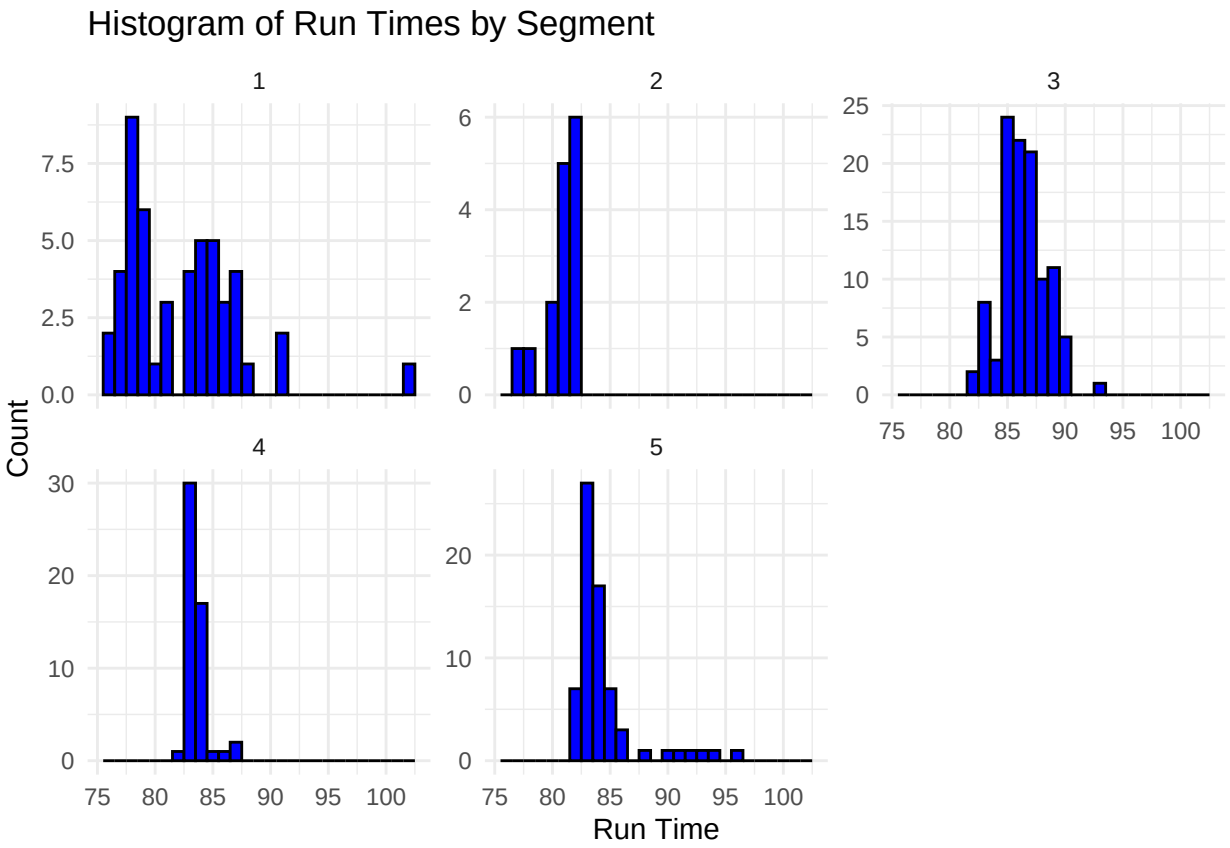
```
ggplot(data, aes(x = run_time, fill = factor(seg))) +  # Use 'fill' to distinguish segments
  geom_histogram(position = "dodge", binwidth = 1) +  # Adjust 'binwidth' as needed for your data
  scale_fill_brewer(palette = "Set1", name = "Segment") +
  labs(title = "Histogram of Run Times by Segment", x = "Run Time", y = "Count") +
  theme_minimal()
```



1.5 Make a histogram plot of for each segments. (4 points)

```r
# Create the histogram plot with faceting
p <- ggplot(data, aes(x = run_time)) +
  geom_histogram(binwidth = 1, fill = 'blue', color = 'black') +  # Adjust binwidth as needed
  facet_wrap(~seg, scales = 'free_y') +  # Facet by segment, allowing different y scales
  labs(title = "Histogram of Run Times by Segment", x = "Run Time", y = "Count") +
  theme_minimal()

# Print the plot
print(p)
```



1.6 Does it look reasonably normal? (4 points)

- For segment 1 and 2 the data is not normal at all.
- For segment 3 and 4 the is reasonably normal
- For segment 5 is normal but skewed towards right.

In conclusion, while some segments may resemble a normal distribution more closely than others, most segments do not appear to be normally distributed based on the shapes of their histograms.

