Praca domowa nr 2

Franciszek Szczepaniak

2022-05-12

Wstęp

W poniższym raporcie zamieszczony został kod wszystkich funkcji, wynikowe ramki danych oraz sprawdzenie, czy wszystkie wyniki funkcji w konkretnych zadaniach są sobie równe. Dodatkowo został porównany czas w jakim różne funkcje wykonują zadania. Pod każdą z funkcji Sqldf znajduje się słowna interpretacja zapytań, najpierw w szczegółowej formie krok po kroku, a następnie “w skrócie”, czyli co tak właściwie zostaje zwrócone przez funkcję.

Na początek wczytuję wszystkie potrzebne ramki danych oraz biblioteki.

```
Badges <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Badges.csv.gz")
Comments <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Comments.csv.gz")
PostLinks <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/PostLinks.csv.gz")
Posts <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Posts.csv.gz")
Tags <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Tags.csv.gz")
Users <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Users.csv.gz")
Votes <- read.csv("D:/przetwarzanie danych ustrukturyzowanych/projekt2/Votes.csv.gz")
options(stringsAsFactors=FALSE)
library(dplyr)
library(sqldf)
library(data.table)
```

Zadanie 1

Funkcje oraz wynik

```
#sqldf
df_sql_1 <- function(df1) {
   sqldf<-sqldf("SELECT Count, TagName
   FROM Tags
   WHERE Count > 1000
   ORDER BY Count DESC")
   sqldf
}
sqldf_1 <- df_sql_1(Tags)
```

Interpretacja

Z ramki danych „Tags” wybieramy te wartości kolumn TagName i Count, gdzie Count jest większe niż 1000. Następnie wiersze powstałej ramki danych ustawiamy w porządku malejącym według wartości Count.

```
#wynik
sqldf_1
```

```
##    Count                 TagName
## 1   9470                   visas
## 2   5119                     usa
## 3   4601                      uk
## 4   4460              air-travel
## 5   3503 customs-and-immigration
## 6   3296                schengen
## 7   2058                 transit
## 8   1695              passports
## 9   1665         indian-citizens
## 10  1517                  trains
## 11  1456                  canada
## 12  1340                luggage
## 13  1258                tickets
## 14  1201    international-travel
## 15  1199              paperwork
## 16  1193        public-transport
## 17  1167           visa-refusals
## 18  1139                 germany
## 19  1107                airports
## 20  1056                  europe
## 21  1046                   legal
## 22  1008                   india
```

```
# base
df_base_1 <- function(df1) {
  base <- na.omit(df1[ df1$Count > 1000 , c("Count","TagName") ])
  base <- base[order(base$Count, decreasing =TRUE),]
  rownames(base)<- NULL
  base
}
base_1 <- df_base_1(Tags)


#dplyr
df_dplyr_1 <- function(df1) {
  dplyr <- Tags %>% filter(Count>1000) %>% select(Count,TagName) %>% arrange(desc(Count))
  dplyr
}
dplyr_1 <- df_dplyr_1(Tags)



#data.table
df_table_1 <- function(df1) {
  tags <- setDT(Tags)
  table <- tags[Count >1000, .(Count,TagName)][order(-Count)]
  table
}
table_1 <- df_table_1(Tags)
```

Porównanie ramek oraz czasu wykonania zadań przez funkcje

```
#Sprawdzam, czy powstałe ramki pokazują to samo
dplyr::all_equal(sqldf_1, base_1,dplyr_1)
```

```
## [1] TRUE
```

```
dplyr::all_equal(dplyr_1,table_1)
```

```
## [1] TRUE
```

```
#Porównuję czasy wykonania
microbenchmark::microbenchmark(
  sqldf = df_sql_1(Tags),
  base = df_base_1(Tags),
  dplyr = df_dplyr_1(Tags),
  data.table = df_table_1(Tags),
  times=10L
  )
```

```
## Unit: microseconds
##        expr     min      lq     mean   median      uq     max neval
##       sqldf 10937.9 11188.5 16138.01 11308.50 12961.4 56470.8    10
##        base   390.8   409.9   859.67   506.40   585.8  4174.3    10
##       dplyr  6674.1  6780.8  8020.20  6984.05  7582.8 15241.0    10
##  data.table   701.1   730.5  1190.88   832.10   969.3  4393.8    10
```

