# *Однофакторний дисперсійний аналіз*

**Мета роботи**: ознайомитись з методикою проведення однофакторного дисперсійного аналізу; навчитись використовувати дисперсійний аналіз для виявлення факторів, що суттєво впливають на тривалість роботи програми.

**Основне завдання**

1. Створити та завантажити вхідні дані.
2. Виконати однофакторний дисперсійний аналіз.
3. Зробити висновок про суттєвість впливу досліджуваного фактора.

**Код програми та отримані результати:**

| № | Метод сортування | Фактор |
|---|---|---|
| 24 | Пірамідальне сортування | Кількість однакових значень |

Функції, що здійснюють пірамідальне сортування:

```
heap.building = function(vec)
{
  len=length(vec)
  heap=vec
  for (j in len:1)
  {
    heap=modify.heap(heap,j)
  }
  return(heap)
}
is.heap = function(heap,root_i)
{
  i=root_i
  res=T
  while(2*i<=length(heap)&res)
  {
    son=c(heap[2*i],heap[2*i+1])
    son=son[!is.na(son)]
    res=all(heap[i]<=son)
    i=i+1
  }
  return(res)
}
modify.heap = function(heap,root_i)
{
  len=length(heap)
  flag=1

  while (root_i*2<=len&&flag==1)
  {
    left_i=root_i*2
    right_i=root_i*2+1
    flag=0
    son=c(heap[left_i],heap[right_i])
    son=son[!is.na(son)]
    min_ind=which.min(son)
    if (heap[root_i]>son[min_ind])
    {
      flag=1
      heap_ind=c(left_i,right_i)[min_ind]

      tmp=heap[heap_ind]
      heap[heap_ind]=heap[root_i]
```

```
      heap[root_i]=tmp

      root_i=heap_ind
    }
  }
  return(heap)
}

heap.sort = function(heap)
{
  sorted=NULL
  len=length(heap)
  while(len>0)
  {
    sorted=c(sorted,heap[1])
    len=length(heap)
    heap[1]=heap[len]
    heap=heap[1:(len-1)]
    heap=modify.heap(heap,root_i=1)
    len=len-1
  }
  return(sorted)
}
```

Функція, що підраховує час виконання сортування кожної вибірки 50 разів:

```
# time  for heapsort
time = function(x){
  destination = c()
  for(i in c(1:50)){
    n = as.numeric(system.time(heap.sort(heap.building(x)))[1])
    destination = c(destination, n)
  }
  mean_value = mean(destination)
  times = destination
  destination
}
```

Функції для підрахунку частки унікальних елементів у вибірці:

```
##percent function
percent = function(sampl){
  length.sampl = length(sampl)
  unique.sampl = length(unique(sampl))
  percent = 100-floor(100*unique.sampl/length.sampl)
  percent
}

#percents with "%"
percent_strings = function(per.s)
{
  percent.array = sprintf("%i%s",per.s, "%" )
  repeats = c()

  for (i in 1:6)
  {
    repeats = c(repeats,rep(percent.array[i],50))
  }
  repeats
}
```

Створимо вибірки для подальшого дослідження в залежності від наданого фактора  кількості однакових значень, що має 6 постійних рівнів:

```
# create samples
repeat_level1 = sample(1:1000000, 2000, replace=TRUE )
```

```
repeat_level2 = sample(1:10000, 2000, replace = TRUE)

repeat_level3 = sample(1000:4500, 2000, replace = TRUE)

repeat_level4 = sample(1000:2000, 2000, replace = TRUE)

repeat_level5 = sample(1000:1500, 2000, replace = TRUE)

repeat_level6 = sample(1000:1100, 2000, replace = TRUE)
```

