Praca Domowa 1

Filip Chrzuszcz

April 15, 2019

Contents

1 `Zadanie 1` **2**
- 1.1 SQL . . . 2
- 1.2 Base . . . 3
- 1.3 Dplyr . . . 3
- 1.4 DT . . . 4
- 1.5 Benchmark . . . 4

2 `Zadanie 2` **5**
- 2.1 SQL . . . 5
- 2.2 Base . . . 5
- 2.3 Dplyr . . . 6
- 2.4 DT . . . 6
- 2.5 Benchmark . . . 6

3 `Zadanie 3` **7**
- 3.1 SQL . . . 7
- 3.2 Base . . . 8
- 3.3 Dplyr . . . 8
- 3.4 DT . . . 9
- 3.5 Benchmark . . . 9

4 `Zadanie 4` **10**
- 4.1 SQL . . . 10
- 4.2 Base . . . 11
- 4.3 Dplyr . . . 11
- 4.4 DT . . . 12
- 4.5 Benchmark . . . 12

5 `Zadanie 5` **13**
- 5.1 SQL . . . 13
- 5.2 Base . . . 13
- 5.3 Dplyr . . . 13
- 5.4 DT . . . 14
- 5.5 Benchmark . . . 14

6 `Zadanie 6` **15**
- 6.1 SQL . . . 15
- 6.2 Base . . . 15
- 6.3 Dplyr . . . 16
- 6.4 DT . . . 16
- 6.5 Benchmark . . . 16

7 `Zadanie 7` **17**
- 7.1 SQL . . . 17
- 7.2 Base . . . 18
- 7.3 Dplyr . . . 19
- 7.4 DT . . . 19
- 7.5 Benchmark . . . 20

Zadanie zaczynam od wczytania wszelkich potrzebnych pakietów oraz ustawienia działania R.

```
options(stringsAsFactors=FALSE)
Tags <- read.csv("Tags.csv")
Users <- read.csv("Users.csv")
Votes <- read.csv("Votes.csv")
Posts <- read.csv("Posts.csv")
PostLinks <- read.csv("PostLinks.csv")
Comments <- read.csv("Comments.csv")
Badges <- read.csv("Badges.csv")
library(knitr)
library(dplyr)
library(data.table)
library(sqldf)
library(microbenchmark)
```

1 Zadanie 1

1.1 SQL

Celem jest znalezienie użytkowników, którzy uzyskali najwięcej Likes, wyswietlamy ich dane wraz z najwyżej ocenionym pytaniem.

```
df_sql_1 <- function(x,y){
sqldf("SELECT
  Users.DisplayName,
  Users.Age,
  Users.Location,
  SUM(Posts.FavoriteCount) AS FavoriteTotal,
  Posts.Title AS MostFavoriteQuestion,
  MAX(Posts.FavoriteCount) AS MostFavoriteQuestionLikes
FROM Posts
JOIN Users ON Users.Id=Posts.OwnerUserId
WHERE Posts.PostTypeId=1
GROUP BY OwnerUserId
ORDER BY FavoriteTotal DESC
LIMIT 10")
}
```

1.2 Base

Polecenie w base wyrzuca pewne warningi, lecz nie mają one znaczenia dla końcowego outputu.

```
df_base_1 <- function(x,y){
  pos <- Posts[Posts$PostTypeId==1,]
  data <- merge(pos,Users,by.x="OwnerUserId",by.y="Id")
  maxi <- aggregate(data$FavoriteCount,by=data["OwnerUserId"],FUN=max,na.rm=TRUE) #bierzemy maksa
  sumi <- aggregate(data$FavoriteCount,by=data["OwnerUserId"],FUN=sum,na.rm=TRUE) #bierzemy sume
  pom <- merge(data,maxi,by.x=c("OwnerUserId","FavoriteCount"),by.y=c("OwnerUserId","x"))
  pom <- merge(pom,sumi,by="OwnerUserId")
  pom <- pom[,c("DisplayName","Age","Location","x","Title","FavoriteCount")]#wybieramy co trzeba
  pom <- pom[order(pom$x,decreasing = TRUE),]
  pom <- pom[1:10,]
  colnames(pom) <- c("DisplayName","Age","Location","FavoriteTotal",
                     "MostFavoriteQuestion","MostFavoriteQuestionLikes")#zgodnosc nazw
  pom$MostFavoriteQuestion <- as.character(pom$MostFavoriteQuestion)
  return(pom)
}
dplyr::all_equal(df_sql_1(Posts,Users),df_base_1(Posts,Users))

## [1] TRUE
```

