PDU 2018/19 - Praca domowa nr 1

Konrad Komisarczyk

15.04.2019

Spis treści

0 Wstęp

Poza bibliotekami `dplyr`, `data.table`, `sqldf`, `microbenchmark`, będziemy też korzystać z `ggplot2` do prezentacji wyników porównywania czasu wykonywania zadań.

```
library(dplyr)
library(data.table)
library(ggplot2)
library(microbenchmark)
```

Będziemy pracować na uproszczonym zrzucie zanonimizowanych danych z serwisu https://travel.stackexchange.com/
Będziemy korzystać z wchodzących w jego skład ramek danych, które ładujemy poniżej:

```
Badges <- read.csv("res/Badges.csv.gz")
Comments <- read.csv("res/Comments.csv.gz")
Posts <- read.csv("res/Posts.csv.gz")
Users <- read.csv("res/Users.csv.gz")
Votes <- read.csv("res/Votes.csv.gz")
```

Przygotowywujemy także dane w formacie `data.table`:

```
DT_Badges <- as.data.table(Badges)
DT_Comments <- as.data.table(Comments)
DT_Posts <- as.data.table(Posts)
DT_Users <- as.data.table(Users)
DT_Votes <- as.data.table(Votes)
```

Rozwiązania wykorzystujące pakiet `data.table` przyjmują dane w tym formacie i nie konwertują ich wewntąrz.

Poandto definiujemy globalnie następujące wartości:

```
options(stringsAsFactors = FALSE)

benchmark_times = 24 # liczba wykonań każdego sposobu w benchmarkach
```

1 Zadanie 1

Zadanie polega na znalezieniu informacji o użytkownikach, którzy zadali najlepsze pod względem liczby polubień pytania.

Znajdujemy następujące informacje o 10 użytkownikach, dla których suma polubień pod ich wszystkimi pytania była największa:
nazwę użytkownika, wiek, lokalizację, sumę liczby polubień pod jego pytaniami, liczbę polubień, tytuł jego najwięcej razy polubionego postu.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

Tabela `Posts` przechowuje w kolumnie `FavoriteCount` wartość `NA` dla postów mających 0 polubień. We wszystkich rozwiązaniach zadania w R powodowało to problemy. Funkcje agregujące w SQL inaczej traktują `NA` niż w R, ale i tak wartość `NA` w tym miejscu jest bardzo nie intuicyjna.

1.1 base R

```r
df_base_1 <- function(Users, Posts) {
  stopifnot(is.data.frame(Users))
  stopifnot(is.data.frame(Posts))

  # Tabela Posts ma wartość NA w kolumnie FavoriteCount dla nigdy nie
  # polubionych pytań.
  # R inaczej traktuje wartości NA przy agregacji niż SQL.
  # Zmienimy te wartości na zera.
  Posts$FavoriteCount[is.na(Posts$FavoriteCount)] <- 0

  Questions <- Posts[Posts$PostTypeId == 1, ]

  # Liczbę polubień wszystkich pytań użytkownika
  FavoriteTotal <- aggregate(Questions["FavoriteCount"],
                             by = Questions["OwnerUserId"],
                             FUN = sum)
  colnames(FavoriteTotal)[2] <- "FavoriteTotal"

  # Największą liczbę polubień wśród pytań użytkownika
  MostFavoriteQuestionLikes <- aggregate(Questions["FavoriteCount"],
                                         by = Questions["OwnerUserId"],
                                         FUN = max)
  colnames(MostFavoriteQuestionLikes)[2] <- "MostFavoriteQuestionLikes"

  # Tytuły najwięcej razy polubionych spośród pytań autorstwa użytkownika
  Q_MFQL <- merge(Questions, MostFavoriteQuestionLikes,
                  by = "OwnerUserId")
  MostFavoriteQuestion <-
    Q_MFQL[Q_MFQL$FavoriteCount == Q_MFQL$MostFavoriteQuestionLikes,
           c("OwnerUserId", "Title")]
  colnames(MostFavoriteQuestion)[2] <- "MostFavoriteQuestion"

  # W ten sposób możemy otrzymać więcej niż 1 tytuł dla użytkownika
  # (jeżeli miał więcej niż 1 pytanie o liczbie polubień równej maksymalnej)
  # SQL zwraca tylko jeden tytuł na użytkownika.
  # Można by w tym miejscu łatwo usunąć duplikaty, ale przy ograniczeniu
  # do 10 rekordów róznica ta nie ma wpływu na wynik.

  # Łączymy otrzymane wyniki dotyczące pytań w jedną tabelę
  FT_MFQL <- merge(FavoriteTotal, MostFavoriteQuestionLikes,
                   by = "OwnerUserId")
  QuestionsResults <- merge(FT_MFQL, MostFavoriteQuestion,
                            by = "OwnerUserId")

  # Dopasowywujemy informacje o użytkownikach
  # do otrzymanych dla nich wyników
  Results <- merge(Users, QuestionsResults,
```

```
                    by.x = "Id", by.y = "OwnerUserId")

  # Wybieramy interesujące nas kolumny
  Results <- Results[, c("DisplayName",
                         "Age",
                         "Location",
                         "FavoriteTotal",
                         "MostFavoriteQuestion",
                         "MostFavoriteQuestionLikes")]

  # Sortujemy i wybieramy pierwsze 10 wierszy
  head(Results[order(QuestionsResults$FavoriteTotal, decreasing = TRUE), ],
       10)
}
```

