

Praca Domowa numer 1

7 kwietnia 2019

Jakub Wiśniewski

0 Przygotowanie do pracy

Przed przystąpieniem do pracy pobieram wymagane pakiety oraz ramki danych

```
library(sqldf)
library(dplyr)
library(data.table)

Badges      <- read.csv("Badges.csv")
Comments    <- read.csv("Comments.csv")
PostLinks    <- read.csv("PostLinks.csv")
Posts        <- read.csv("Posts.csv")
Tags         <- read.csv("Tags.csv")
Users        <- read.csv("Users.csv")
Votes        <- read.csv("Votes.csv")
```

1 Zadanie 2

1.1 Zapytanie SQL

```
sqldf("SELECT
Posts.ID,
Posts.Title,
Posts2.PositiveAnswerCount
FROM Posts
JOIN

(SELECT
Posts.ParentID,
COUNT(*) AS PositiveAnswerCount
FROM Posts
WHERE Posts.PostTypeID=2 AND Posts.Score>0
GROUP BY Posts.ParentID )

AS Posts2
ON Posts.ID=Posts2.ParentID
```

```
ORDER BY Posts2.PositiveAnswerCount DESC
LIMIT 10") -> df2
```

1.2 Bazowy R

```
# wybieram selecta w nawiasach
Posts2 <- as.data.frame(table(Posts[Posts$PostTypeId == 2 & Posts$Score > 0 ,
                                "ParentId"]), stringsAsFactors = FALSE)

# przyporządkowuję nazwy w kolumnach
colnames(Posts2) <- c("ParentId", "PositiveAnswerCount")
lewy <- Posts[c("Id", "Title")]

# łączę
x <- merge(x= lewy, y=Posts2, by.x = 'Id', by.y = 'ParentId', all.x = TRUE, all.y = TRUE)

# sortuję malejąco oraz wybieram 10 pierwszych
x <- head(x[order(-x$PositiveAnswerCount),], n = 10)
rownames(x) <- NULL

# sprawdzam czy ramki danych są takie same
all.equal(x, df2)

## [1] TRUE
```

1.3 Dplyr

```
# nakładam filtry, biorę interesującą mnie grupę, liczę poszczególne wystąpienia i grupuję
x <- Posts %>% filter(PostTypeId == 2 & Score > 0 )
x <- x %>% select(ParentId) %>% count(ParentId) %>% group_by(ParentId)

# zmieniam nazwy
Posts2 <- x
colnames(Posts2)[2] <- "PositiveAnswerCount"

# przygotowuję się do złączenia
x <- Posts %>% select(Id, Title)

# łączę Id i Title z naszym Posts2
xy <- inner_join(x, Posts2, by = c("Id" = "ParentId"))

# sortuję po liczbie odpowiedzi i wybieram 10 pierwszych
xy <- xy %>% arrange(desc(PositiveAnswerCount))
xy <- slice(xy, 1:10)

# sprawdzam czy się zgadza
all_equal(xy, df2)

## [1] TRUE
```

1.4 Data.table

```
# zmieniam wynik zapytania SQL na data.table do późniejszego porównania
df2 <- as.data.table(df2)

# zamieniam na data table i wykonuję przekształcenie do Posts2
Posts1 <- as.data.table(Posts)
Posts2 <- Posts1[PostTypeId == 2 & Score > 0, .N, by = ParentId ]
setnames(Posts2, "N", "PositiveAnswerCount" )

# wybieram kolumny
Posts1 <- Posts1[, c("Id", "Title")]

# i sortuję Posts1 po Id oraz Posts2 po ParentId
setkey(Posts1, Id)
setkey(Posts2, ParentId)

# dzięki wcześniejszej operacji wykonuję Joina
Posts1 <- Posts1[Posts2, nomatch = 0]

# sortuję malejąco i wybieram 10 pierwszych
setorder(Posts1, -PositiveAnswerCount)
Posts1 <- Posts1[1:10]

# sprawdzam czy są tożsame
all.equal(Posts1, df2)

## [1] TRUE
```

