Praca Domowa 1

Jan Borowski

14 kwietnia 2019

Spis treści

1 Wstęp

W poniższym raporcie przedstawie wyniki mojej pracy nad zadaniami z pierwszej pracy domowej. Wszystkie zadania zamknięte są w funkcjach odpowiednio`df_base_i`,`df_dplyr_i`,`df_data.table_i` gdzie i to numer zdania. Wszystkie funkcje zwracają ramki danych oraz nie przyjmują żadnych argumentów uznałem, że w tym konkretnym wypadku zmienne globalne nie będą problemem ponieważ używam tylko 7 wczytanych ramek danych . Zgodność ramek z zapytaniem SQLa będzie sprawdzana przy użyciu funkcij `all_equal` z pakietu dplyr pod każdą z pokazywanych funkcji.Szybkość wykonywania się kodu będzie sprawdzana z użyciem funkcji `microbenchmark` .Ponadto globalnie ustawiona będzie opcja `stringsAsFactors=FALSE`.
Wczytywanie danych i pakietów:

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

2 Zadanie 1

2.1 SQL

```
df_sql_1 <- function(){
x<- sqldf::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")
 return(x)}
df_sql1 <- df_sql_1()
#ramka do porównań
```

Interpretacja zapytania:
Otrzymujemy 10 osób z największą ilością otrzymanych gwiazdek oraz ich najbardziej lajkowanym pytaniem.

2.2 Base

```
df_base_1 <- function(){
tmp <- Posts[Posts[,"PostTypeId"]==1,]
# Tworzę tymczasową ramkę danych z PostTypeId=1
left <-  aggregate(tmp$FavoriteCount,
            by=list("OwnerUserId"=tmp$OwnerUserId)
            ,FUN= function(x){sum(x,na.rm = TRUE)})
# Przygotowuję kolumnę z sumą
colnames(left)[2] <-"FavoriteTotal"
righ <-  aggregate(tmp$FavoriteCount,
            by=list("OwnerUserId"=tmp$OwnerUserId)
            ,FUN=function(x){max(x,na.rm = TRUE,warn=-10)})
# Przygotowuje kolumnę z maksami
colnames(righ)[2] <- "MostFavoriteQuestionLikes"
title_max<- merge(tmp[,c("Title","OwnerUserId","FavoriteCount")],
            righ,by.x=c("OwnerUserId","FavoriteCount"),
            by.y=c("OwnerUserId","MostFavoriteQuestionLikes"))
# łączę ze sobą przygotowane kolumny
colnames(title_max)[2] <- "MostFavoriteQuestionLikes"
title_max_sum <- merge(title_max,left,by="OwnerUserId")
wynik <- merge(Users[,c("Age","DisplayName","Location","Id")]
            ,title_max_sum,by.x="Id",by.y="OwnerUserId")
#  łączę z wybranami kolumnami z Users
wynik <- wynik[,c("DisplayName","Age","Location","FavoriteTotal"
```

```
              ,"Title","MostFavoriteQuestionLikes")]
wynik <- head(wynik[rev(order(wynik[["FavoriteTotal"]])),],10)
# sortuje i wybieram pierwszych 10
colnames(wynik)[5] <- "MostFavoriteQuestion"
row.names(wynik) <- 1:10
return(wynik)}
all_equal(df_sql1,df_base_1())

## [1] TRUE
```

Przy kolumnie z maxami wystąpią ostrzeżenia o zmianie NA na -Inf są to wiersze gdzie nie ma obserwacji w kolumnie FavoriteCount ,ale ponieważ i tak na końcu wybieram 10 największych (Czyli napewno zawierających obserwacje). Nie ma potrzeby zmieniać -Inf na NA.

2.3 Dplyr

```
df_dplyr_1 <- function(){
tmp <- filter(Posts,PostTypeId==1)
max_sum <- select(tmp,FavoriteCount,OwnerUserId,Title) %>%
          group_by(OwnerUserId)%>%
          summarise(FavoriteTotal = sum(FavoriteCount, na.rm = TRUE)
          ,MostFavoriteQuestionLikes=as.integer(max(FavoriteCount,
                                                    na.rm=TRUE,warn=-10)))
# Tworzę ramkę zawierajacą max i sumę z  kolumny FavoriteCount
# pogrupowanej po OwnerUserId
max_sum_title <- inner_join(select(tmp,Title,OwnerUserId,
                                   FavoriteCount)
            ,max_sum,by=c("OwnerUserId"="OwnerUserId",
            "FavoriteCount"="MostFavoriteQuestionLikes"))
#Dodaje do tego kolumne Title łącząc po OwnerUserID i
# FavoriteCount
max_sum_title <- rename(max_sum_title,MostFavoriteQuestionLikes
                        =FavoriteCount)
# Zmieniam nazwy kolumn
wynik <- inner_join(select(Users,Age,DisplayName,Location,Id),
            max_sum_title,by=c("Id"="OwnerUserId")) %>%
            arrange(desc(FavoriteTotal))%>% slice(1:10) %>%
            select(DisplayName,Age,Location,
                   FavoriteTotal,
            Title,MostFavoriteQuestionLikes)
# Łączę wcześnjej stworzoną ramkę z wybranymi kolumnami z Users
# Sortuję i wybieram pierwsze 10
wynik <- rename(wynik,MostFavoriteQuestion=Title)
return(wynik)}
wynik <- df_dplyr_1()
all_equal(wynik,df_sql1)

## [1] TRUE
```