Визначаємо час сортування для 6 вибірок по 50 разів (6*50=300 експериментів):

```
one = time(repeat_level1)
two = time(repeat_level2)
three = time(repeat_level3)
four = time(repeat_level4)
five = time(repeat_level5)
six = time(repeat_level6)

times.array = c(one,two,three,four,five,six)

#time array
times = vector(mode = "list", length = 6)
times[[1]] = one
times[[2]] = two
times[[3]] = three
times[[4]] = four
times[[5]] = five
times[[6]] = six

#all times
times
```

```
> #all times
> times
[[1]]
 [1] 0.059 0.059 0.056 0.065 0.056 0.065 0.065 0.060 0.064 0.057 0.056 0.061 0.056 0.060 0.056 0.060
[17] 0.062 0.061 0.055 0.059 0.056 0.058 0.058 0.060 0.056 0.060 0.056 0.059 0.055 0.057 0.056 0.058
[33] 0.057 0.057 0.056 0.058 0.057 0.060 0.059 0.059 0.062 0.059 0.062 0.058 0.058 0.060 0.056 0.060
[49] 0.057 0.058

[[2]]
 [1] 0.059 0.058 0.057 0.058 0.061 0.063 0.061 0.059 0.056 0.061 0.057 0.057 0.056 0.058 0.060 0.062
[17] 0.057 0.058 0.058 0.059 0.057 0.056 0.056 0.058 0.056 0.058 0.060 0.058 0.057 0.058 0.057 0.058
[33] 0.056 0.058 0.055 0.059 0.057 0.058 0.058 0.059 0.056 0.058 0.055 0.059 0.059 0.059 0.057 0.058
[49] 0.055 0.060

[[3]]
 [1] 0.056 0.059 0.056 0.059 0.062 0.063 0.060 0.061 0.056 0.058 0.063 0.062 0.057 0.058 0.056 0.059
[17] 0.056 0.057 0.057 0.057 0.056 0.058 0.057 0.059 0.056 0.058 0.058 0.057 0.057 0.058 0.056 0.060
[33] 0.058 0.060 0.057 0.057 0.056 0.058 0.056 0.059 0.057 0.058 0.055 0.057 0.058 0.057 0.057 0.058
[49] 0.055 0.058

[[4]]
 [1] 0.056 0.057 0.057 0.058 0.060 0.063 0.063 0.063 0.058 0.060 0.063 0.058 0.059 0.057 0.056 0.057
[17] 0.057 0.058 0.057 0.060 0.059 0.059 0.058 0.058 0.057 0.060 0.059 0.058 0.057 0.061 0.060 0.058
[33] 0.056 0.062 0.056 0.058 0.056 0.058 0.056 0.059 0.061 0.057 0.057 0.061 0.055 0.058 0.056 0.059
[49] 0.057 0.058

[[5]]
 [1] 0.058 0.058 0.057 0.058 0.060 0.062 0.059 0.058 0.057 0.057 0.055 0.058 0.056 0.058 0.059 0.060
[17] 0.057 0.064 0.057 0.057 0.058 0.057 0.056 0.059 0.057 0.057 0.056 0.059 0.059 0.065 0.061 0.057
[33] 0.055 0.056 0.057 0.058 0.057 0.057 0.055 0.058 0.055 0.057 0.058 0.059 0.057 0.058 0.058 0.060
[49] 0.057 0.058

[[6]]
 [1] 0.055 0.056 0.056 0.058 0.059 0.063 0.061 0.061 0.057 0.058 0.056 0.057 0.058 0.058 0.056 0.058
[17] 0.057 0.059 0.055 0.059 0.056 0.059 0.057 0.058 0.056 0.059 0.059 0.058 0.064 0.058 0.057 0.058
[33] 0.057 0.058 0.057 0.058 0.055 0.058 0.056 0.058 0.057 0.058 0.059 0.060 0.056 0.058 0.057 0.060
[49] 0.058 0.060
```

Подамо дані у вигляді таблиці:

```
#create data
```

```
percents = c(percent(repeat_level1),
             percent(repeat_level2),
             percent(repeat_level3),
             percent(repeat_level4),
             percent(repeat_level5),
             percent(repeat_level6))
percent.string = percent_strings(percents)
datas = data.frame(times.array, percent.string)
```