1.3 Dplyr

```
df_dplyr_1 <- function(x,y){
  pos <- filter(Posts,PostTypeId==1)
  pom <- inner_join(Users,pos,by=c("Id"="OwnerUserId"))
  data <-group_by(pom,Id) %>%
    summarise(FavoriteTotal = sum(FavoriteCount,na.rm=TRUE),MostFavoriteQuestionLikes=max(FavoriteCount,na.rm=TRUE
    rename(OwnerUserId=Id)
  data <- inner_join(data,pom,by=c("OwnerUserId"="Id")) %>% #niezbedne joiny
    arrange(OwnerUserId,desc(MostFavoriteQuestionLikes)) %>%
    group_by(OwnerUserId) %>% top_n(n=1,wt=FavoriteCount) %>%
    select(DisplayName,Age,Location,FavoriteTotal,Title,MostFavoriteQuestionLikes,OwnerUserId) %>%
    rename(MostFavoriteQuestion=Title) %>% #zgodnosc kolumn
    ungroup() %>% #pozbycie sie factorow
    select(-OwnerUserId) %>%
    arrange(desc(FavoriteTotal)) %>% head(10)
  data$MostFavoriteQuestionLikes <- as.integer(data$MostFavoriteQuestionLikes) #odpowiednie typu kolumn
  data$MostFavoriteQuestion <- as.character(data$MostFavoriteQuestion)
  return(data)
}
dplyr::all_equal(df_sql_1(Posts,Users),df_dplyr_1(Posts,Users))

## [1] TRUE
```

1.4 DT

```
df_dt_1 <- function(x,y){
pos <- as.data.table(Posts)[PostTypeId==1]
maxi <- pos[,max(FavoriteCount,na.rm=TRUE),by=OwnerUserId] #bierzemy maksa oraz sume
sumi <- pos[,sum(FavoriteCount,na.rm=TRUE),by=OwnerUserId]
colnames(sumi) <- c("OwnerUserId","FavoriteTotal")
pom <- merge(pos,maxi,by.x=c("OwnerUserId","FavoriteCount"),by.y=c("OwnerUserId","V1"))
pom <- merge(pom,sumi,by="OwnerUserId")[,c("OwnerUserId","FavoriteTotal","FavoriteCount","Title")]
colnames(pom) <- c("OwnerUserId","FavoriteTotal","MostFavoriteQuestionLikes","MostFavoriteQuestion")
post <- merge(pom,Users,by.x="OwnerUserId",by.y="Id")[order(FavoriteTotal,decreasing = TRUE)][1:10]
post <- post[,c("DisplayName","Age","Location","FavoriteTotal",
                "MostFavoriteQuestion","MostFavoriteQuestionLikes")]
post$MostFavoriteQuestionLikes <- as.integer(post$MostFavoriteQuestionLikes)
post$MostFavoriteQuestion <- as.character(post$MostFavoriteQuestion)
return(post)
}
dplyr::all_equal(df_sql_1(Posts,Users),df_dt_1(Posts,Users))
```

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

1.5 Benchmark

Wykonuje benchmark na małej próbce z powodu słabej wydajności mojego komputera, jednakże nie ma to zbyt wielkiego wpływu na jego rzetelność.

```
microbenchmark::microbenchmark(                   base = df_base_1(Posts,Users),
                                                  dplyr=df_dplyr_1(Posts,Users),
                                                   data.table=df_dt_1(Posts,Users),
                                                   sql = df_sql_1(Posts,Users),times=10)
```

```
## Unit: milliseconds
##        expr        min         lq      mean    median        uq      max
##        base 1466.64794 1756.35613 2904.0894 2725.1277 3500.6723 4795.379
##       dplyr 2145.78783 2515.30033 3048.7399 2829.7944 3410.1360 5035.259
##  data.table   62.39767   89.71782  441.9593  307.1432  466.8701 1857.646
##         sql  663.55614 1087.44950 2152.4872 1874.5345 2714.7396 5608.224
##  neval
##     10
##     10
##     10
##     10
```

2 Zadanie 2

2.1 SQL

Znajdujemy pytania o najwiekszej licznie pozytywnie ocenionych odpowiedzi.

```
df_sql_2 <- function(x){
second <- 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")
}
```

2.2 Base

```
df_base_2 <- function(x){
  post1 <- Posts[,c("Id","Title")]
  Posts2 <- as.data.frame(table("ParentId" = Posts[Posts$PostTypeId==2 & Posts$Score>0,"ParentId"]),
                          responseName = "PositiveAnswerCount",stringsAsFactors = FALSE)
  wynik <- merge(post1,Posts2,by.x="Id",by.y="ParentId")
  sort <- order(wynik$PositiveAnswerCount,decreasing = TRUE) #sortowanko
  wynik <- wynik[sort,][0:10,]
  return(wynik)
}
all_equal(df_base_2(Posts),df_sql_2(Posts))
```