2.4 Data.Table

```
df_data.table_1 <- function(){
Posts.dt <- data.table(Posts)
Users.dt <- data.table(Users)
# Rzutowanie danych na typ data.table
# a właściwie tworzenie obiektu tego typu
max_sum_title.dt <- Posts.dt[PostTypeId==1,
   .(MostFavoriteQuestion=Title[which.max(FavoriteCount)],
    MostFavoriteQuestionLikes=as.double(
    max(FavoriteCount,na.rm = TRUE,warn=-10))
  ,FavoriteTotal=as.integer(sum(FavoriteCount,na.rm = TRUE)))
  ,by=OwnerUserId]
# tworzę tabele zawierająca max i sumę z kolumny
# FavoriteCount oraz odpowiadający maxsowi rekord
# z kolumny Title
wynik.dt <- merge(max_sum_title.dt,
                  Users.dt[,.(Id,Age,DisplayName,Location)],
                  by.x="OwnerUserId",by.y="Id")
# Łączę tę tablę z wybranymi kolumnami z Votes
wynik.dt <-wynik.dt[order(-FavoriteTotal),
                    .(DisplayName,Age,Location,FavoriteTotal,
                      MostFavoriteQuestion,
                      MostFavoriteQuestionLikes=
                  as.integer(MostFavoriteQuestionLikes))][1:10]
# odpowiednio sortuję i wybieram potrzebne kolumny
# biorę 10 pierwszych wierszy

return(wynik.dt)}
all_equal(df_sql1,df_data.table_1())

## [1] TRUE
```

Rzutowanie typów danych odbywa sie w funkcji ponieważ doszedłem do wniosku ,że jest to niezbędne przy korzystaniu z data.table i powinno być uwzględnione w benchmarku skoro porównujemy czasy jednego wykonania.

2.5 Benchamrk

```
a <- capture.output(microbenchmark::microbenchmark(times=100,
  sql_1=df_sql_1(),
  base_1=df_base_1(),
  dplyr_1=df_dplyr_1(),
  data.table_1=df_data.table_1()))

saveRDS(a,file="bench1.rds")
x <- readRDS(file="bench1.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr       min        lq     mean    median        uq      max
##         sql_1 260.45312 265.21781 285.7546 272.36599 294.54648 431.5740
##        base_1 294.46102 322.49246 333.8665 326.65752 338.64159 473.2196
```

```
##       dplyr_1  90.73642 120.78239 129.5014 125.28237 143.54845 250.7126
##  data.table_1  44.94067  55.59512  77.9214  79.87675  85.46156 189.0339
##  neval
##    100
##    100
##    100
##    100
```

3 Zadanie 2

3.1 SQL

```
df_sql_2 <- function(){
x<- sqldf::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")
 return(x)}
df_sql2 <- df_sql_2()
#ramka do porównań
```

Interpretacja zapytania:
W efekcie otrzymujemy Id i Treść 10 pytań na które jest najwięcej pozytywnie ocenionych odpowiedzi.

3.2 Base

```
df_base_2 <- function(){
Posts2 <- as.data.frame(table("ParentID"=Posts[Posts[
          ,"PostTypeId"]==2 &
          Posts[,"Score"]>0,"ParentId"]),
          responseName = "PositiveAnswerCount")
          # wybieram i tworzę Posts2 używając funkcji table
wynik <- merge(as.data.frame(Posts[,c("Id","Title")])
        ,Posts2,all.x =TRUE,all.y=FALSE,by.x="Id",by.y="ParentID")
          # łączę swoją table z tabelą Posts2
          # (można użyć inner joina)
wynik <- wynik[!(is.na(wynik[["PositiveAnswerCount"]])),]
          # Usuwam NA
d <- wynik[rev(order(wynik[["Id"]])),]# najpierw sortuje po ID
d <- d[rev(order(d[["PositiveAnswerCount"]])),]
# potem po PositveAnswerCount
q <- slice(d,c(1:10))# Biorę 10 pierwszych wierszy
row.names(q) <- c(1:10)# nazwy wierszy
q}
all_equal(df_sql2,df_base_2())

## [1] TRUE
```

3.3 Dplyr

```
df_dplyr_2 <- function(){
Posts1 <- (select(Posts,c("Id","Title")))
#Wybieram potrzebne kolumny z Posts
Posts2 <-(select(filter(Posts,PostTypeId==2&Score>0),ParentId))
# wybieram odpowiednie wiersze
Posts2 <-  Posts2 %>% group_by(ParentId) %>%
  count(name = "PositiveAnswerCount")
# zliczam je
wynik <- left_join(Posts1,Posts2,by =c("Id"="ParentId"))
wynik <- arrange(wynik,Id)
wynik<- arrange(wynik,desc(PositiveAnswerCount))
# odpowiednio sortuje i wybieram 10 pierwszych
wynik <- slice(wynik,1:10)
wynik}
all_equal(df_sql2,df_dplyr_2())

## [1] TRUE
```

Tym razem usuwanie NA nie jest potrzebne ponieważ funkcja `desc` zostawia je na końcu tablicy a mnie interesuje 10 pierwszych wierszy.