1.2 dplyr

```r
df_dplyr_1 <- function(Users, Posts) {
  stopifnot(is.data.frame(Users))
  stopifnot(is.data.frame(Posts))

  Questions <- Posts %>%
    filter(PostTypeId == 1)

  # Tabela Posts ma wartość NA w kolumnie FavoriteCount dla nigdy nie
  # polubionych pytań.
  # R inaczej traktuje wartości NA przy agregacji niż SQL.
  # Zmienimy te wartości na zera.
  na_to_zero <- function(x) ifelse(is.na(x), 0, x)

  # Obliczamy dla każdego użytkownika:
  # liczbę polubień jego wszystkich pytań,
  # i liczbę polubień jego najwięcej razy polubionego pytania
  Favorites <- Questions %>%
    mutate(FavoriteCount = na_to_zero(FavoriteCount)) %>%
    group_by(OwnerUserId) %>%
    summarise(FavoriteTotal = sum(FavoriteCount),
              MostFavoriteQuestionLikes = max(FavoriteCount))
  ## W tym miejscu można by już policzyć wartość kolumny
  ## MostFavoriteQuestion jako Title[which.max(FavoriteCount)]
  ## wewntątrz summarise,
  ## jednak znacznie zwiększa to czas działania funkcji.
  ## (aż do średnio 20s w raporcie)

  # Zatem teraz dopasowywujemy tytuł najwięcej razy polubionego postu
  Favorites <- Favorites %>%
    inner_join(Questions,
               by = c("OwnerUserId",
                      "MostFavoriteQuestionLikes" = "FavoriteCount"))

  Favorites %>%
    # Dopasowywujemy do obliczonych wartości informacje o użytkowniku
    inner_join(Users, by = c("OwnerUserId" = "Id")) %>%
    select(DisplayName, Age, Location, FavoriteTotal,
           MostFavoriteQuestion = Title, MostFavoriteQuestionLikes) %>%
    arrange(desc(FavoriteTotal)) %>%
    slice(1:10)
}
```

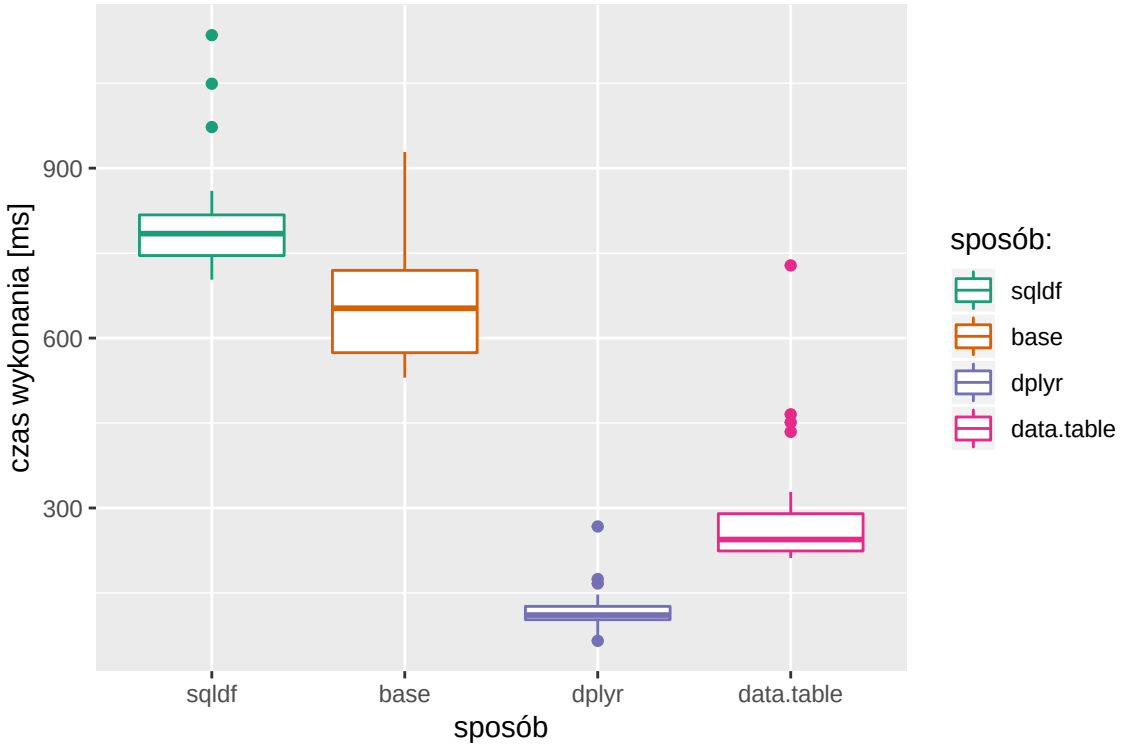
1.3 data.table

```
df_table_1 <- function(Users, Posts) {
  stopifnot(is.data.table(Users))
  stopifnot(is.data.table(Posts))

  # Tabela Posts ma wartość NA w kolumnie FavoriteCount dla nigdy nie
  # polubionych pytań.
  # R inaczej traktuje wartości NA przy agregacji niż SQL.
  # Zmienimy te wartości na zera.
  na_to_zero <- function(x) ifelse(is.na(x), 0, x)
  Posts[, FavoriteCount := na_to_zero(FavoriteCount)]

  Questions <- Posts[PostTypeId == 1]

  # Znajdujemy sumę polubień pod pytaniami każdego użytkownika
  # i jego najwięcej razy polubione pytanie (jego liczbę polubień i tytuł)
  QuestionsByUsers <-
    Questions[, .(FavoriteTotal = sum(FavoriteCount),
                  MostFavoriteQuestionLikes = max(FavoriteCount),
                  MostFavoriteQuestion = Title[which.max(FavoriteCount)]),
              by = OwnerUserId]

  # Dopasowywujemy do policzonych wartości informacje o użytkownikach
  Results <- merge(QuestionsByUsers, Users,
                   by.x = "OwnerUserId", by.y = "Id")

  # Wybieramy jedynie interesujące nas kolumny
  Results <- Results[, .(DisplayName, Age, Location, FavoriteTotal,
                         MostFavoriteQuestion, MostFavoriteQuestionLikes)]

  # Sortujemy i wybieramy pierwsze 10
  setorder(Results, -FavoriteTotal)[1:10]
}
```