2 Zadanie 5

2.1 Zapytanie SQL

```
sqldf("SELECT Posts.Title, CmtTotScr.CommentsTotalScore FROM (

    SELECT PostID, UserID, SUM(Score) AS CommentsTotalScore FROM Comments
    GROUP BY PostID, UserID) AS CmtTotScr

    JOIN Posts ON Posts.ID=CmtTotScr.PostID AND Posts.OwnerUserId=CmtTotScr.UserID
    WHERE Posts.PostTypeId=1
    ORDER BY CmtTotScr.CommentsTotalScore DESC
    LIMIT 10") -> df2
```

2.2 Bazowy R

```
CmtTotScr <- aggregate(Comments$Score , by = list(Comments$PostId, Comments$UserId), FUN=sum)
```

```

colnames(CmtTotScr) <- c("PostId", "UserId", "CommentsTotalScore")

C <- merge(Posts, CmtTotScr, by.x = c("Id", "OwnerUserId"), by.y = cbind("PostId", "UserId"))
C <- C[C$PostTypeId == 1, c("Title", "CommentsTotalScore")]
C <- C[order(-C$CommentsTotalScore),]

rownames(C) <- NULL
df1 <- head(C, n=10)
all.equal(df1, df2)

## [1] TRUE

```

2.3 Dplyr

```

CmtTotScr <- Comments %>% select(PostId, UserId, Score)
CmtTotScr <- CmtTotScr %>% group_by(PostId, UserId) %>% summarise(CommentsTotalScore = sum(Score))
colnames(x)[3] <- "CommentsTotalScore"

## Error in names(x) <- value: 'names' attribute [3] must be the same length as the vector [2]

Posts1 <- Posts %>% filter(PostTypeId == 1)
CmtTotScr <- left_join(Posts1, CmtTotScr, by = c("Id" = "PostId", "OwnerUserId" = "UserId"))

df1 <- CmtTotScr %>% arrange(desc(CommentsTotalScore)) %>% top_n(10) %>% select(Title, CommentsTotalScore)

## Selecting by CommentsTotalScore

all_equal(df2, df1)

## [1] TRUE

```

2.4 Data.table

```

dt2 <- as.data.table(df2)
Posts_table <- as.data.table(Posts)
Comments_table <- as.data.table(Comments)
CmtTotScr <- Comments_table[,.(CommentsTotalScore = sum(Score)), by = .(PostId, UserId)]

setkey(CmtTotScr, PostId, UserId)
setkey(Posts_table, Id, OwnerUserId)

Posts_table <- Posts_table[CmtTotScr, nomatch = 0 ]

Posts_table <- Posts_table[PostTypeId == 1, ,]

Posts_table <- Posts_table[, .(Title, CommentsTotalScore)]

setorder(Posts_table, -CommentsTotalScore)
dt1 <- Posts_table[1:10, ,]

```

```
all.equal(dt1, dt2)

## [1] TRUE
```

3 Zadanie 6

3.1 Zapytanie SQL

```
sqldf("SELECT DISTINCT
      Users.Id,
      Users.DisplayName,
      Users.Reputation,
      Users.Age,
      Users.Location
      FROM ( SELECT Name, UserID FROM Badges WHERE Name IN (
              SELECT Name FROM Badges WHERE Class=1
              GROUP BY Name HAVING COUNT(*) BETWEEN 2 AND 10 ) AND Class=1 ) AS ValuableBadges
      JOIN Users ON ValuableBadges.UserId=Users.Id") -> df2
```

3.2 Bazowy R

Najpierw zliczamy liczbę pojawień się zmiennej "Name", przy "Class" równym 1. Następnie zmieniamy nazwy kolumn i wybieramy te, rzędy, w których "Count" wynosi między 2 a 10

```
x <- as.data.frame(table(Badges[Badges$Class == 1, "Name"]), stringsAsFactors = FALSE)
colnames(x) <- c("Name" , "Count")
x <- x[x$Count <= 10 & x$Count >= 2 ,]
rownames(x) <- NULL
```