3.4 Data.Table

```
df_data.table_2 <- function(){
Posts.dt <- data.table(Posts)
# Rzutowanie na typ data.table
# data.table szybsze niż as.data.table
Posts2.dt <- (Posts.dt[PostTypeId==2 & Score>0,
               .(PositiveAnswerCount=.N),by=ParentId])
# wybranie odpowiednich wierszy i ich zliczenie
wynik <- merge(Posts.dt[,list(Id,Title)],Posts2.dt,
              by.x="Id",by.y = "ParentId",
              all.x = TRUE,all.y = FALSE)
# łączenie ramek danych Posts i Posts2 po kolumnach
# odpowiednio Id i ParentId
wynik <- wynik[order(Id)][order(-PositiveAnswerCount)]
wynik <- wynik[1:10]
wynik}
all_equal(df_data.table_2(),df_sql2)

## [1] TRUE
```

Ponownie jak w poprzednim zadaniu rzutowanie danych odbywa się w funkcji.

3.5 Benchamrk

```
 w<- capture.output(microbenchmark::microbenchmark(times=100,
   sql_2=df_sql_2(),
```

```
   base_2=df_base_2(),
   dplyr_2=df_dplyr_2(),
   data.table_2=df_data.table_2()))

saveRDS(w,file="bench2.rds")
x <- readRDS(file="bench2.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr       min        lq      mean    median        uq      max
##         sql_2 192.01692 202.17930 227.60977 219.49642 247.54228 319.0551
##        base_2 158.17213 187.61027 214.25191 206.05007 241.07753 405.1724
##       dplyr_2  73.60376  77.43488 100.17108  91.53820 110.40772 417.1631
##  data.table_2  21.84558  23.31464  43.35463  31.83325  55.75937 217.9157
##  neval
##    100
##    100
##    100
##    100
```

4 Zadanie 3

4.1 SQL

```
df_sql_3 <- function(){
x<- sqldf::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")
 return(x)}
df_sql3 <- df_sql_3()
#ramka do porównań
```

Interpretacja zapytania:
Otrzymujemy pytania, które otrzymały najwiecej UpVotes(głosy pozytywne) w danym roku.

4.2 Base

```
df_base_3 <- function(){
d <- substring(Votes[Votes[,"VoteTypeId"]==2,]$CreationDate,1,4)
# Tworzenie kolumny Year
UpVotesPerYer <- as.data.frame(cbind("PostId"=Votes[Votes[,
                "VoteTypeId"]==2,"PostId"],"Year"=d),
                stringsAsFactors = FALSE)
# tworzenie ramki danych z Year zamiast z CreationDate
# oraz filtrowanie żeby VoteTypeId=2
UpVotesPerYer <- as.data.frame(table(UpVotesPerYer),
                responseName = "Count",stringsAsFactors = FALSE)
# zliczanie danych z ramki
UpVotesPerYer <- UpVotesPerYer[UpVotesPerYer[,"Count"]>0,][,
                c("PostId","Count","Year")]
# filtrowanie  i wybieranie potrzebnych kolumn
UpVotesPerYer <- cbind(("PostId"=as.integer(UpVotesPerYer$PostId)),
                UpVotesPerYer[,c("Count","Year")])
# tworzę ostatecznie ramkę danych UpVotesPerYer
colnames(UpVotesPerYer) <- c("PostId","Count","Year")
join <- merge(Posts,UpVotesPerYer,all=TRUE,by.x="Id",by.y="PostId")
# łączę powstałą ramkę z Posts
```

```
join <- join[join[,"PostTypeId"]==1,c("Title","Year","Count")]
join<- join[rev(order(join$Count)),]
#Odpowiedni filtruje i sortuje
lewa <- aggregate(join$Count,by=list("Year"=join$Year),FUN = max)
# liczę maksy w kolumnie Count pogrupowaniej po Year
prawa <-join[,c("Title","Year")]
prawa <- aggregate(join$Title,by=list("Year"=join$Year),
                FUN = function(x){x[1]})
# więcej o tym pod wstawką
wynik <- cbind(prawa,lewa)
colnames(wynik) <- c("Year","Title","NWM","Count")
wynik <- wynik[,c("Year","Title","Count")]
wynik }
all_equal(df_sql3,df_base_3())

## [1] TRUE
```

Zamiast tego `aggregate` można użyc `merge` po kolumnie Year i Title. Nie wymaga to wtedy tylu sortowań po drodzę.

4.3 Dplyr

```
df_dplyr_3 <- function(){
Year <- filter(Votes,VoteTypeId==2) %>% pull(CreationDate)
Year <- substring(Year,1 ,4)
# Tworzenie kolumny Year

Year <- as.data.frame(Year)
UpVotesPerYer <- filter(Votes,VoteTypeId==2)%>%
        select(PostId)%>% bind_cols(Year)%>%
        group_by(PostId,Year) %>% count(name="Count")
# tworzę ramkę UpVotesPerYer odrazu zliczając
# dane pogrupowane po PostId
wynik <- full_join(Posts,UpVotesPerYer,
        by=c("Id"="PostId"))
# łączę utworzoną tablelę z ramką Posts
prawy <- filter(wynik,PostTypeId==1)%>%
        select(c(Title,Year,Count)) %>% arrange(desc(Count)) %>%
        group_by(Year,.drop=TRUE) %>% summarise(Title=Title[1],
        Count=as.integer(max(Count)))
# filtruje po PostTypeId=1 wybieram potrzebne kolumny
# tworzę kolumnę z Title i liczę maxa z kolumny Count
# obie kolumny są grupowane po Year (wybieram Title z maksymalnym Count)
wynik <- select(prawy,c(Title,Year,Count))
wynik <- slice(wynik,c(2:8))
# wybieram w ten sposób ponieważ pierwszy wiersz zawiera NA
wynik}
all_equal(df_sql3,df_dplyr_3())

## [1] TRUE
```

Ponownie do wybrania właściwego Tytułu zamiast `summarise` można użyć `inner_join` po kolumnach Year i Title.