1.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_1 <- microbenchmark::microbenchmark(
  sqldf = df_sql_1(Users, Posts),
  base = df_base_1(Users, Posts),
  dplyr = df_dplyr_1(Users, Posts),
  data.table = df_table_1(DT_Users, DT_Posts),
  times = benchmark_times
)
```

```
Unit: milliseconds
       expr min    lq     mean median    uq  max neval
      sqldf 703 744.5 809.9583  784.5 817.5 1135    24
       base 530 573.5 669.0417  652.5 732.0  928    24
      dplyr  65 102.5 118.4583  111.0 127.0  267    24
 data.table 212 223.0 292.5833  244.0 293.5  728    24
```



Rysunek 1.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 1.

2 Zadanie 2

Zadanie polega na znalezieniu 10 pytań o największej liczbie pozytywnie ocenionych odpowiedzi.

Wyświetlamy następujące informacje o tych pytaniach:
ID, tytuł, liczbę pozytywnie ocenionych odpowiedzi

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

2.1 base R

```r
df_base_2 <- function(Posts) {
  stopifnot(is.data.frame(Posts))

  # Wybieramy spośród postów będące odpowiedziami o pozytywnej ocenie
  PositiveAnswers <- Posts[(Posts$PostTypeId == 2) & (Posts$Score > 0), ]

  # Zliczamy liczbę pozytywnych odpowiedzi dla każdego posta
  # (mającego jakieś pozytywne odpowiedzi)
  Posts2 <- aggregate(PositiveAnswers["PostTypeId"],
                      by = PositiveAnswers["ParentId"],
                      FUN = length)
  colnames(Posts2)[2] <- "PositiveAnswerCount"

  # Dopasowywujemy policzone wyniki dla odpowiedzi
  # do pytań na które te odpowiedzi odpowiadają
  Results <- merge(Posts, Posts2, by.x = "Id", by.y = "ParentId")

  # Wybieramy jedynie interesujące nas kolumny
  Results <- Results[, c("Id", "Title", "PositiveAnswerCount")]

  # Sortujemy i wybieramy pierwsze 10
  head(Results[order(Results$PositiveAnswerCount, decreasing = TRUE), ],
       10)
}
```

2.2 dplyr

```r
df_dplyr_2 <- function(Posts) {
  stopifnot(is.data.frame(Posts))

  Posts2 <- Posts %>%
    # Wybieramy odpowiedzi o dodatniej ocenie
    filter(PostTypeId == 2, Score > 0) %>%
    # Zliczamy ich liczbę dla każdego rodzica
    group_by(ParentId) %>%
    summarise(PositiveAnswerCount = n())