1.7 Identify change points with `cpt.meanvar` function. Use `PELT` method and `Normal` for `test.stat`. Plot your data with identified segments mean. (4 points)

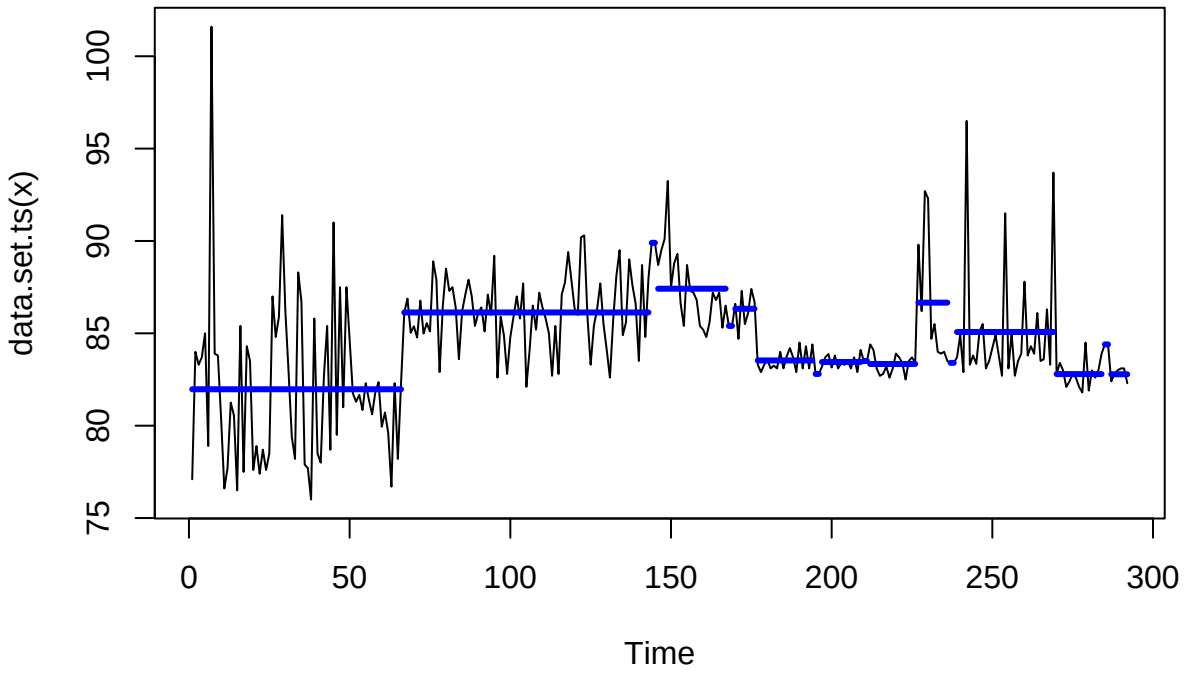
> hints: run `cpt.meanvar` on the `run_time` column (i.e. `df$run_time`)
>
> use `pen.value` funtion to see current value of penalty (MBIC value), use that value as guide for your penalty range in next question.

```
cpt <- cpt.meanvar(data$run_time, method = "PELT", test.stat = "Normal")

penalty_value <- pen.value(cpt)
penalty_value
```

```
## [1] 22.70702
```

```
plot(cpt,cpt.width=3,cpt.col='blue')
```