4.4 Data.Table

```
df_data.table_3 <-function(){
Votes.dt <- data.table(Votes)
Votes.dt <- Votes.dt[VoteTypeId==2,]
Posts.dt <- data.table(Posts)
#Rzutowanie danych na typ data.table
# oraz ich filtrowanie
Year <- Votes.dt[,CreationDate]
Year <- substring(Votes.dt[,CreationDate],1,4)
# tworzenie kolumny Year
UpVotesPerYear.dt <- data.table(PostId=Votes.dt[,PostId],Year=Year)
UpVotesPerYear.dt<- UpVotesPerYear.dt[,.(Count=.N),by=.(PostId,Year)]
# tworzę tablice UpVotesPerYear
wynik.dt <-merge(Posts.dt,UpVotesPerYear.dt,
           all=TRUE,by.x="Id",by.y="PostId")
# łączę powstałą ramkę z Posts
wynik.dt <- wynik.dt[PostTypeId==1,]
wynik.dt <- wynik.dt[order(-Count),.(Title=(Title)[1],
            Count=max(Count)),by=Year]
# Wybieram tytuły dla grup w kolumnie Year
# oraz liczę max od Count dla grupy Year
wynik.dt <- wynik.dt[order(Year),.(Title,Year,Count)]
wynik.dt <- wynik.dt[1:7]
# Sortuje i wybieram 7 zamiast usuwać NA
wynik.dt}
all_equal(df_sql3,df_data.table_3())

## [1] TRUE
```

Używam funkcij `substring` zamiast funkcji wbudowanych w pakiet data.table ponieważ jest ona kilkukronie szybsza.

4.5 Benchamrk

```
 e<- capture.output(microbenchmark::microbenchmark(times=100,
  sql_3=df_sql_3(),
  base_3=df_base_3(),
  dplyr_3=df_dplyr_3(),
  data.table_3=df_data.table_3()))

saveRDS(e,file="bench3.rds")
x <- readRDS(file="bench3.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr      min       lq      mean    median        uq       max
```

```
##         sql_3 991.3631 1014.7122 1103.6242 1108.4104 1181.3788 1315.4081
##        base_3 931.4454 1052.2417 1176.1500 1146.6353 1269.4202 1799.2788
##       dplyr_3 443.4355  504.3792  588.4936  555.1026  643.1683 1052.0050
##  data.table_3 174.6013  228.4478  280.8423  256.0609  321.3706  603.2349
##  neval
##    100
##    100
##    100
##    100
```

5 Zadanie 4

5.1 SQL

```
df_sql_4 <- function(){
x<- sqldf::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")
 return(x)}
df_sql4 <- df_sql_4()
#ramka do porównań
```

Interpretacja zapytania:
Otrzymujemy pytania dla których różnica pomiędzy najwyżej ocenioną odpowiedzią a odpowiedziami zaakceptowanymi wynosiła więcej niż 50.

5.2 Base

```
df_base_4 <- function(){
w <- Posts[Posts[,"PostTypeId"]==2,]
BestAnswer1 <- aggregate(w$Score,by=list("ParentId"=w$ParentId),max)
# Tworzę ramkę zawierającą max od Score pogrupowane po Year
BestAnswer <- merge(BestAnswer1,w[,c("ParentId","Id","Score")],by.x=c("ParentId","x"),
                    by.y=c("ParentId","Score"),all=FALSE)
# łączę wcześneij stworzone kolumny z ramką Posts
BestAnswer <- BestAnswer[!duplicated(BestAnswer$ParentId),]
# usuwam powstające powtórki
colnames(BestAnswer)[2] <- "MaxScore"
Question <-  (Posts[Posts[,"PostTypeId"]==1,])
# filtrowanie po PostTypeId=1
Przejscie <- merge(Question,BestAnswer,all=TRUE,by.x="Id",by.y="ParentId")
# łączenie ramek Posts i BestAnswer full_join
wynik <- merge(Przejscie,Posts[,c("Score","Id")],all.x=FALSE,all.y=TRUE,
```

```
                by.x="AcceptedAnswerId",by.y="Id",sort=TRUE)
# łączę z ramką Posts right_join
diff <-  wynik$MaxScore -Posts$Score
# tworzenie kolumny Difference
wynik <- cbind(wynik,diff)
wynik <- wynik[wynik[,"diff"]>50,c("Id","Title","MaxScore","Score.y","diff")]
# wybieranie właściwych kolum i wierszy z Difference większym od 50
wynik <- na.omit(wynik)
# usuwanie Na
wynik <- wynik[rev(order(wynik[,"diff"])),]
# sortowanie
colnames(wynik) <- c("Id","Title","MaxScore","AcceptedScore","Difference")
row.names(wynik) <- c(1:length(wynik[["Id"]]))
# właściwe nazwanie kolumn i wierszy
wynik}
all_equal(df_base_4(),df_sql4)

## [1] TRUE
```