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

2.3 Dplyr

```
df_dplyr_2 <- function(x){
  tbl <-filter(Posts,PostTypeId==2) %>% filter(Score>0) %>%
    group_by(ParentId) %>%
    summarise(PositiveAnswerCount=n()) #fajna funkcja do liczenia wierszy w danej grupie
  tbl <- inner_join(Posts,tbl,by=c("Id"="ParentId")) %>%
    arrange(desc(PositiveAnswerCount)) %>%
    select(Id,Title,PositiveAnswerCount) %>% #bierzemy co trzeba
    head(n=10)
  return(tbl)
}
all_equal(df_dplyr_2(Posts),df_sql_2(Posts))

## [1] TRUE
```

2.4 DT

```
df_dt_2<- function(x){
tbl <- as.data.table(Posts)[PostTypeId==2][Score>0][,.(.N),by=.(ParentId)] #mozna fajnie laczyc polecenia
pos <- as.data.table(Posts) #wczytanie jako data.table daje wiele mozliwosc
tbl <- merge(tbl,pos,by.x="ParentId",by.y="Id")[order(N,decreasing = TRUE)][1:10][,c("ParentId","Title","N")]
colnames(tbl) <- c("Id","Title","PositiveAnswerCount")
return(tbl)
}
all_equal(df_dt_2(Posts),df_sql_2(Posts))

## [1] TRUE
```

2.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_2(Posts),
                               dplyr=df_dplyr_2(Posts),
                               data.table=df_dt_2(Posts),
                               sql = df_sql_2(Posts),times=10)

## Unit: milliseconds
##        expr       min        lq      mean    median        uq      max
##        base 149.04146 157.57984 170.20799 166.43546 185.25425 198.4127
##       dplyr  82.06254  92.42316 107.38018  95.98864 121.96938 157.7479
##  data.table  67.06970  73.95622  88.01858  90.14536  96.18941 116.4657
##         sql 388.18024 417.19077 465.70697 435.32226 498.45450 623.1942
##  neval
##     10
##     10
##     10
##     10
```

3 Zadanie 3

3.1 SQL

Znajdujemy pytania, które w danym roku otrzymały najwięcej UpVotes.

```r
df_sql_3 <- function(x,y){
sqldf("
SELECT
Posts.Title,
UpVotesPerYear.Year,
MAX(UpVotesPerYear.Count) AS Count
FROM (
SELECT
PostId,
COUNT(*) AS Count,
STRFTIME('%Y', Votes.CreationDate) AS Year
FROM Votes
WHERE VoteTypeId=2
GROUP BY PostId, Year
) AS UpVotesPerYear
JOIN Posts ON Posts.Id=UpVotesPerYear.PostId
WHERE Posts.PostTypeId=1
GROUP BY Year")
}
```

3.2 Base

```
df_base_3 <- function(x,y){
date <- as.Date(Votes$CreationDate, format='%Y-%m-%d') #odpowiednie wczytanie daty
date<- as.data.frame(as.numeric(format(date, '%Y')))
vot <- cbind(Votes,date)
colnames(vot) <- c("BountyAmount","CreationDate","Id","PostId","UserId","VoteTypeId","Year")
vot <- vot[vot$VoteTypeId==2,]
vot <- as.data.frame(table(Count = vot[,c("PostId","Year")]),stringsAsFactors = F)
pos <- Posts[Posts$PostTypeId==1,]
vot1 <- merge(vot,pos,by.x="PostId",by.y="Id")
vot <- aggregate(vot1$Freq,list(vot1$Year),FUN=max) #agregowanie danych
colnames(vot) <- c("Year","Count")
vot <- merge(vot,vot1,by.x=c("Year","Count"),by.y=c("Year","Freq"))
vot <- vot[,c("Year","Title","Count")]
vot <- vot[,c(2,1,3)]
vot$Count <- as.integer(vot$Count) #typ kolumn
return(vot)
}
dplyr::all_equal(df_sql_3(Posts,Votes),df_base_3(Posts,Votes))