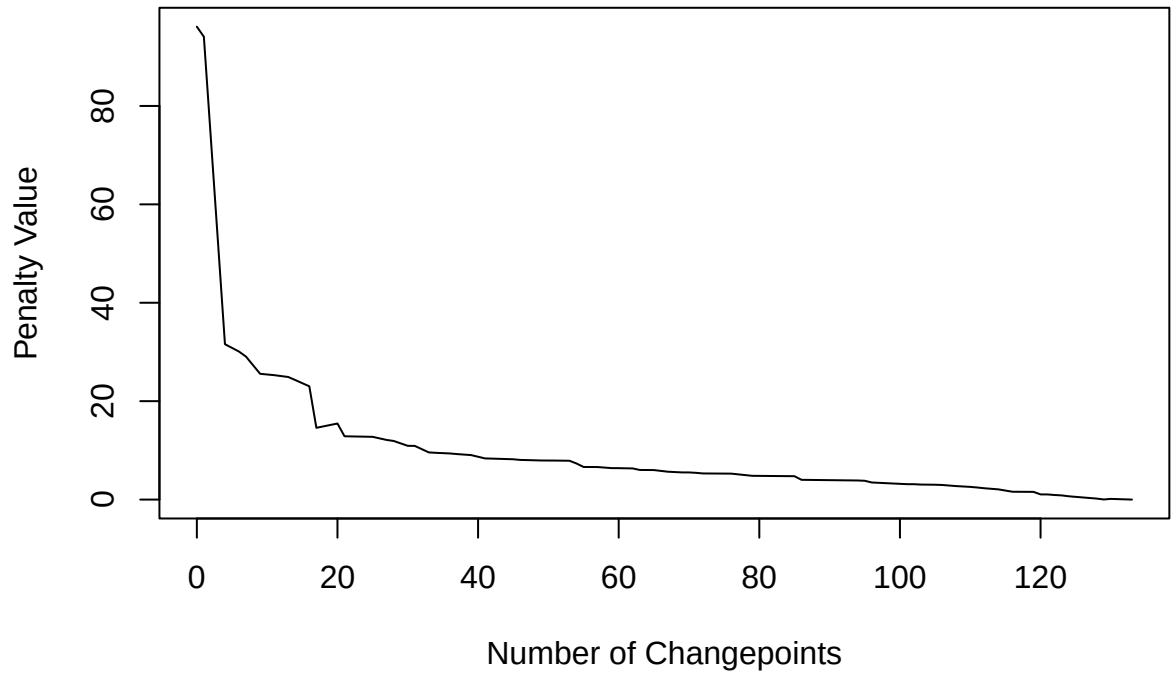


1.8 Using CROPS procedure find optimal number of points. Plot data with optimal number of segments. (4 points)

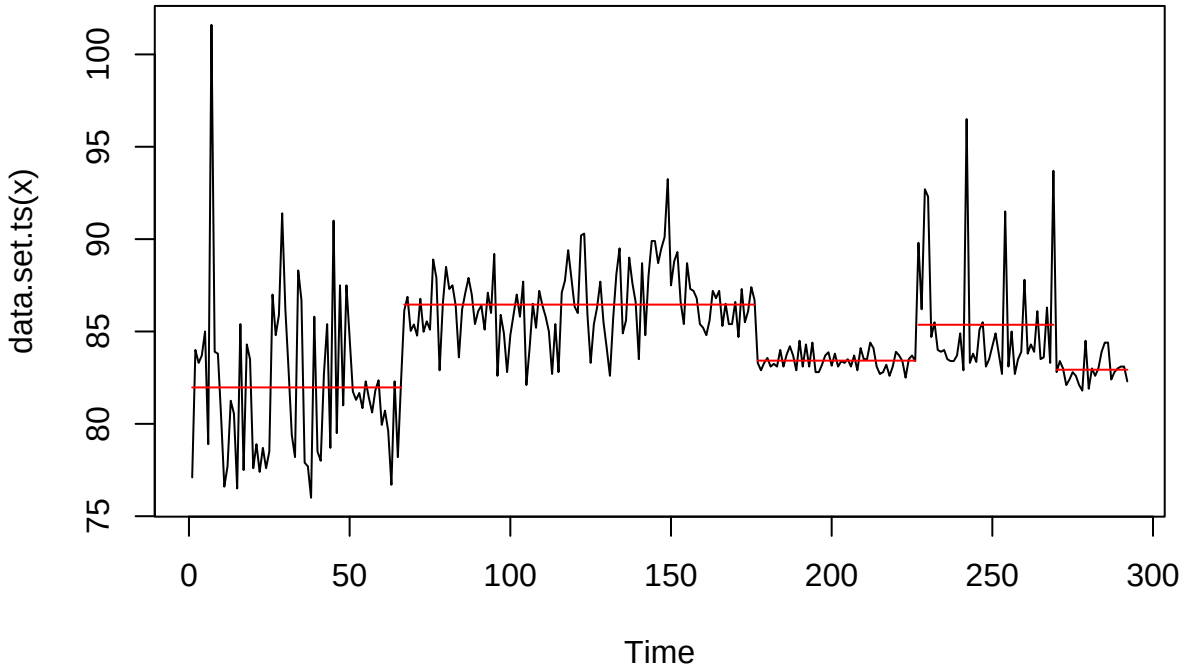
```
penalty_range <- c(0, 10 * penalty_value)
cpt_crops <- cpt.meanvar(data$run_time, method="PELT", penalty="CROPS",
                         test.stat="Normal", pen.value=penalty_range)
```

```
## [1] "Maximum number of runs of algorithm = 135"
## [1] "Completed runs = 2"
## [1] "Completed runs = 3"
## [1] "Completed runs = 5"
## [1] "Completed runs = 9"
## [1] "Completed runs = 17"
## [1] "Completed runs = 31"
## [1] "Completed runs = 54"
## [1] "Completed runs = 87"
## [1] "Completed runs = 106"
## [1] "Completed runs = 108"
## [1] "Completed runs = 110"
```

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




```
plot(cpt_crops,ncpts=4)
```



1.9 Does your initial segment guess matches with optimized by CROPS? (4 points) My initial guess in number of clusters in almost matched. Only varied by slight change in the position of change points.

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? (4 points)

Changepoint analysis, such as the `cpt.meanvar` function with the `Normal` test statistic, primarily requires that data within each segment are independent and identically distributed. The method can still effectively detect changes in mean or variance, even if the overall data do not adhere to a normal distribution, we can use other distributions that suits our needs.

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