5.3 Dplyr

```
df_dplyr_4 <- function(){
BestAnswer <- select(filter(Posts,PostTypeId==2),c(Id,ParentId,Score))%>%
          group_by(ParentId) %>% arrange(-Score,.by_group=TRUE) %>%
          summarise("Id"=Id[1],"MaxScore"=(max(Score)))%>%
          select(Id,ParentId,MaxScore)
#Tworzę ramkę BestAnswers
Question <-  filter(Posts,PostTypeId==1)
# tworzę ramkę Question
join1 <- inner_join(Question,BestAnswer,by=c("Id"="ParentId"))
# łączę ramki Question i BestAnswer
join2 <- right_join(join1,Posts,by=c("AcceptedAnswerId"="Id"))
# łączę powstałą ramkę z Posts
wynik <- select(join2,Id,Title.x,MaxScore)
wynik <- rename(wynik,Id=Id,Title=Title.x,MaxScore=MaxScore)
# wybieram i odpowiednio nazywam kolumny
wynik <- bind_cols(wynik,"AcceptedScore"=Posts$Score,
          "Difference"=(wynik$MaxScore-Posts$Score))
# tworzę kolunę Difference
wynik <- filter(wynik,Difference>50) %>% arrange(-Difference)
wynik}
all_equal(df_sql4,df_dplyr_4())

## [1] TRUE
```

5.4 Data.Table

```r
df_data.table_4 <- function(){
Posts.dt <- data.table(Posts)
# Rzutowanie na typ danych data.table
BestAnswer.dt <- Posts.dt[order(-Score)][PostTypeId==2,.(Id[1],max(Score)),by=ParentId]
BestAnswer.dt <- BestAnswer.dt[,.(Id=V1,ParentId,MaxScore=V2)][order(ParentId)]
# tworzenie tabeli BestAnswers
Question.dt <- Posts.dt[PostTypeId==1,]
# tworzenie tabeli Question
join1.dt <- merge(Question.dt,BestAnswer.dt,all.x=TRUE,
                  all.y=TRUE,by.x="Id",by.y="ParentId")
# łączenie tabeli Question i BestAnswer
join2.dt <- merge(join1.dt,Posts.dt[,.(Id)],all.y=TRUE,
                  by.x="AcceptedAnswerId",by.y="Id")
# łączenie z tabelą Posts
wynik.dt <- data.table(Id=join2.dt[,Id],Title=join2.dt[,Title],
                       MaxScore=join2.dt[,MaxScore],
                       AcceptedScore=Posts.dt[,Score],
                       Difference=(join2.dt[,MaxScore]-Posts.dt[,Score]))
# dodawanie kolumny Difference
wynik.dt <- wynik.dt[Difference>50][order(-Difference)]
wynik.dt}

all_equal(df_sql4,df_data.table_4())

## [1] TRUE
```

5.5 Benchmark

```r
 r<- capture.output(microbenchmark::microbenchmark(times=100,
  sql_4=df_sql_4(),
  base_4=df_base_4(),
  dplyr_4=df_dplyr_4(),
  data.table_4=df_data.table_4()))

saveRDS(r,file="bench4.rds")
x <- readRDS(file="bench4.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr       min        lq     mean   median       uq       max
##         sql_4 242.69743 249.01532 269.2635 255.6448 280.8345  350.7182
##        base_4 572.40145 618.41922 691.6354 658.0494 755.8071 1083.8664
##       dplyr_4 292.22439 331.63083 369.6672 349.8464 388.4436  564.4908
##  data.table_4  59.95729  96.13303 123.8721 104.8204 136.7063  325.3721
##  neval
##    100
##    100
##    100
##    100
```

6 Zadanie 5

6.1 SQL

```
df_sql_5 <- function(){
x<- sqldf::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")
 return(x)}
df_sql5 <- df_sql_5()
#ramka do porównań
```

Interpretacja zapytania:
Otrzymujemy 10 pytań których autorzy otrzymali najwięcej głosów za własne komentarze pod nimi.

6.2 Base

```
df_base_5 <- function(){
Posts2 <- Comments[,c("PostId","UserId","Score")]
# Wybieram kolumny z ramki Coments
Posts2 <- aggregate(Posts2$Score,by=list("PostId"=Posts2$PostId
                                         ,"UserId"=Posts2$UserId),sum)
# Liczę sumę w kolumnie Score grupując po PostId,UserId
colnames(Posts2) <- c("PostId","UserId","CommentsTotalScore")
# zmieniam nazwy kolumn
Posts1 <- Posts[Posts[,"PostTypeId"] ==1,c("Title","Id","OwnerUserId")]
# filtruje po PostTypeId =1
wynik <- merge(Posts1,Posts2,by.x=c("Id","OwnerUserId"),by.y=c("PostId"
                                                               ,"UserId"))
# łaczę przygotwoną ramkę z wybranymi kolumnami z Posts
wynik <- na.omit(wynik)
# usuwam NA
wynik <- wynik[,c("Title","CommentsTotalScore")]
wynik <- wynik[rev(order(wynik[["CommentsTotalScore"]])),]
# sortuje
wynik <- head(wynik,10)
rownames(wynik) <- c(1:10)
wynik}
```

```
all_equal(df_base_5(),df_sql5)

## [1] TRUE
```

6.3 Dplyr

```
df_dplyr_5 <- function(){
CmtTotScr <- select(Comments,c("PostId","UserId","Score")) %>%
  group_by(PostId,UserId) %>%
  summarise("CommentsTotalScore" = sum (Score))
# Liczę sumę z Score   grupując po PostId,UserId
Posts1 <- select(filter(Posts,PostTypeId==1),c("Title","Id","OwnerUserId"))
# wybieram i filtruje potrzebne kolumny z Posts
wynik <- left_join(Posts1,CmtTotScr,by=c("Id"="PostId","OwnerUserId"="UserId")) %>%
  select(Title,CommentsTotalScore) %>% arrange(desc(CommentsTotalScore))
# łączę wcześniej przygotowną ramkę z CmtTotScr
wynik <- slice(wynik,1:10)
wynik}
all_equal(df_sql5,df_dplyr_5())

## [1] TRUE
```