Zadanie 2

Funkcje oraz wynik

```
df_sql_2 <- function(df1,df2) {
  sqldf <- sqldf("SELECT Location, COUNT(*) AS Count
  FROM (
  SELECT Posts.OwnerUserId, Users.Id, Users.Location
  FROM Users
  JOIN Posts ON Users.Id = Posts.OwnerUserId
  )
  WHERE Location NOT IN ('')
  GROUP BY Location
  ORDER BY Count DESC
  LIMIT 10")
  sqldf
}
sqldf_2 <- df_sql_2(Posts,Users)
```

Interpretacja

Najpierw tworzę ramkę danych składajacą się z połączonej kolumnowo ramki Users oraz Posts przy czym znajdują się w niej tylko te wiersze, w których wartości z kolumny OwnerUserId są równe wartościom z kolumny Id. Z tej ramki wybieram kolumny o nazwie OwnerUserId, Id oraz Location. Usuwam wiesze w których lokacja z kolumny Location jest pustą nazwą. Następnie w powstałej ramce zliczam ile razy każda lokacja pojawiła się w kolumnie Location. Po tym ustawiam wiersze ramki danych w porządku malejącym według wyników zliczeń, które są zapisane w kolumnie Count. Ograniczam ramkę do pierwszych 10 wierszy.

W skrócie:

Tworzę ramkę danych, w której policzone jest, ile każda lokacja, która nie ma pustej nazwy i której Id jest równe OwnerUserId, występuje w ramce Users. Wyniki te są w kolejności malejącej. Wybieram tylko 10 pierwszych wierszy.

```
#wynik
sqldf_2
```

```
##                         Location Count
## 1      Christchurch, New Zealand  2765
## 2                    New York, NY  1788
## 3         London, United Kingdom  1708
## 4                             UK  1590
## 5  Sunshine Coast QLD, Australia  1550
## 6                       Australia  1183
## 7               Vancouver, Canada   967
## 8                     Netherlands   935
## 9             on the server farm    924
## 10                  Pennsylvania   921
```

```
# base
df_base_2 <- function(df1,df2) {
  Posts <- as.data.frame(Posts)
  Users <- as.data.frame(Users)
  pom <- merge(Users,Posts, by.x="Id",by.y="OwnerUserId")
  pom <- pom[,c("Id","Location")]
  base <- pom[!(pom$Location %in% ("")),]
  base <- aggregate(x = base$Location,by= base["Location"], FUN = length)
  colnames(base)[2] <- "Count"
  base <- base[order(base$Count, decreasing = TRUE),]
  rownames(base)<- NULL
  head(base,10)
}
base_2<-df_base_2(Posts,Users)


#dplyr
df_dplyr_2 <- function(df1,df2) {
  dplyr <- Users %>% inner_join(Posts, by=c("Id"="OwnerUserId")) %>%
    select(Id,Location) %>% filter(Location!='') %>% select(Location) %>%
    group_by(Location) %>% count(Location,sort=TRUE) %>% head(10) %>% rename(Count=n)

  dplyr
}
dplyr_2 <- df_dplyr_2(Posts,Users)

#data.table
df_table_2 <- function(df1,df2) {
  users <- setDT(Users)
  posts <- setDT(Posts)
  table <- merge(users,posts,by.x="Id",by.y="OwnerUserId")
  table <- table[Location != (''),.N,by=.(Location)][order(-N)][,.(Location,Count=N)]
  head(table,10)
}
table_2 <- df_table_2(Posts,Users)
```