laba6g.R × datas ×

Filter

| | times.array | percent.string |
|---|---|---|
| 29 | 0.055 | 0% |
| 30 | 0.057 | 0% |
| 31 | 0.056 | 0% |
| 32 | 0.058 | 0% |
| 33 | 0.057 | 0% |
| 34 | 0.057 | 0% |
| 35 | 0.056 | 0% |
| 36 | 0.058 | 0% |
| 37 | 0.057 | 0% |
| 38 | 0.060 | 0% |
| 39 | 0.059 | 0% |
| 40 | 0.059 | 0% |
| 41 | 0.062 | 0% |
| 42 | 0.059 | 0% |
| 43 | 0.062 | 0% |
| 44 | 0.058 | 0% |
| 45 | 0.058 | 0% |
| 46 | 0.060 | 0% |
| 47 | 0.056 | 0% |
| 48 | 0.060 | 0% |
| 49 | 0.057 | 0% |
| 50 | 0.058 | 0% |
| 51 | 0.059 | 11% |
| 52 | 0.058 | 11% |
| 53 | 0.057 | 11% |
| 54 | 0.058 | 11% |
| 55 | 0.061 | 11% |
| 56 | 0.063 | 11% |
| 57 | 0.061 | 11% |
| 58 | 0.059 | 11% |

Showing 29 to 58 of 300 entries

```
str(datas)
```

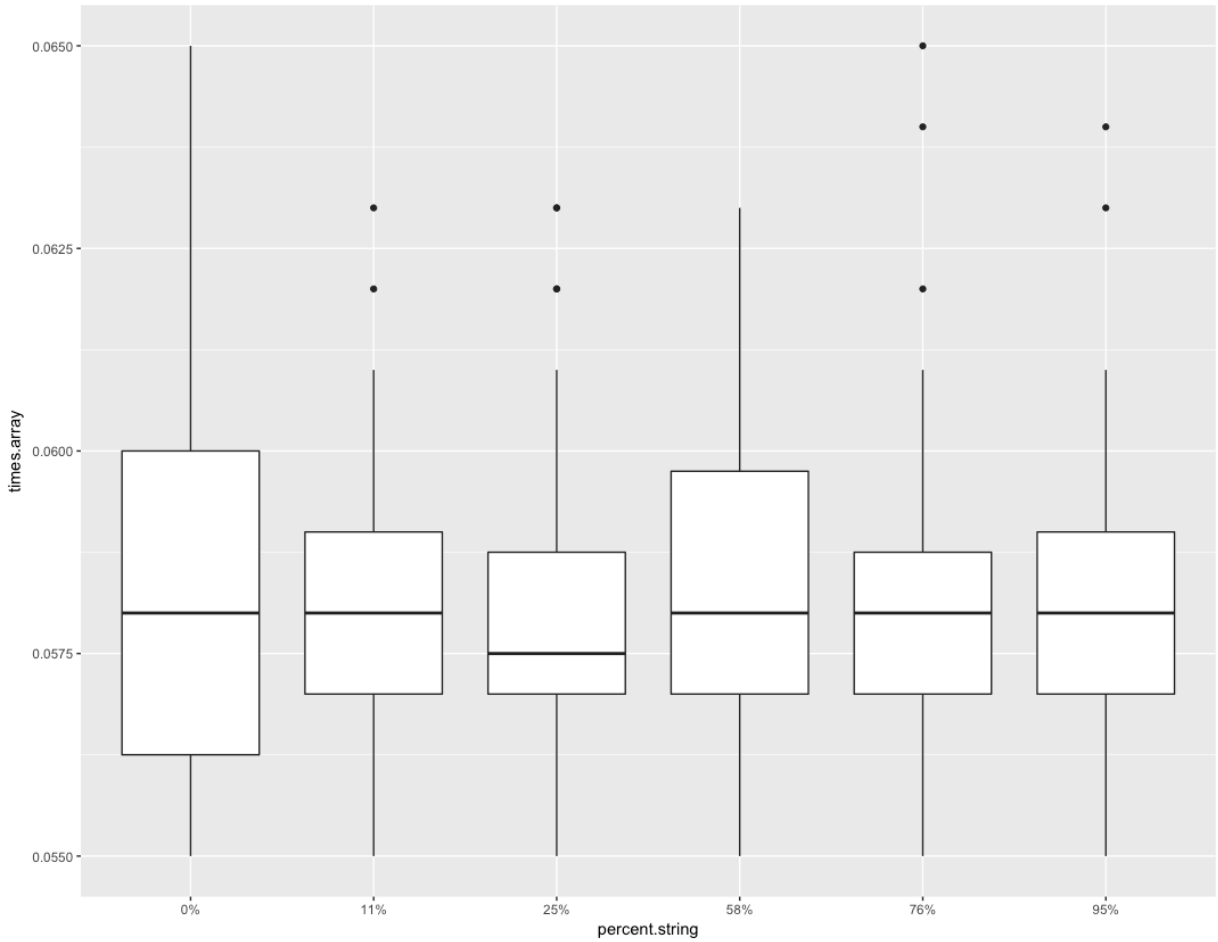
```
> str(datas)
'data.frame':   300 obs. of  2 variables:
 $ times.array   : num  0.059 0.059 0.056 0.065 0.056 ...
 $ percent.string: Factor w/ 6 levels "0%","11%","25%",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Бачимо, що відсоток унікальних елементів у вибірці є фактором, що має 6 постійних рівнів.

Побудуємо «ящик з вусами» для даних 50 експериментів над кожною з вибірок.