6.4 Data.Table

```
df_data.table_5 <- function(){
Comments.dt <- data.table(Comments)
Posts.dt <-  data.table(Posts)
# Rzutowanie danych na typ data.table
CmtTotScr.dt <- Comments.dt[,CommentsTotalScore:=.(sum(Score)),keyby=.(PostId,UserId)][,
                                                                    .(PostId,UserId,CommentsTotalScore)]
# liczenie sumy od Score grupując po PostId,UserId
wynik.dt <- merge(Posts.dt[PostTypeId==1,.(Title,Id,OwnerUserId)],CmtTotScr.dt,
                  by.x =c("Id","OwnerUserId"),by.y=c("PostId","UserId"),
                  all.x = TRUE,all.y=FALSE)
# robię righ join z wybranymi kolumnami i wierszami z Posts
wynik.dt <- wynik.dt[!(duplicated(wynik.dt))]
# usuwanie duplikatów
wynik.dt <- wynik.dt[,.(Title,CommentsTotalScore)][order(-CommentsTotalScore)][1:10]
# odpowidnie sortowanie
# wybór kolumn i 10 pierwszych wierszy
wynik.dt}
all_equal(df_sql5,df_data.table_5())

## [1] TRUE
```

6.5 Benchmark

```
 t<- capture.output(microbenchmark::microbenchmark(times=100,
  sql_5=df_sql_5(),
  base_5=df_base_5(),
  dplyr_5=df_dplyr_5(),
  data.table_5=df_data.table_5()))

saveRDS(t,file="bench5.rds")
x <- readRDS(file="bench5.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr       min        lq      mean    median        uq       max
##         sql_5  444.1690  448.9194  469.9707  451.3463  479.8041  636.2288
##        base_5 1756.1122 1812.3031 1923.7458 1871.0665 1978.1599 2498.4008
##       dplyr_5  183.0042  221.4068  263.0130  249.9233  278.6287  623.3894
##  data.table_5  142.5496  176.7755  204.8946  188.4030  216.6870  373.4274
##  neval
##    100
##    100
##    100
##    100
```

7 Zadanie 6

7.1 SQL

```
df_sql_6 <- function(){
x<- sqldf::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")
 return(x)}
df_sql6 <- df_sql_6()
#ramka do porównań
```

Interpretacja zapytania:
Otrzymujemy dane o wszystkich użytkownikach , którzy mają od 2 do 10 złotych odznak.

7.2 Base

```
df_base_6 <- function(){
inner_tmp <-  (Badges[Badges[,"Class"]==1,"Name"])
inner_tmp<- (table(inner_tmp))
# zliczam w kolumnie Name dla Class =1
inner_tmp <- as.data.frame(inner_tmp[2<=inner_tmp& inner_tmp<=10],StringAsFactor=FALSE)
# rzutuję na typ data.frame i filtruje
colnames(inner_tmp) <- "Name"
ValuableBadges <- Badges[(Badges$Name%in% inner_tmp$Name)& Badges$Class==1,
                         c("Name","UserId")]
# wybieram kolumny Name, UserId z Badges
# wiersz biorę wtedy kiedy Name znajduje się w wcześniej przygotowanej
# ramce i mają Class =1
wynik <- merge(Users[,c("Id","DisplayName","Reputation","Age","Location")],
               ValuableBadges,by.x="Id",by.y="UserId")
# łącze powstałą ramkę danych z wybranymi kolumnami z Posts
```

```
wynik <- wynik[!duplicated(wynik$Id),1:5]
# usuwam duplikaty i wybieram 5 pierwszych kolumn
wynik}
all_equal(df_sql6,df_base_6())

## [1] TRUE
```

7.3 Dplyr

```
df_dplyr_6 <- function(){
inner_tmp <- filter(Badges,Class==1)%>%select(Name)%>%group_by(Name)%>%
                count(Name)%>% filter(2<=n& n<=10)%>% select(Name)
# Tworzę ramkę z obserwacjami z Name występującymi
# od 2 do 10 razy i mającymi Class=1
ValuableBadges <- filter(Badges,Class==1)%>% select(Name,UserId)%>%
                filter(Name %in% pull(inner_tmp,Name))
# wybieram odpowiednie kolumny z ramki Badges i wybieram te wiersze
# których Name jest w poprzedniej ramce
wynik <- inner_join(ValuableBadges,Users,by=c("UserId"="Id"))
# łączę powstałą ramkę z ramką Users
wynik <- distinct(select(wynik,Id=UserId,DisplayName,Reputation,Age,Location))
# wybieram potrzebne kolumny i sortuje
wynik}
all_equal(df_sql6,df_dplyr_6())

## [1] TRUE
```