## [1] TRUE
```

3.3 Dplyr

```
df_dplyr_3 <- function(x,y){
date <- as.Date(Votes$CreationDate, format='%Y-%m-%d')
date<- as.data.frame(as.numeric(format(date, '%Y')))
vot <- cbind(Votes,date)
colnames(vot) <- c("BountyAmount","CreationDate","Id",
                   "PostId","UserId","VoteTypeId","Year")
pom <- filter(vot,VoteTypeId==2) %>% group_by(PostId) %>% select(PostId,Year)
vot <- filter(vot,VoteTypeId==2) %>% group_by(PostId,Year) %>% summarise(Count = n())
vot <- inner_join(vot,pom,by="PostId")
pos <- filter(Posts,PostTypeId==1) #wybieramy co trzeba
vot1 <- inner_join(vot,pos,by=c("PostId"="Id")) %>% select(Title,Count,Year.x,PostId)
vot <- ungroup(vot1) #znow te facotry
vot <- distinct(vot, .keep_all = TRUE) %>% group_by(Year.x) %>% summarise(Count=max(Count))
vot <- inner_join(vot,vot1,by=c("Year.x"="Year.x","Count"="Count"))
vot <- distinct(vot, .keep_all = TRUE) %>% select(Year.x,Count,Title)#bierzemy unikalne
colnames(vot) <- c("Year","Count","Title")
vot$Count <- as.integer(vot$Count)
vot$Year <- as.character(vot$Year)
return(vot)
}
dplyr::all_equal(df_sql_3(Posts,Votes),df_dplyr_3(Posts,Votes))

## [1] TRUE
```

3.4 DT

```
df_dt_3 <- function(x,y){
date <- as.Date(Votes$CreationDate, format='%Y-%m-%d')
date<- as.data.frame(as.numeric(format(date, '%Y')))
vot <- cbind(Votes,date) #laczymy date z reszta tabeli
colnames(vot) <- c("BountyAmount","CreationDate","Id","PostId","UserId","VoteTypeId","Year")
vot <- as.data.table(vot)[VoteTypeId==2][, .(Count = .N), by = .(PostId,Year)]
pos <- as.data.table(Posts)[PostTypeId==1] #odpowiednie wczytanie
vot1 <- merge(vot,pos,by.x="PostId",by.y="Id")
vot <- vot1[,max(Count),by=Year]
vot <- merge(vot,vot1,by.x=c("Year","V1"),by.y=c("Year","Count"))[,c("Title","Year","V1")]
names(vot)[names(vot) == "V1"] = "Count" #odpowiednia nazwa
vot$Year <- as.character(vot$Year)
return(vot)
}

dplyr::all_equal(df_sql_3(Posts,Votes),df_dt_3(Posts,Votes))

## [1] TRUE
```

3.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_3(Posts,Votes),
                               dplyr=df_dplyr_3(Posts,Votes),
                               data.table=df_dt_3 (Posts,Votes),times=10)

## Unit: milliseconds
##        expr       min       lq      mean   median        uq      max neval
##        base 2540.2819 2648.167 2716.6483 2697.487 2793.4248 2995.115    10
##       dplyr 1411.2981 1428.186 1529.0191 1530.322 1605.9465 1648.394    10
##  data.table  689.6859  811.772  851.2928  855.296  887.0114 1033.180    10
```

4 Zadanie 4

4.1 SQL

Znajdujemy pytania, gdzie wystąpiła największa różnica pomiędzy najwyżej oceniona odpowiedzią, a odpowiedzią oznaczoną jako Accepted Answer.

```
df_sql_4 <-function(x){
sqldf("SELECT
  Questions.Id,
  Questions.Title,
  BestAnswers.MaxScore,
  Posts.Score AS AcceptedScore,
  BestAnswers.MaxScore-Posts.Score AS Difference
FROM (
SELECT Id, ParentId, MAX(Score) AS MaxScore
  FROM Posts
  WHERE PostTypeId==2
  GROUP BY ParentId
  ) AS BestAnswers
JOIN (
SELECT * FROM Posts
WHERE PostTypeId==1
  ) AS Questions
ON Questions.Id=BestAnswers.ParentId
JOIN Posts ON Questions.AcceptedAnswerId=Posts.Id
WHERE Difference>50
ORDER BY Difference DESC")
}
```

4.2 Base

```
df_base_4 <- function(x){
  BestAnswers <- Posts[Posts$PostTypeId==2,c("Id","ParentId","Score")]
  BestAnswers1 <- aggregate(Score~ParentId,BestAnswers,max,na.rm=T)
  BestAnswers2 <- merge(BestAnswers1,BestAnswers,all=F) #odpowiedni merge
  BestAnswers2 <- BestAnswers2[order(BestAnswers2$ParentId,BestAnswers2$Id),]
  BestAnswers2 <- BestAnswers2[!duplicated(BestAnswers2$ParentId),] #pozbycie sie duplikatow
  colnames(BestAnswers2) <- c("ParentId","MaxScore","Idx")
  Questions <- Posts[Posts$PostTypeId==1,]
  wynik <- merge(Questions,BestAnswers2,by.x="Id",by.y="ParentId")
  wynik <- merge(wynik,Posts,by.x="AcceptedAnswerId",by.y="Id")
  wynik$Diffrence <- wynik$MaxScore - wynik$Score.y
  wynik <- wynik[wynik$Diffrence>50,] #wybranie tych z roznica powyzej 50
  wynik <- wynik[order(wynik$Diffrence,decreasing = TRUE),]
  wynik <- wynik[,c("Id","Title.x","MaxScore","Score.y","Diffrence")]
  colnames(wynik) <- c("Id","Title","MaxScore",
                       "AcceptedScore","Difference") #pasujace tytuly kolumm
return(wynik)
}
dplyr::all_equal(df_sql_4(Posts),df_base_4(Posts))

## [1] TRUE
```