```
install.packages("ggplot2")
library(ggplot2)
```

```
ggplot(datas, aes(x = percent.string, y = times.array)) + geom_boxplot()
```




Бачимо, що квантилі розподілу, зокрема медіани, для кожної з вибірок різні. Отже, постає задача дослідити суттєвість впливу такого фактора як кількість однакових елементів у вибірці на час виконання сортування. Для цього використаємо однофакторний дисперсійний аналіз (оскільки досліджується залежність величини часу від лише одного фактора).

```
result = aov(times.array ~ percent.string, data = datas)
summary(result) ## p-value > 0.05 cant break that times are the same
```

```
> summary(result) ## p-value > 0.05 cant break that times are the same
                Df    Sum Sq   Mean Sq F value Pr(>F)
percent.string   5 0.0000282 5.645e-06   1.347  0.245
Residuals      294 0.0012324 4.192e-06
```

Можна побачити, що p-значення = 0.245, тобто є більшим за 5 відсотків, це каже про те, що ми не можемо відхилити гіпотезу про те, що час при різній кількості однакових елементів є однаковим.

*Можемо зробити висновок, що кількість повторюваних елементів не суттєво впливає на час роботи програми.*

Ще можемо подивитися, чи спостерігається суттєва різниці між певною парою факторів для цього скористаємось наступною функцією:

```
# what diference is between every 2 factors
tuk = TukeyHSD(result)
plot(TukeyHSD(result))
```

```
> TukeyHSD(result)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = times.array ~ percent.string, data = datas)

$percent.string
                 diff           lwr          upr     p adj
11%-0%  -6.800000e-04 -0.0018546582 0.0004946582 0.5589996
25%-0%  -8.200000e-04 -0.0019946582 0.0003546582 0.3433510
58%-0%  -2.600000e-04 -0.0014346582 0.0009146582 0.9883064
76%-0%  -7.600000e-04 -0.0019346582 0.0004146582 0.4315896
95%-0%  -7.600000e-04 -0.0019346582 0.0004146582 0.4315896
25%-11% -1.400000e-04 -0.0013146582 0.0010346582 0.9993797
58%-11%  4.200000e-04 -0.0007546582 0.0015946582 0.9090716
76%-11% -8.000000e-05 -0.0012546582 0.0010946582 0.9999605
95%-11% -8.000000e-05 -0.0012546582 0.0010946582 0.9999605
58%-25%  5.600000e-04 -0.0006146582 0.0017346582 0.7464086
76%-25%  6.000000e-05 -0.0011146582 0.0012346582 0.9999905
95%-25%  6.000000e-05 -0.0011146582 0.0012346582 0.9999905
76%-58% -5.000000e-04 -0.0016746582 0.0006746582 0.8263496
95%-58% -5.000000e-04 -0.0016746582 0.0006746582 0.8263496
95%-76% -1.706968e-15 -0.0011746582 0.0011746582 1.0000000
```

Ці дані показують те, що різниця між факторами 25% та 0% є найбільшою порівняно з іншими парами (це можна спостерігати на боксплоті), але несуттєвою. Та найменшу різницю мають фактори 95% та 76%.