7.4 Data.Table

```
df_data.table_6 <- function(){
Users.dt <- data.table(Users)
Badges.dt <- data.table(Badges)
inner_tmp.dt <- Badges.dt[Class==1,.N,by=Name]
#Przygotowanie pomocniczej kolumny zliczającej Name
inner_tmp.dt <- inner_tmp.dt[2<=N&N<=10]
# Przygotowanie ramki z odpowiednimi Name

ValuableBadges.dt <- Badges.dt[Name%in%inner_tmp.dt[,Name]&Class==1,.(Name,UserId)]
# wybranie tych wierszy z Badges w których Name znajduje
# się w wcześniej przygotowanej ramce oraz ich Class =1
ValuableBadges.dt <- ValuableBadges.dt[!duplicated(ValuableBadges.dt[,UserId])]
# usuwam duplikaty
wynik.dt <- Users.dt[ValuableBadges.dt,on=c(Id="UserId"),
                     .(Id,DisplayName,Reputation,Age,Location)]
# wybieram potrzebne kolumny
wynik.dt}

all_equal(df_sql6,df_data.table_6())
```

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

7.5 Benchmark

```
 y<- capture.output(microbenchmark::microbenchmark(times=100,
  sql_6=df_sql_6(),
  base_6=df_base_6(),
  dplyr_6=df_dplyr_6(),
  data.table_6=df_data.table_6()))

saveRDS(y,file="bench6.rds")
x <- readRDS(file="bench6.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr        min         lq      mean     median         uq
##         sql_6 209.331457 213.331536 221.09513 218.779601 225.589631
##        base_6   7.946978   8.216284  10.19123   8.594029   8.934586
##       dplyr_6  19.776121  21.315961  24.00396  22.090485  24.481219
##  data.table_6  12.333582  12.961370  16.38600  13.338045  14.624222
##        max neval
##  263.59774   100
##   41.88720   100
##   57.25668   100
##   49.86831   100
```

8 Zadanie 7

8.1 SQL

```
df_sql_7 <- function(){
x<- sqldf::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")
 return(x)}
df_sql7 <- df_sql_7()
#ramka do porównań
```

Interpretacja zapytania:
W rezultacie otrzymujemy 10 pytań, które mają najwiecej głosów UpVote sprzed 2016 roku i żadnych głosów UpVote po roku 2016.
`UWAGA` Należy zauważyć , że kolumna suma od Total nie jest potrzebna do uzyskania wyniku pomimo tego jest ona uwzględniona w poniższym kodzie aby można było jak najlepej porównać działanie SQL i innych narzędzi.

8.2 Base

```
df_base_7 <- function(){
inner_tmp <- Votes[Votes[,"VoteTypeId"]==2,c("PostId","CreationDate")]
d <- substring(inner_tmp$CreationDate,1,4)
```

```
# Zmieniam foramt zapisu daty zachowując tylko rok
d <- ifelse(d==2017 |d ==2016,"new","old")
# Tworzę kolumne gdzie new ==2017 lub 2016 a pozstałe to old
colnames(inner_tmp)[2] <- "VoteDate"
inner_tmp[,"VoteDate"] <- d
inner_tmp <- as.data.frame(table(inner_tmp),responseName="Total",
                           stringsAsFactors=FALSE)
#zliczam wiersze w kolumnie
NewVotes <- ifelse(inner_tmp$VoteDate=="new",inner_tmp$Total,0)
OldVotes <- ifelse(inner_tmp$VoteDate=="old",inner_tmp$Total,0)
# tworzę  dwie nowe kolumny NewVotes i OldVotes
VotesByAge2 <- cbind(inner_tmp,NewVotes,OldVotes)
part1 <- aggregate(list(VotesByAge2$NewVotes,VotesByAge2$OldVotes),
                   by=list(VotesByAge2$PostId),(max))
# liczę maxy z kolumn NewVotes i OldVotes
# grupuję po PostId
part3 <- aggregate(VotesByAge2$Total,by=list(VotesByAge2$PostId),
                   FUN = function(x){sum(as.numeric(x))})
# sumuję kolumnę Total w każdej grupie
VotesByAge2 <- merge(part3,part1,by="Group.1")
# łączę ze sobą 2 poprzednie ramki
colnames(VotesByAge2) <- c("PostId","Votes","NewVotes","OldVotes")
VotesByAge2 <- VotesByAge2[VotesByAge2[,"NewVotes"]==0,
                           c("PostId","NewVotes","OldVotes","Votes")]
# wybieram potrzebne kolumny i wiersze gdzie NewVotes=0
VotesByAge2[["PostId"]] <- as.integer(VotesByAge2$PostId)
result <- merge(VotesByAge2,Posts[Posts[,"PostTypeId"]==1,c("Title","Id")],
                by.x="PostId",by.y="Id",all = FALSE)
# łącze z wybranymi kolumnami z ramki Posts
result <- result[rev(order(result[["PostId"]])),]
result<-result[c("Title","OldVotes")]
result <- head(result[rev(order(result[["OldVotes"]])),],10)
# odpowiedni sortuje i wybierma 10 pierwszych
result}
all_equal(df_sql7,df_base_7(),convert = TRUE)

## [1] TRUE
```

Używam argumentu `convert=TRUE` ponieważ rzutowanie typów bardzo spowalniłoby funkcje.