  # Dopasowywujemy informacje o pytaniach - rodzicach
  # dla których zliczaliśmy
  inner_join(Posts, Posts2, by = c("Id" = "ParentId")) %>%
    select(Id, Title, PositiveAnswerCount) %>%
    arrange(desc(PositiveAnswerCount)) %>%
    slice(1:10)
}
```

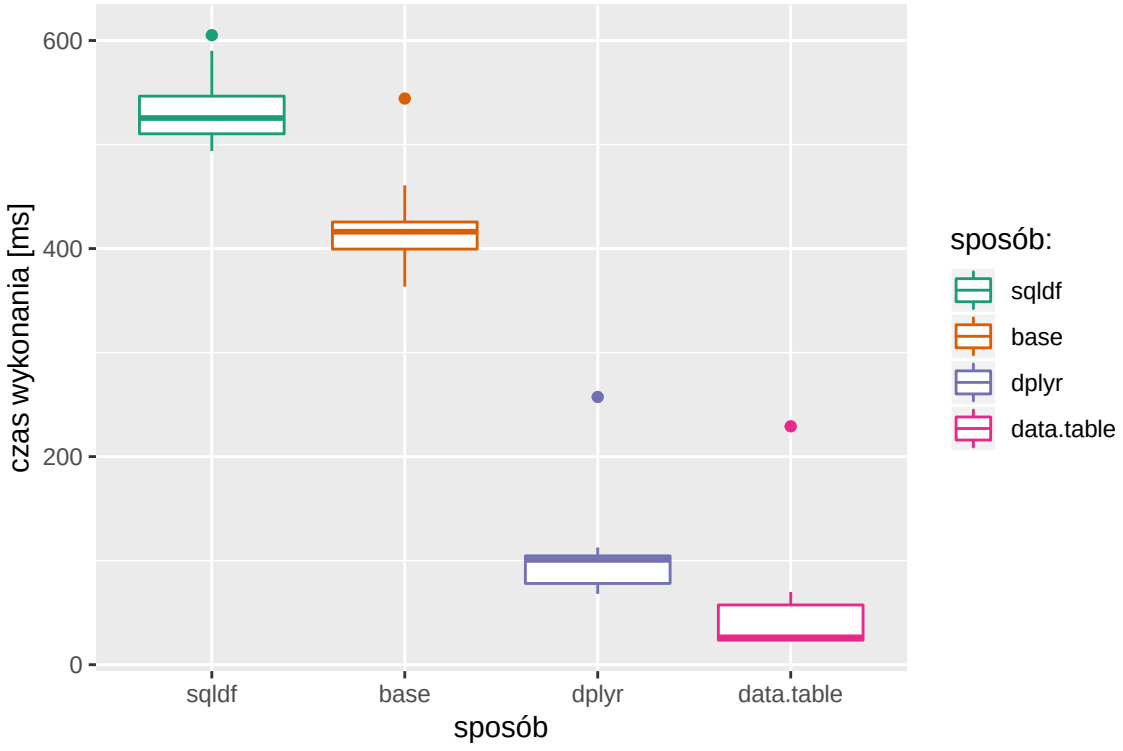
2.3 data.table

```
df_table_2 <- function(Posts) {
  stopifnot(is.data.table(Posts))

  # Wybieramy posty będące pozytywnie ocenionymi odpowiedziami
  PositiveAnswers <- Posts[PostTypeId == 2 & Score > 0]

  # Dla każdego postu-rodzica zliczamy liczbę pozytywnych odpowiedzi
  # odpowiadających na niego
  Posts2 <- PositiveAnswers[, .(PositiveAnswerCount = .N), by = ParentId]

  # Dopasowywujemy do każdego pytania (mającego jakieś pozytywne odpowiedzi)
  # liczbę pozytywnych odpowiedzi na nie
  Result <- merge(Posts, Posts2, by.x = "Id", by.y = "ParentId")

  # Wybieramy interesujące nas kolumny
  Result <- Result[, .(Id, Title, PositiveAnswerCount)]

  # Sortujemy i wybieramy pierwsze 10
  setorder(Result, -PositiveAnswerCount)[1:10]
}
```

2.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_2 <- microbenchmark::microbenchmark(
  sqldf = df_sql_2(Posts),
  base = df_base_2(Posts),
  dplyr = df_dplyr_2(Posts),
  data.table = df_table_2(DT_Posts),
  times = benchmark_times
)
```

```
Unit: milliseconds
       expr min    lq     mean median    uq max neval
      sqldf 494 510.0 532.5833  525.5 548.5 605    24
       base 363 399.5 418.7500  416.0 426.5 544    24
      dplyr  68  78.0 101.0417  101.0 105.5 257    24
 data.table  23  24.0  47.2500   26.5  57.5 229    24
```



Rysunek 2.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 2.

3 Zadanie 3

Zadanie polega na znalezieniu dla każdego roku pytania, które otrzymało w trakcie tego roku najwięcej UpVotes.