Porównanie ramek oraz czasu wykonania zadań przez funkcje

```
# Sprwadzenie poprawności i czasóW


dplyr::all_equal(sqldf_2, base_2,dplyr_2)
```

```
## [1] TRUE
```

```
dplyr::all_equal(dplyr_2,table_2)
```

```
## [1] TRUE
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_2(Posts,Users),
  base = df_base_2(Posts,Users),
  dplyr = df_dplyr_2(Posts,Users),
  data.table = df_table_2(Posts,Users),times=10L)
```

```
## Unit: milliseconds
##        expr       min        lq      mean     median        uq       max neval
##       sqldf 1503.2686 1571.1964 1715.5528 1701.59165 1835.6411 2036.9217    10
##        base  341.9370  388.3367  429.7974  400.64825  450.9465  685.1418    10
##       dplyr  143.6098  147.8042  163.3538  148.74760  152.2338  231.4843    10
##  data.table   55.1908   56.1238  102.0602   62.86095  120.8145  328.9320    10
```

Zadanie 3

Funkcje oraz wynik

```
#sqldf
df_sql_3 <- function(df1) {
  sqldf<-sqldf("SELECT Year, SUM(Number) AS TotalNumber
  FROM (
  SELECT
  Name,
  COUNT(*) AS Number,
  STRFTIME('%Y', Badges.Date) AS Year
  FROM Badges
  WHERE Class = 1
  GROUP BY Name, Year
  )
  GROUP BY Year
  ORDER BY TotalNumber")
  sqldf
}
sqldf_3 <- df_sql_3(Badges)
```

Interpretacja

Z ramki Badges usuwam wiersze, w których wartości z kolumny Class nie są równe 1. Z tej ramki tworzę nową ,gdzie w kolumnie Number zliczona jest ilość w jakiej występują konkretne pary Name i Year. Następnie liczę sumę wystąpień dla każdego pojawiającego się roku (Year). Wyniki zapisane w kolejności rosnącej znajdują się w kolumnie nazwanej TotalNumber.

W skrócie:

Zliczam ile razy Year, przy którym Class jest równe 1, występuje w ramce i ustawiam wyniki rosnąco.

```
#wynik
sqldf_3
```

```
##    Year TotalNumber
## 1  2011          16
## 2  2012          23
## 3  2013          66
## 4  2021         153
## 5  2014         197
## 6  2020         265
## 7  2015         328
## 8  2016         509
## 9  2017         552
## 10 2018         697
## 11 2019         718
```

```
#base
df_base_3 <- function(df1) {
  date <- as.POSIXct(Badges$Date, format ="%Y-%m-%dT%H:%M:%S%OS" )
  Badges$Year <- format(date, format="%Y")
  pom <- aggregate(x=Badges[Badges$Class == 1,c("Name","Year")],
                   by=Badges[Badges$Class == 1,c("Name","Year")],
                   FUN = length)
  pom[4]<- NULL
  colnames(pom)[3] <- "Number"
  rownames(pom) <- NULL
  pom <- aggregate(x=pom$Number, by=list(Year=pom$Year),FUN = sum)
  colnames(pom)[2]<- "TotalNumber"
  pom <- pom[order(pom$TotalNumber),]
  rownames(pom) <- NULL
  pom


}

base_3 <- df_base_3(Badges)


#dplyr
df_dplyr_3 <- function(df1) {
  date <- as.POSIXct(Badges$Date, format ="%Y-%m-%dT%H:%M:%S%OS" )
  Badges$Year <- format(date, format="%Y")
  dplyr <- Badges %>% filter(Class==1) %>% group_by(Name,Year) %>%
    count(Name,Year) %>% rename(Number=n) %>% group_by(Year) %>%
    summarize(TotalNumber=sum(Number)) %>% arrange(TotalNumber)
  dplyr
}

dplyr_3 <- df_dplyr_3(Badges)

#data.table
df_table_3 <- function(df1) {
  date <- as.POSIXct(Badges$Date, format ="%Y-%m-%dT%H:%M:%S%OS" )
  Badges$Year <- format(date, format="%Y")
  badges <- setDT(Badges)
  table <- (badges[Class==1,.N,by=.(Name,Year)]
            [,.(Name,Year,Number=N)][,sum(Number),by= Year]
            [,.(Year,TotalNumber=V1)][order(TotalNumber)])

  table
}
table_3 <- df_table_3(Badges)
```