4.3 Dplyr

```
df_dplyr_4 <- function(x){
  pom <- filter(Posts,PostTypeId==2) %>% group_by(ParentId) %>% summarise(MaxScore = max(Score))
  pos <- filter(Posts,PostTypeId==2)
  pom <- inner_join(pom,pos,by=c("ParentId"="ParentId","MaxScore"="Score")) %>%
    arrange(ParentId,Id) %>%
    filter(!duplicated(ParentId,Id)) %>% select(Id,ParentId,MaxScore) #fajne laczenie polecen pipe'ami
  post <- filter(Posts,PostTypeId==1)
  wynik <- inner_join(post,pom,by=c("Id"="ParentId")) #laczenie ramek
  wynik <- inner_join(wynik,Posts,by=c("AcceptedAnswerId"="Id")) %>%
    select("Id","Title.x","MaxScore","Score.y") %>%
    mutate(Difference=MaxScore-Score.y) %>%  #kolumna z roznica
    arrange(desc(Difference)) %>% filter(Difference>50) %>% rename(Title=Title.x,AcceptedScore=Score.y)
  wynik$MaxScore <- as.integer(wynik$MaxScore) #odpowiedni output
  wynik$Difference <- as.integer(wynik$Difference)
  return(wynik)
}
dplyr::all_equal(df_sql_4(Posts),df_dplyr_4(Posts))

## [1] TRUE
```

4.4 DT

```
df_dt_4 <- function(x){
tbl <- as.data.table(Posts)[PostTypeId==2]
tb <- tbl[,.(MaxScore=max(Score)),by=ParentId] #agregacja po parentId
wynik <- merge(tb,tbl,by.x=c("ParentId","MaxScore"),by.y=c("ParentId","Score"))
wynik <- wynik[order(ParentId,Id)][!duplicated(ParentId)]
pos <- as.data.table(Posts)[PostTypeId==1] #odpowiednie wybranie danych
post <- as.data.table(Posts)
wynik <- merge(wynik,pos,by.x="ParentId",by.y="Id")[,.(MaxScore,ParentId,Title.y,AcceptedAnswerId.y)]
names(wynik)[names(wynik) == "ParentId"] = "Id" #zmiana nazwy jednej kolumny
wynik <- merge(wynik,post,by.x="AcceptedAnswerId.y",by.y="Id")[,.(Id,Title.y,MaxScore,Score)]
wynik <- wynik[,.((Difference=MaxScore-Score),Title.y,MaxScore,Score,Id)][V1>50][order(-V1)]
#wybieramy co trzeba
colnames(wynik) <- c("Difference","Title","MaxScore","AcceptedScore","Id")
return(wynik)
}

dplyr::all_equal(df_sql_4(Posts),df_dt_4(Posts))

## [1] TRUE
```

4.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_4(Posts),
                               dplyr=df_dplyr_4(Posts),
                               data.table=df_dt_4(Posts),
                               sql = df_sql_4(Posts),times=10)

## Unit: milliseconds
##        expr       min        lq      mean    median        uq      max
##        base 471.26275 481.92161 511.07402 505.39743 510.92871 629.2353
##       dplyr 196.77372 199.18798 211.57017 206.72323 225.36426 235.2425
##  data.table  67.62245  67.87531  91.61025  95.74571  99.26295 152.8045
##         sql 325.66064 326.67624 328.79016 329.09962 330.14763 331.6815
##  neval
##     10
##     10
##     10
##     10
```

5 Zadanie 5

5.1 SQL

```
df_sql_5<-function(x){
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")
}
```

Otrzymujemy pytania do których komentarze dodane przez samego pytającego miały sumarycznie najwyższą ocenę.

5.2 Base

```
df_base_5 <- function(x){
  x <- Comments[,c("PostId","UserId")] #kod nie wymaga w tym przypadku wielkich komentarzy
  tym <- aggregate(Comments[,"Score"],by=x,FUN=sum)
  colnames(tym) <- c("PostId","UserId","CommentsTotalScore")
  tym1 <- merge(tym,Posts,by.x=c("PostId","UserId"),by.y=c("Id","OwnerUserId"))
  tym1 <- tym1[tym1$PostTypeId==1,]
  tym1 <- tym1[order(tym1$CommentsTotalScore,decreasing = TRUE),]
  tym1 <- tym1[c(1:10),c("Title","CommentsTotalScore")]
  return(tym1)
}
dplyr::all_equal(df_sql_5(Posts),df_base_5(Posts))