Wyświetlamy dla każdego roku następujące informacje:
który to rok, tytuł pytania mającego najwięcej UpVotes, liczbę UpVotes zebranych przez to pytanie.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

3.1 base R

```r
df_base_3 <- function(Posts, Votes) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Votes))

  # Wybieramy tylko votes typu 2, czyli UpMod
  UpVotes <- Votes[Votes$VoteTypeId == 2, ]

  # Podmieniamy datę na sam rok
  UpVotes$CreationDate <- format(as.Date(UpVotes$CreationDate), "%Y")
  colnames(UpVotes)[2] <- "Year"

  # Zliczamy liczbę UpVotes każdego postu dla każdego roku
  UpVotesPerYear <- aggregate(UpVotes["Id"],
                              by = c(UpVotes["PostId"], UpVotes["Year"]),
                              FUN = length)
  colnames(UpVotesPerYear)[3] <- "Count"

  Questions <- Posts[Posts$PostTypeId == 1, ]

  # Wybieramy spośród postów tylko pytania i dopasowywujemy informacje
  # o pytaniu
  UVPY_Q <- merge(UpVotesPerYear, Questions, by.x = "PostId", by.y = "Id")

  # Wybieramy jedynie interesujące nas kolumny
  UVPY_Q <- UVPY_Q[, c("Year", "Count", "Title")]

  # Obliczamy maksymalną liczbę Upvotes dla posta w danym roku
  MaxUpVotes <- aggregate(UVPY_Q["Count"], by = UVPY_Q["Year"],
                          FUN = max)

  # Dopasowywujemy tytuły najwięcej z UpVotowanych postów w danym roku
  # do ich roku i liczby UpVotes
  merge(UVPY_Q, MaxUpVotes, by = c("Year", "Count"))
}
```

3.2 dplyr

```r
df_dplyr_3 <- function(Posts, Votes) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Votes))

  # Liczymy dla każdego roku liczbę UpVotes dla każdego postu w tym roku
  UpVotesPerYear <- Votes %>%
    filter(VoteTypeId == 2) %>%
    mutate(Year = format(as.Date(CreationDate), "%Y")) %>%
    group_by(PostId, Year) %>%
    summarise(Count = n())

  Questions <- filter(Posts, PostTypeId == 1)

  # Dla każdego roku zliczamy jaką maksymalną liczbę UpVotes
  # zebrało jakieś pytanie z tego roku
  MaxUpVotesPerYear <- UpVotesPerYear %>%
    inner_join(Questions, by = c("PostId" = "Id")) %>%
    group_by(Year) %>%
    summarise(Count = max(Count))

  # Dopasowywujemy tytuł do każdego pytania
  MaxUpVotesPerYear %>%
    # Najpierw musimy wyciągnąć PostId tego pytania
    inner_join(UpVotesPerYear, by = c("Year", "Count")) %>%
    # Teraz możemy za pomocą PostId odnaleźć tytuł w tabeli Questions
    inner_join(Questions, by = c("PostId" = "Id")) %>%
    select(Title, Year, Count)
}
```