8.3 Dplyr

```
df_dplyr_7 <- function(){
inner_tmp <- filter(Votes,VoteTypeId==2)%>% select(PostId,CreationDate)
d <- substring(pull(inner_tmp,CreationDate),1,4)
# Przerabiam format daty
d <- ifelse(d==2017 |d ==2016,"new","old")
# Tworze nową kolumne gdzie new == 2017 lub 2016 a pozostałe to old
inner_tmp <- select(inner_tmp,CreationDate,PostId) %>%
  mutate(d) %>%select(PostId,d) %>% group_by(PostId,d,.drop = TRUE)%>%
  count(name="Total")%>% select(PostId,VoteDate=d,Total)
```

```
# zliczam w kolumnie Total grupując po PostId
NewVotes <- ifelse(inner_tmp%>%pull(VoteDate)=="new",pull(inner_tmp,Total),0)
OldVotes <- ifelse(pull(inner_tmp,VoteDate)=="old",pull(inner_tmp,Total),0)
# towrzę nowe kolumny NewVotes i OldVotes
# więcej pod wstawką
VotesByAge2 <- select(inner_tmp,PostId,Total,VoteDate)%>%
  cbind(NewVotes=NewVotes,OldVotes=OldVotes)%>% group_by(PostId) %>%
  summarise(Votes=sum(Total),NewVotes=max(NewVotes),OldVotes=max(OldVotes))%>%
  filter(NewVotes==0)
# liczę maxy z kolumn NewVotes i OldVotes
# grupuję po PostId
result <- inner_join(VotesByAge2,Posts,by=c("PostId"="Id"))%>%
  filter(PostTypeId==1) %>%
  select(Title,OldVotes) %>% arrange(desc(OldVotes))%>% slice(1:10)
# łączę z Posts filtruje po PostTypeId = 1
# odpowiednio sortuje i wybieam 10 pierwsych
result}
all_equal(df_sql7,df_dplyr_7(),convert = TRUE)

## [1] TRUE
```

Używam argumentu `convert=TRUE` ponieważ rzutowanie typów bardzo spowalniłoby funkcje. Używam `ifelse` pomimo istnienia odpowiednika `case_when` ponieważ jest on wielokrotnie wolniejszy. Podobnie funkcja `substring` jest najszybszym sposobem na wybranie tylko roku z coluny CreationDate.

8.4 Data.Table

```
df_data.table_7 <- function(){
Votes.dt <- data.table(Votes)
Posts.dt <- data.table(Posts)
# Rzutowanie danych
inner_tmp.dt <- Votes.dt[VoteTypeId==2,.(CreationDate,PostId)]
d <- substring(inner_tmp.dt[,CreationDate],1,4)
d <- ifelse(d==2017 |d ==2016,"new","old")
# Tworzę kolumne gdzie new ==2017 lub 2016 a pozstałe to old
inner_tmp.dt <- data.table(CreationDate=d,PostId=inner_tmp.dt[,PostId])
inner_tmp.dt <- inner_tmp.dt[,.("VoteDate"=CreationDate,PostId)]
inner_tmp.dt <- inner_tmp.dt[,.("Total"=.N),by=list(PostId,VoteDate)]
# zliczam grupując po PostId i VoteDate
NewVotes <- ifelse(inner_tmp.dt[,VoteDate]=="new",inner_tmp.dt[,Total],0)
OldVotes <- ifelse(inner_tmp.dt[,VoteDate]=="old",inner_tmp.dt[,Total],0)
# tworzę dwie nowe kolumny NewVote i OldVote
VotesByAge2.dt <- data.table(inner_tmp.dt,NewVotes,OldVotes)
VotesByAge2.dt <- VotesByAge2.dt[,.("NewVotes"=max(NewVotes),
                    "OldVotes"=max(OldVotes),"Votes"=sum(Total)),
                    by=PostId][NewVotes==0,]
# liczę maksy i sume z kolumn odpowiednio
# NewVotes,OldVotes,Total grupujac po PostID
result.dt <- VotesByAge2.dt[Posts.dt,on=.(PostId=Id)]
result.dt <- result.dt[PostTypeId==1,
                    .(Title,OldVotes=as.integer(OldVotes))][order(-OldVotes)][1:10]
```

```
# Odpowiednio sortuje i biorę 10 pierwszych wierszy
result.dt}
all_equal(df_sql7,df_data.table_7())

## [1] TRUE
```

Używam **ifelse** ponieważ jest to najszybszy sposób na stworzenie takich kolumn. Podobnie funkcja **substring** jest najszybszym sposobem na wybranie tylko roku z kolumny CreationDate.

8.5 Benchamrk

```
 u <- capture.output(microbenchmark::microbenchmark(times=100,
  sql_7=df_sql_7(),
  base_7=df_base_7(),
  dplyr_7=df_dplyr_7(),
  data.table_7=df_data.table_7()))

saveRDS(u,file="bench7.rds")
x <- readRDS(file="bench7.rds")
cat(x,sep="\n")

## Unit: milliseconds
##          expr       min        lq      mean    median        uq       max
##         sql_7  956.3856  962.6795  982.9397  970.8928  997.3949 1060.0005
##        base_7 2223.1511 2312.0584 2386.4252 2363.7288 2452.2850 2666.8908
##       dplyr_7  495.0803  548.8400  578.5054  568.0370  597.1444  781.8804
##  data.table_7  257.3031  290.4793  338.1485  314.5600  378.8935  536.2811
##  neval
##    100
##    100
##    100
##    100
```

9 Podsumowanie

Z benchmarków wynika, że w większości przypadków najszybszym pakietem jest data.table pomimo rzutowania danych w funkcjach. Nie jest tak oczywiście zawsze, w przypadku prostych zapytań np. zadanie 6 najszybsze są funkcje bazowe. W zadaniu 7 natomiast są one zdecydowanie najwolniejsze (w tym przypadku wynika to raczej z nieoptymalnego kodu) . Można jednak zauważyć reguły co do szybkości wykonywania. Prawie zawsze mamy doczynienia z sytuacją base>dplyr>data.table jeżeli chodzi o czasy wykonywania.