## [1] TRUE
```

5.3 Dplyr

```
df_dplyr_5 <- function(x){
pom <- group_by(Comments,PostId,UserId) %>% summarise(x = sum(Score)) %>%
  rename(CommentsTotalScore = x)
pos <- filter(Posts,PostTypeId==1)
pom <- inner_join(pom,pos, by=c("PostId"="Id","UserId"="OwnerUserId")) %>%
  arrange(desc(CommentsTotalScore))
pom <- ungroup(pom) #pozbycie sie kolumny po ktorej grupowalismy
pom <- pom[1:10,] %>% select(Title,CommentsTotalScore)
return(pom)
}
dplyr::all_equal(df_sql_5(Posts),df_dplyr_5(Posts))

## [1] TRUE
```

5.4 DT

```
df_dt_5 <- function(x){
tb <- as.data.table(Comments)
pos <- as.data.table(Posts)[PostTypeId==1]
tb <- tb[,.(CommentsTotalScore=sum(Score)),by=c("PostId","UserId")]
tb <- merge(tb,pos,by.x=c("PostId","UserId"),by.y=c("Id","OwnerUserId"))
tb <- tb[,.(Title,CommentsTotalScore)][order(CommentsTotalScore,decreasing = TRUE)][1:10]
return(tb)
}

dplyr::all_equal(df_sql_5(Posts),df_dt_5(Posts))

## [1] TRUE
```

5.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_5(Posts),
                               dplyr=df_dplyr_5(Posts),
                               data.table=df_dt_5(Posts),
                               sql = df_sql_5(Posts),times=10)

## Unit: milliseconds
##        expr       min         lq       mean     median         uq
##        base 2438.2630 2495.44441 2578.58173 2531.39248 2665.18204
##       dplyr  235.9695  265.67208  278.18029  273.94557  287.69707
##  data.table   40.8855   41.83313   43.38918   42.23522   43.94918
##         sql  599.9890  601.68722  614.08697  604.59197  607.32518
##         max neval
##  2791.82005    10
##   349.32556    10
##    51.29104    10
##   672.37690    10
```

6 Zadanie 6

6.1 SQL

Informacje o użytkownikach, którym odznakę klasy pierwszej przyznano od 2 do 10 razy.

```
df_sql_6 <- function(x,y){
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")
}
```

6.2 Base

```
df_base_6 <- function(x,y){
tym <- Badges[Badges$Class==1,]
tym <- as.data.frame(tym$Name)
tym <- as.data.frame(table(tym)) #tabela kontyngencji jako ramka danych
tym <- tym[tym$Freq>=2,]
tym <- tym[tym$Freq<=10,]
tym1 <- Badges[Badges$Class==1,]
tym1 <- tym1[,c("UserId","Name")]
tym <- merge(tym,tym1,by.x="tym",by.y="Name")
wynik <- merge(tym,Users,by.x="UserId",by.y="Id") #odpowiedni merge
wynik <- wynik[,c("UserId","Reputation","DisplayName","Age","Location")]
wynik <- wynik[!duplicated(wynik),] #pozbycie sie duplikatow
colnames(wynik) <- c("Id","Reputation","DisplayName","Age","Location")
return(wynik)
}

dplyr::all_equal(df_sql_6(Users,Badges),df_base_6(Users,Badges))

## [1] TRUE
```

6.3 Dplyr

```
df_dplyr_6 <- function(x,y) {
x <- filter(Badges,Class==1) %>% group_by(Name) %>%
  summarise(count=n()) %>% filter(count>=2) %>%
  filter(count<=10) #grupowanie i ograniczenie count
y <- filter(Badges,Class==1, Name %in% x$Name) %>% select(Name,UserId)
z <- inner_join(y,Users,by=c("UserId"="Id")) %>%
  select(UserId,DisplayName,Reputation,Age,Location) #laceenie ramek
colnames(z) <- c("Id","DisplayName","Reputation","Age","Location")
z <- unique(z) #unikalne
return(z)
}
dplyr::all_equal(df_sql_6(Users,Badges),df_dplyr_6(Users,Badges))

## [1] TRUE
```

6.4 DT

```
df_dt_6 <- function(x,y){
tbl <- as.data.table(Badges)
tbl <- tbl[Class==1][,.(.N),by= .(Name)][N<=10][N>=2] #znow ladny syntax do laczenia polecen
pom <- as.data.table(Badges)[Class==1][Name %in%tbl$Name][,.(Name,UserId)]
pom <- merge(pom,Users,by.x="UserId",by.y="Id")
pom <- pom[,.(UserId,DisplayName,Reputation,Age,Location)][!duplicated(DisplayName)]
#wybranie odpowiednich kolumn
setnames(pom,old="UserId",new="Id")
return(pom)
}

dplyr::all_equal(df_sql_6(Users,Badges),df_dt_6(Users,Badges))

## [1] TRUE
```