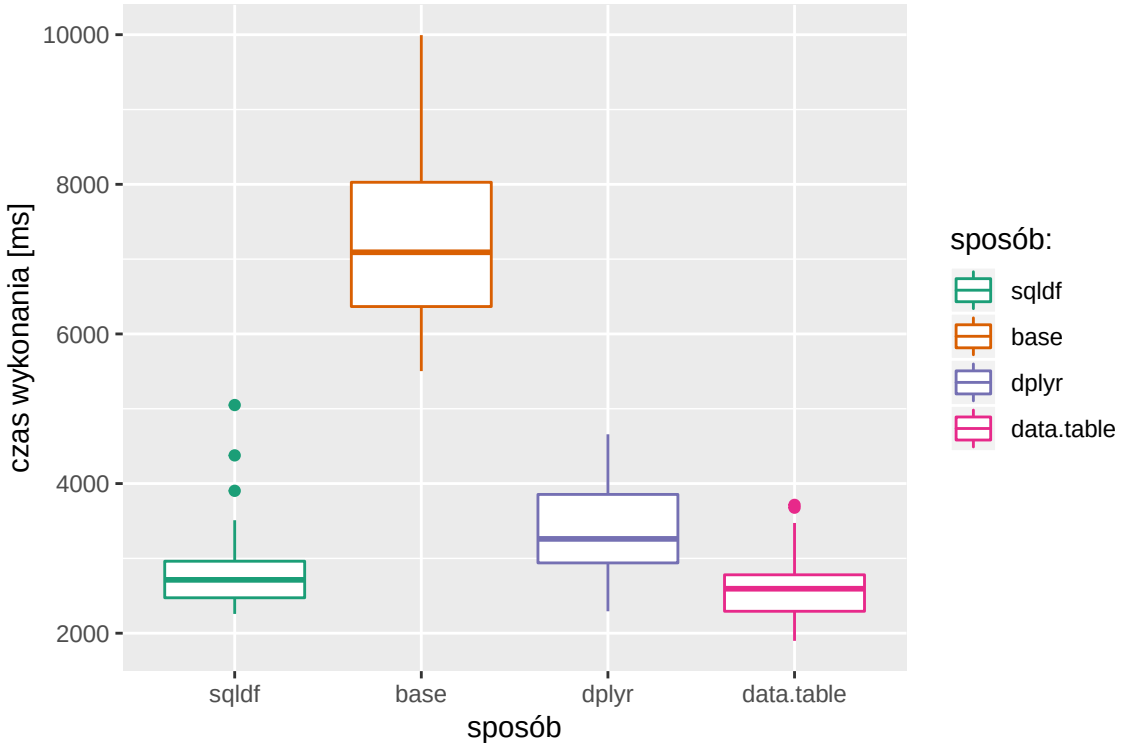
3.3 data.table

```
df_table_3 <- function(Posts, Votes) {
  stopifnot(is.data.table(Posts))
  stopifnot(is.data.table(Votes))

  # Wybieramy głosy typu UpMod, czyli o VoteTypeId = 2
  UpMods <- Votes[VoteTypeId == 2]

  # Dodajemy kolumnę Year
  UpMods <- UpMods[, Year := format(as.Date(CreationDate), "%Y")]

  # Zliczamy liczbę UpVotes dla każdego postu dla każdego roku
  UpVotesPerYear <- UpMods[, .(Count = .N), by = .(PostId, Year)]

  Questions <- Posts[Posts$PostTypeId == 1]

  # Dopasowywujemy informacje o poście do każdego postu
  UVPY_Q <- merge(UpVotesPerYear, Questions,
                  by.x = "PostId", by.y = "Id")

  # Grupujemy po roku i wybieramy post mający najwięcej UpVotes
  # dla każdego roku
  UVPY_Q[, .(Count = max(Count), Title = Title[which.max(Count)]),
         by = Year]
}
```

3.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_3 <- microbenchmark::microbenchmark(
  sqldf = df_sql_3(Posts, Votes),
  base = df_base_3(Posts, Votes),
  dplyr = df_dplyr_3(Posts, Votes),
  data.table = df_table_3(DT_Posts, DT_Votes),
  times = benchmark_times
)
```

```
Unit: seconds
       expr   min     lq     mean median     uq   max neval
      sqldf 2.258 2.4710 2.894917 2.7130 2.9970 5.050    24
       base 5.503 6.3275 7.310292 7.0915 8.0550 9.994    24
      dplyr 2.293 2.8510 3.392708 3.2605 3.8895 4.659    24
 data.table 1.898 2.2840 2.620250 2.5940 2.7945 3.715    24
```



Rysunek 3.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 3.

4 Zadanie 4

Zadanie polega na znalezieniu takich pytań, w których przypadku różnica między oceną (Score) najlepiej ocenionej odpowiedzi, a oceną zaakceptowanej odpowiedzi jest większa od 50.

Zwraca następujące informacje o tych pytaniach:
ID, tytuł pytania, ocenę najlepiej ocenionej odpowiedzi, ocenę zaakceptowanej odpowiedzi oraz różnicę między tymi ocenami,
w kolejności od najwiekszej różnicy ocen.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

4.1 base R

```r
df_base_4 <- function(Posts) {
  stopifnot(is.data.frame(Posts))

  Questions <- Posts[Posts$PostTypeId == 1, ]

  Answers <- Posts[Posts$PostTypeId == 2, ]

  # Znajdujemy najwyższe oceny odpowiedzi dla każdego rodzica (pytania).
  BestScores <- aggregate(Answers["Score"], by = Answers["ParentId"],
                          FUN = max)

  # Łączymy BestScores z Answers i wybieramy odpowiedzi
  # o maksymalnych ocenach.
  BestAnswers <- merge(Answers, BestScores, by = "ParentId")
  BestAnswers <- BestAnswers[BestAnswers$Score.x == BestAnswers$Score.y, ]

  # Wybieramy tylko interesujące nas kolumny i nazywamy odpowiednio.
  BestAnswers <- BestAnswers[, c("Id", "ParentId", "Score.x")]
  colnames(BestAnswers)[3] <- "MaxScore"

  # BestAnswers zawiera więcej rekordów niż odpowiadająca tabela z SQL,
  # ponieważ w przypadku gdy istnieje kilka odpowiedzi o maksymalnej
  # ocenie, zawiera je wszystkie.