Następnie wybieramy te Nazwy z y , które występują również w x.

```
y <- Badges[Badges$Class == 1, c("Name", "UserId")]
y <- y[y$Name %in% x$Name,]
rownames(y) <- NULL
ValuableBadges <- y
```

Teraz wybieramy z Users te ID, które pojawiają się również w ValuableBadges. Następnie sortujemy zapytanie słowe, aby było takie same jak to, które dostaliśmy w bazowym R. Na koniec sprawdzamy

```
u <- Users[Users$Id %in% ValuableBadges$UserId, c("Id", "DisplayName", "Reputation", "Age", "Location")]
rownames(u) <- NULL
u <- u[order(u$Id),]
df2 <- df2[order(df2$Id),]
rownames(df2) <- NULL

all.equal(u, df2)

## [1] TRUE
```

3.3 Dplyr

Na początku dsiewam te obserwacje z Badges, które nie mają Class równej 1, następnie wybieram nazwę i zliczam jej wystąpienia

```
x <- Badges %>% filter(Class == 1) %>% select(Name) %>% count(Name)
```

zmieniam nazwę 2 kolumny na count

```
colnames(x)[2] <- "count"
```

ponownie filtruję te, które nie mają liczności między 2 a 10

```
x <- x %>% filter(count >= 2 & count <= 10)
```

Biorę te obserwacje z Badges, których nazwa występuje w x, następnie biorę te z Class równym jeden i wybieram kolumny Name i UserId.

```
y <- Badges %>% filter(Badges$Name %in% x$Name) %>% filter(Class == 1)%>% select(Name , UserId)
```

Wybieram kolumny do Joina.

```
u <- Users %>% select(Id, DisplayName, Reputation, Age, Location)
```

Łączę po Id i UserId.

```
df1 <- inner_join(u, y, by = c("Id" = "UserId" ))
```

Wybieram unikatowe obserwacje.

```
df1 <- df1 %>% distinct(Id, DisplayName, Reputation, Age, Location)
```

Na koniec sprawdzam czy ramki danych są takie same.

```
all.equal(df1,df2 )
```

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

3.4 Data.table

Na początek zamieniam wynik zapytania i potrzebne ramki danych na data.table

```
dt2 <- data.table(df2)
Badges_table <- data.table(Badges)
User_table <- data.table(Users)
```

Wybieram obserwacje z Class równym 1, następnie grupuję po nazwie i zliczam poszczególne wystąpienia nazw jako Count.

```
x <- Badges_table[Class == 1, .(Count = .N), by = Name]
```

Wybieram te obserwacje z Count pomiędzy 2 a 10.

```
x <- x[Count <=10 & Count >= 2,,]
```

Wybieram obserwacje, których nazwa jest w x, mają Class równy 1, oraz zwracam kolumny Name i UserId jako data.table

```
ValuableBadges <- Badges_table[Badges$Name %in% x$Name & Class == 1, .(Name, UserId)]
```

Sortuję tablice danych przygotowując je do złączenia. Dzięki setkey markuję je jako posortowane.

```
setkey(ValuableBadges, UserId)
setkey(User_table, Id)
```

Łączę tablice danych

```
User_table <- User_table[ValuableBadges, nomatch= 0]
```

Wybieram interesujące mnie kolumny, następnie biorę te, których obserwacje są unikatowe

```
dt1 <- unique(User_table[,.(Id, DisplayName,Reputation, Age, Location),])
```

Sortuję tablice, markuję je jako posortowane. Sprawdzam czy tablice danych są takie same.

```
setkey(dt1, Id)
setkey(dt2, Id)
all.equal(dt1, dt2)

## [1] TRUE
```