6.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_6(Users,Badges),
                               data.table=df_dt_6(Users,Badges),
                               dplyr=df_dplyr_6(Users,Badges),
                               sql = df_sql_6(Users,Badges),times=10)

## Unit: milliseconds
##        expr       min        lq      mean    median        uq       max
##        base  11.59680  11.63181  12.08182  11.84501  12.31503  13.19261
##  data.table  16.67658  16.88538  17.20026  17.03064  17.40674  18.68102
##       dplyr  27.88332  28.07257  32.74459  31.07745  32.59981  52.88410
##         sql 291.45809 294.08060 298.26669 295.94331 298.78827 320.25213
##  neval
##     10
##     10
##     10
##     10
```

7 Zadanie 7

7.1 SQL

Otrzymujemy pytania, które przed 2016 otrzymały najwięcej UpVotes.

```
df_sql_7 <- function(x,y){sqldf("SELECT
Posts.Title,
VotesByAge2.OldVotes
FROM Posts
JOIN (
SELECT
PostId,
MAX(CASE WHEN VoteDate = 'new' THEN Total ELSE 0 END) NewVotes,
MAX(CASE WHEN VoteDate = 'old' THEN Total ELSE 0 END) OldVotes,
SUM(Total) AS Votes
FROM (
SELECT
PostId,
CASE STRFTIME('%Y', CreationDate)
WHEN '2017' THEN 'new'
WHEN '2016' THEN 'new'
ELSE 'old'
END VoteDate,
COUNT(*) AS Total
FROM Votes
WHERE VoteTypeId=2
GROUP BY PostId, VoteDate
) AS VotesByAge
GROUP BY VotesByAge.PostId
HAVING NewVotes=0
) AS VotesByAge2 ON VotesByAge2.PostId=Posts.ID
WHERE Posts.PostTypeId=1
ORDER BY VotesByAge2.OldVotes DESC
LIMIT 10")
}
```

7.2 Base

```
df_base_7 <- function(x,y){
  Votes2 <- Votes[Votes$VoteTypeId==2,]
  Year <- as.data.frame(substring(Votes2$CreationDate, 1, 4))
  OLdNew <- Year
  OLdNew <- as.data.frame(ifelse(Year=="2017" | Year=="2016", "new", "old"))
  VotesByAge0 <- cbind(Votes2[, c("PostId")], OLdNew)
  colnames(VotesByAge0) <- c("PostId", "VoteDate")
  VotesByAge <- as.data.frame(table(VotesByAge0))
  VotesByAge <- VotesByAge[VotesByAge$Freq>0,] #wybranie tych powyzej 0
  VotesByAge <- VotesByAge[order(VotesByAge$PostId),]
  rownames(VotesByAge) <- NULL
  colnames(VotesByAge)[3] <- "Total" #dobre nazwy kolumn
  VotesByAge["PostId"] <- as.numeric(levels(VotesByAge$PostId))[VotesByAge$PostId] #odpowiednia postac
  VotesByAge["VoteDate"] <- as.character(levels(VotesByAge$VoteDate))[VotesByAge$VoteDate]
  Votes3 <- aggregate(VotesByAge$Total, VotesByAge["PostId"], sum)[2]
  VotesByAgeOld <- VotesByAge[VotesByAge$VoteDate=="old",]
  OldMaxes <- aggregate(VotesByAgeOld[,"Total"], VotesByAgeOld["PostId"], max)
  colnames(OldMaxes)[2] <- "OldVotes"
  VotesByAgeNew <- VotesByAge[VotesByAge$VoteDate=="new",] #wybranie odpowiednich danych
  NewMaxes <- aggregate(VotesByAgeNew[, "Total"], VotesByAgeNew["PostId"], max)
  colnames(NewMaxes)[2] <- "NewVotes"
  Maxes <- merge(NewMaxes, OldMaxes, by.x ="PostId", by.y="PostId", T, T)
  Maxes <- Maxes[order(Maxes$PostId),]
  Maxes <- cbind(Maxes, Votes3) #laczymy wyniki
  colnames(Maxes)[4] <- "Votes"
  Maxes[is.na(Maxes$NewVotes), "NewVotes"] <- 0 #pozbywamy sie NA zamieniejac je na 0
  VotesByAge2 <- Maxes[Maxes$NewVotes==0,]
  Posts1 <- Posts[Posts$PostTypeId==1,]
  datas<- merge(Posts1, VotesByAge2, by.x="Id", by.y="PostId")
  out <- datas[, c("Title", "OldVotes")]
  out <- out[order(out$OldVotes, decreasing = T),]
  out <- head(out, 10)
  return(out)
}
dplyr::all_equal(df_sql_7(),df_base_7())