  # Joinujemy
  BA_Q <- merge(BestAnswers, Questions, by.x = "ParentId", by.y = "Id")
  Result <- merge(BA_Q, Posts, by.x = "AcceptedAnswerId", by.y = "Id")

  # Obliczamy wartość różnicy.
  Result$Difference <- Result$MaxScore - Result$Score.y

  # Wybieramy jedynie interesujące nas kolumny i ustawiamy nazwy.
  Result <- Result[, c("ParentId.x", "Title.x", "MaxScore", "Score.y",
                       "Difference")]
  colnames(Result)[c(1, 2, 4)] <- c("Id", "Title", "AcceptedScore")

  # Wybieramy jedynie pytania dla których różnica > 50.
  Result <- Result[Result$Difference > 50, ]

  # Sortujemy.
  Result[order(Result$Difference, decreasing = TRUE), ]
}
```

4.2 dplyr

```r
df_dplyr_4 <- function(Posts) {
  stopifnot(is.data.frame(Posts))

  Questions <- Posts %>%
    filter(PostTypeId == 1)

  Answers <- Posts %>%
    filter(PostTypeId == 2)

  # Dla każdego posta wybieramy najwyższy Score odpowiedzi do niego.
  BestAnswers <- Answers %>%
    group_by(ParentId) %>%
    summarise(MaxScore = max(Score))

  Questions %>%
    # Dopasowujemy do każdego pytania najwyższy Score odpowiedzi do niego.
    inner_join(BestAnswers, by = c("Id" = "ParentId")) %>%
    # Teraz dla każdego pytania znajdujemy jego zaakceptowaną odpowiedź.
    inner_join(Posts, by = c("AcceptedAnswerId" = "Id")) %>%
    # Wybieramy interesujące nas kolumny.
    select(Id, Title = Title.x, MaxScore, AcceptedScore = Score.y) %>%
    # Obliczamy różnice.
    mutate(Difference = (MaxScore - AcceptedScore)) %>%
    # Wybieramy tylko posty o różnicach > 50.
    filter(Difference > 50) %>%
    # Sortujemy.
    arrange(desc(Difference))
}
```

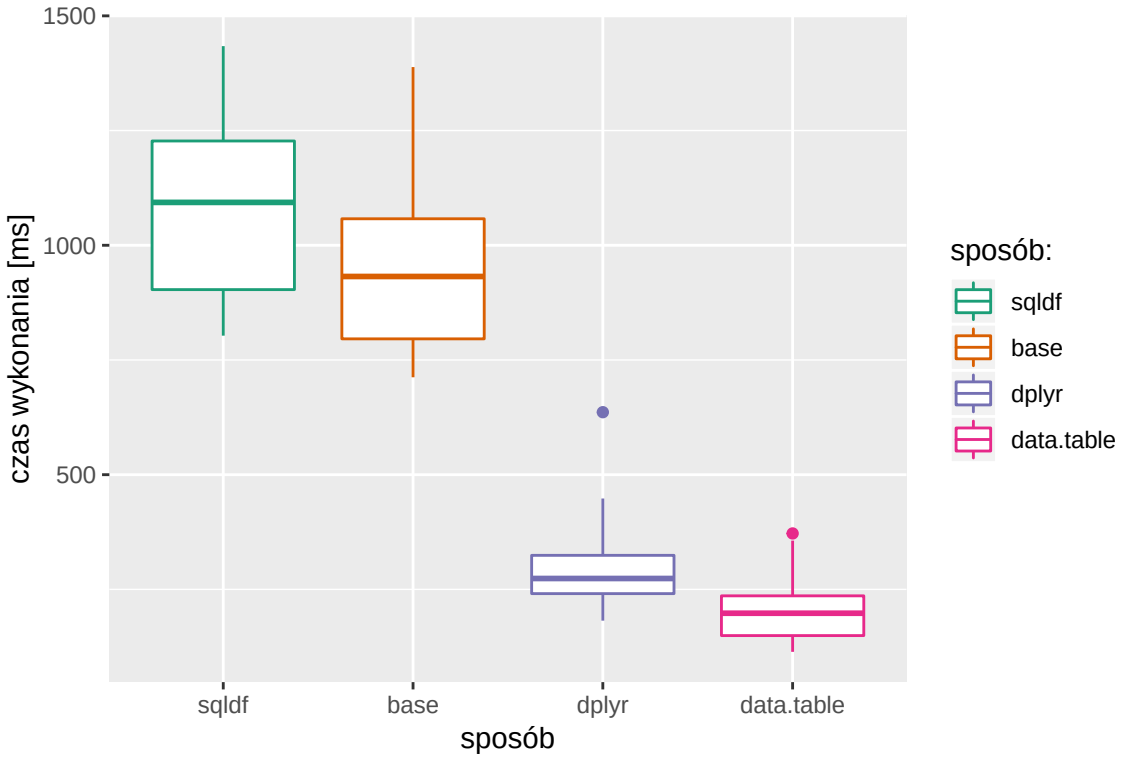
4.3 data.table