Porównanie ramek oraz czasu wykonania zadań przez funkcje

```
# Sprwadzenie poprawności i czasóW


dplyr::all_equal(sqldf_3, base_3,dplyr_3)
```

```
## [1] TRUE
```

```
dplyr::all_equal(dplyr_3,table_3)
```

```
## [1] TRUE
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_3(Badges),
  base = df_base_3(Badges),
  dplyr = df_dplyr_3(Badges),
  data.table = df_table_3(Badges),times=10L
)
```

```
## Unit: milliseconds
##        expr       min        lq      mean    median        uq       max neval
##       sqldf  144.3795  146.6166  148.1525  147.2797  149.7904  155.4391    10
##        base 2749.9925 2753.2980 2780.1091 2778.6581 2808.6433 2815.2261    10
##       dplyr 2770.9019 2775.6630 2819.9010 2817.0858 2831.8966 2937.8162    10
##  data.table 2752.1013 2752.7826 2759.5180 2758.6866 2762.9679 2774.0268    10
```

Zadanie 4

Funkcje oraz wynik

```
#sqldf
df_sql_4 <- function(df1,df2) {
  sqldf<-sqldf("SELECT
  Users.AccountId,
  Users.DisplayName,
  Users.Location,
  AVG(PostAuth.AnswersCount) as AverageAnswersCount
  FROM
  (
  SELECT
  AnsCount.AnswersCount,
  Posts.Id,
  Posts.OwnerUserId
  FROM (
  SELECT Posts.ParentId, COUNT(*) AS AnswersCount
  FROM Posts
  WHERE Posts.PostTypeId = 2
  GROUP BY Posts.ParentId
  ) AS AnsCount
  JOIN Posts ON Posts.Id = AnsCount.ParentId
  ) AS PostAuth
  JOIN Users ON Users.AccountId=PostAuth.OwnerUserId
  GROUP BY OwnerUserId
  ORDER BY AverageAnswersCount DESC, AccountId ASC
  LIMIT 10
")
  sqldf
}
sqldf_4 <- df_sql_4(Users,Posts)
```

Interpretacja

Na początku tworzę ramkę danych, w której mam zliczone ile razy występuje konkretne ParentId, przy którym PostTypeId jest równe 2. Kolumna z wynikami zliczenia nazywa się AnswersCount. Łączę powyższą ramkę z ramką Posts, ale wybierając tylko te wiersze w których Id i Parent Id są sobie równe. Z powstałej ramki wybieram tylko kolumny Id, OwnerUserId oraz AnswersCount. Tę ramkę znowu łączę kolumnami z ramką Posts, ale wybierając tylko wiersze, w których Id równa się ParentId. Z powstałej ramki danych zliczam średnią wartość AnswersCount dla unikatowych trójek AccountId, DisplayName oraz Location. Wyniki zliczonych średnich znajdują się w kolumnie o nazwie AverageAnswersCount. Wybieram tylko 10 pierwszych wierszy, zaczynając od tych z najwyższą wartością w AverageAnswersCount a następnie jeśli jakieś dwie wartości AccountId są równe, to układając wiersze z nimi w kolejności rosnącej.

W skrócie:

Dla unikatowych trójek AccountId, DisplayName oraz Location liczę średnią z liczby wystąpień ParentId, które równa się Id , którego wartość PostTypeId równa się 2 oraz którego wartość OwnerUserId równa się AccountId. Wybieram tylko 10 pierwszych wierszy.

```
#wynik
sqldf_4
```

```
##    AccountId     DisplayName          Location AverageAnswersCount
## 1        280           csmba San Francisco, CA                  11
## 2      40811          vocaro      San Jose, CA                  11
## 3        204            Josh         Australia                  10
## 4      44093   Emma Arbogast         Salem, OR                  10
## 5      11758        rvarcher Oklahoma City, OK                   9
## 6      19588      JD Isaacks       Atlanta, GA                   8
## 7      20473     Jeremy Boyd       Houston, TX                   8
## 8      42364        Petrogad                                    8
## 9      54571       Christian                                    8
## 10     79346 Thomas Matthews        California                   8
```

```
#base
df_base_4 <- function(df1,df2) {
  Posts <- as.data.frame(Posts)
  Users <- as.data.frame(Users)
  AnsCount <- Posts[Posts$PostTypeId==2,]
  AnsCount <- aggregate(AnsCount$ParentId,AnsCount["ParentId"],length)
  colnames(AnsCount)[2] <- "AnswersCount"
  pom <- merge(Posts,AnsCount,by.x="Id",by.y="ParentId")
  PostAuth <- pom[,c("AnswersCount","Id","OwnerUserId")]
  base <- merge(Users,PostAuth,by.x ="AccountId", by.y="OwnerUserId")
  base <- aggregate(x= base$AnswersCount,
                    b= base[c("AccountId","DisplayName","Location")],
                    FUN = function(x) c(AverageAnswersCount= mean(x)))
  colnames(base)[4] <- "AverageAnswersCount"
  base <- base[ order(base$AverageAnswersCount,base$AccountId , decreasing = c(TRUE, FALSE)),
]


  head(base,10)

}


base_4 <- df_base_4(Users,Posts)


#dplyr
df_dplyr_4 <- function(df1,df2) {
  AnsCount <- Posts %>% filter(PostTypeId==2) %>% group_by(ParentId) %>%
    count(ParentId) %>% rename(AnswersCount=n)
  PostAuth <- Posts %>% inner_join(AnsCount, by=c("Id"="ParentId")) %>%
    select(AnswersCount,Id,OwnerUserId)
  dplyr <- Users %>% inner_join(PostAuth,by=c("AccountId"="OwnerUserId")) %>%
    group_by(AccountId,DisplayName,Location) %>%
    summarize(AverageAnswersCount = mean(AnswersCount, na.rm=TRUE)) %>%
    arrange(desc(AverageAnswersCount),AccountId) %>%  head(10)

  dplyr

}
dplyr_4 <- df_dplyr_4(Users,Posts)


#data.table
df_table_4 <- function(df1,df2) {
  posts <- setDT(Posts)
  users <- setDT(Users)
  AnsCount<- posts[PostTypeId==2,.N,by=.(ParentId)][,.(ParentId,AnswersCount=N)]
  AnsCountPosts <- merge(Posts,AnsCount,by.x="Id",by.y="ParentId")
  PostAuth <- AnsCountPosts[,.(AnswersCount,Id,OwnerUserId)]
  PostAuthUsers <- merge(Users,PostAuth,by.x="AccountId",by.y="OwnerUserId")
  table <- (PostAuthUsers[,mean(AnswersCount),by=.(AccountId,DisplayName,Location)]
            [,.(AccountId,DisplayName,Location,AverageAnswersCount=V1)]
            [order(-AverageAnswersCount,AccountId)])
```

```
  head(table,10)

}
table_4 <- df_table_4(Users,Posts)
```

Porównanie ramek oraz czasu wykonania zadań przez funkcje

```
# Sprwadzenie poprawności i czasóW
dplyr::all_equal(sqldf_4, base_4,dplyr_4)
```

```
## [1] TRUE
```

```
dplyr::all_equal(dplyr_4,table_4)
```

```
## [1] TRUE
```

```
microbenchmark::microbenchmark(
  base = df_base_4(Users,Posts),
  sqldf = df_sql_4(Users,Posts),
  dplyr = df_dplyr_4(Users,Posts),
  data.table = df_table_4(Users,Posts),times=10L
)
```

```
## Unit: milliseconds
##        expr       min        lq       mean    median        uq       max neval
##        base  475.2486  488.6576  527.56481  546.3662  553.3026  574.6030    10
##       sqldf 1582.5931 1589.6445 1611.60848 1596.3783 1653.9613 1658.0699    10
##       dplyr  531.2629  561.7016  596.62167  588.1137  637.6246  673.8758    10
##  data.table   30.6729   36.3309   69.01279   36.6819   38.2624  308.7933    10
```

Podsumowanie

Trudno określić, które funkcje działały najszybciej . W dwóch z czterech zadań najlepsze wyniki miały funkcje z data.table, ale wszystkie inne wyniki były mocno zróżnicowane, w zależności od wykonywanego zadania.