## [1] TRUE
```

7.3 Dplyr

```r
df_dplyr_7 <- function(x,y){
Votes2 <- filter(Votes, VoteTypeId==2)
Year <- as.data.frame(substring(Votes2$CreationDate, 1, 4))
VoteDate <- ifelse(Year=="2017" | Year=="2016", "new", "old")
#case_when jest znacznie wolniejsze
mutate(Votes2, VoteDate) %>%
  group_by(PostId, VoteDate)%>%
  summarise(Total =n())  -> VotesByAge #suma po odpowiednich grupach
VotesByAge%>%
  filter(VoteDate=="old") %>%
  group_by(PostId) %>%
  mutate(OldVotes = max(Total)) -> dt1 #nowa kolumna
VotesByAge%>%
  filter(VoteDate=="new") %>%
  group_by(PostId) %>%  #grupujemy
  mutate(NewVotes = max(Total)) -> dt2
merge(dt1, dt2, by="PostId", all=T)%>%
  select(PostId, NewVotes, OldVotes) -> pom #wybranie tego co wazne
pom%>%
  mutate(NewVotes = replace(NewVotes, is.na(NewVotes), 0)) %>%
  mutate(OldVotes = replace(OldVotes, is.na(OldVotes), 0)) -> pomoc
pomoc %>%
  mutate(Votes = OldVotes) %>%
  filter(NewVotes ==0) -> VT
out <- filter(Posts, PostTypeId ==1) %>%
  inner_join(VT, by=c("Id" = "PostId"))%>%
  select(Title, OldVotes) %>%
  arrange(desc(OldVotes)) %>% #odpowiednio sortujemy
  head(10)
out$OldVotes <- as.integer(out$OldVotes)
return(out)
}
dplyr::all_equal(df_sql_7(Posts,Votes),df_dplyr_7(Posts,Votes))

## [1] TRUE
```

7.4 DT

```r
df_dt_7 <- function(x,y){
Votes <- as.data.table(Votes)
Posts <- as.data.table(Posts)
Votes2 <- Votes[VoteTypeId==2]
Year <- setDT(data.frame(substring(Votes2$CreationDate, 1, 4)))
colnames(Year) <- "colYear"#ustawienie nazwy
VoteDate <- Year[colYear%in% c("2017", "2016"), OldNew := "New"] #wybranie odpowiednio new i old
VoteDate <- Year[!colYear%in% c("2017", "2016"), OldNew := "Old"]
VotesByAge <- cbind(Votes2, VoteDate = VoteDate[, OldNew])[, .N, keyby=.(PostId, VoteDate)]
VotesByAge[, "Total" := sum(N), PostId][VoteDate=="New", "NewVotes" := max(N), PostId][VoteDate=="Old", "OldVotes"
][Total==OldVotes, .(NewVotes, OldVotes, Total), PostId ] -> VotesByAge2 #wygodne ustawienie zmiennej
setkey(VotesByAge2, PostId)
setkey(Posts, Id)
data <- Posts[VotesByAge2, nomatch=0]
out <- data[PostTypeId==1, .(Title, OldVotes)] #wybranie odpowiednich
out <- out[order(-OldVotes)][1:10]
return(out)
}
dplyr::all_equal(df_sql_7(Posts,Votes),df_dt_7(Posts,Votes))

## [1] TRUE
```

7.5 Benchmark

```
microbenchmark::microbenchmark(base=df_base_7(Posts,Votes),
                               data.table=df_dt_7(Posts,Votes),
                               dplyr=df_dplyr_7(Posts,Votes),
                               sql = df_sql_7(Posts,Votes),times=10)

## Unit: milliseconds
##        expr       min         lq       mean     median         uq
##        base 2111.0692  2186.7692  2423.7930  2351.1971  2667.2067
##  data.table  294.6868   358.2996   507.4591   531.9952   560.8997
##       dplyr 8841.2082 10012.9089 11700.7655 11017.2935 12289.1902
##         sql 1330.4123  1392.5078  1572.8476  1487.0263  1761.7492
##         max neval
##   2912.8770    10
##    831.4758    10
##  19286.1066    10
##   2027.5478    10
```

Niejako podsumowując prędkość dzialania data.table, dplyr oraz bazowego R nasuwa się dość jasny obraz prędkości ich działąnia. Data.table jest w zdecydowanej większości przypadków najszybszy, ale dplyr bije go na głowę pod względem łatwości w użyciu oraz prostocie kodu który się w nim pisze.