```
df_table_4 <- function(Posts) {
  stopifnot(is.data.table(Posts))

  Questions <- Posts[PostTypeId == 1]

  Answers <- Posts[PostTypeId == 2]

  # Wybieramy dla każdego postu Id i Score odpowiedzi o najwyższym Score.
  BestAnswers <- Answers[, .(MaxScore = max(Score),
                             Id = Id[which.max(Score)]),
                         by = ParentId]

  # Dopasowywujemy do każdego pytania Id i Score
  # najlepszej odpowiedzi do niego.
  BA_Q <- merge(BestAnswers, Questions, by.x = "ParentId", by.y = "Id")

  # Dopasowywujemy teraz do tego do każdego pytania informację
  # o jego zaakceptowanej odpowiedzi.
  Results <- merge(BA_Q, Posts, by.x = "AcceptedAnswerId", by.y = "Id")

  # Wybieramy interesujące nas kolumny i liczymy różnice.
  Results <- Results[, .(Id = ParentId.x, Title = Title.x,
                         MaxScore, AcceptedScore = Score.y,
                         Difference = MaxScore - Score.y)]

  # Wybieramy tylko posty z różnicą > 50.
  Results <- Results[Difference > 50]

  # Sortujemy i zwracamy wynik.o
  setorder(Results, -Difference)
}
```

4.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_4 <- microbenchmark::microbenchmark(
  sqldf = df_sql_4(Posts),
  base = df_base_4(Posts),
  dplyr = df_dplyr_4(Posts),
  data.table = df_table_4(DT_Posts),
  times = benchmark_times
)
```

```
Unit: milliseconds
       expr min    lq      mean median     uq  max neval
      sqldf 803 899.5 1076.2917 1093.5 1238.0 1434    24
       base 712 795.5  959.4583  932.0 1062.5 1388    24
      dplyr 182 240.5  298.4583  274.0  328.0  636    24
 data.table 114 145.5  200.8750  198.0  236.5  372    24
```



Rysunek 4.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 4.

5 Zadanie 5

Nazwijmy komentarze autora danego pytania pod danym pytaniem wyjaśnieniami.

Zadanie polega na znalezieniu 10 "najlepiej wyjasnionych pytań", czyli takich pytań, dla których suma ocen (score) wyjaśnień była najwyższa.

Zwraca następujące informacje o tych pytaniach:
tytuł i sumę ocen wyjasnień, w kolejności od największej sumy ocen wyjaśnień.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

5.1 base R

```r
df_base_5 <- function(Posts, Comments) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Comments))

  # Dla każdego postu i użytkownika liczymy sumę ocen komentarzy
  # danego użytkownika pod danym postem.
  # Różni się ona od tabeli CmtTotScr z SQL, ponieważ UserId może być
  # równe NA, a R pomija takie przypadki.
  # Różnica w tym miejscu nie wpływa jednak na wynik końcowy, bo rekordy
  # mające UserId=NA i tak się zgubią przy joinowaniu po tym polu
  CmtTotScr <- aggregate(Comments["Score"],
                         by = c(Comments["PostId"], Comments["UserId"]),
                         FUN = sum)
  colnames(CmtTotScr)[3] <- "CommentsTotalScore"

  Questions <- Posts[Posts$PostTypeId == 1, ]

  # Dopasowywujemy do pytań policzone CommentsTotalScore.
  Result <- merge(CmtTotScr, Questions,
                  by.x = c("PostId", "UserId"),
                  by.y = c("Id", "OwnerUserId"))

  # Wybieramy jedynie interesujące nas kolumny.
  Result <- Result[, c("Title", "CommentsTotalScore")]

  # Wybieramy 10 postów o największym CommentsTotalScore.
  head(Result[order(Result$CommentsTotalScore, decreasing = TRUE), ], 10)
}
```

5.2 dplyr

```r
df_dplyr_5 <- function(Posts, Comments) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Comments))

  # Dla każdego postu, dla każdego udzielającego
  # się pod nim w komentarzach użytkownika liczymy sumę Scorów
  # wszystkich jego komentarzy.
  CmtTotScr <- Comments %>%
    group_by(PostId, UserId) %>%
    summarise(CommentsTotalScore = sum(Score)) %>%
    ungroup()

  Questions <- Posts %>%
    filter(PostTypeId == 1)

  CmtTotScr %>%
    # Wybieramy dla każdego pytania komentarze napisane pod nim przez
    # autora tego pytania. Dopasowywujemy do każdego pytania sumę Scorów
    # komentarzy autora pytania pod tym pytaniem.
    inner_join(Questions,
               by = c("PostId" = "Id", "UserId" = "OwnerUserId")) %>%
    # Wybieramy kolumny, sortujemy i wybieramy pierwsze 10.
    select(Title, CommentsTotalScore) %>%
    arrange(desc(CommentsTotalScore)) %>%
    slice(1:10)
}
```

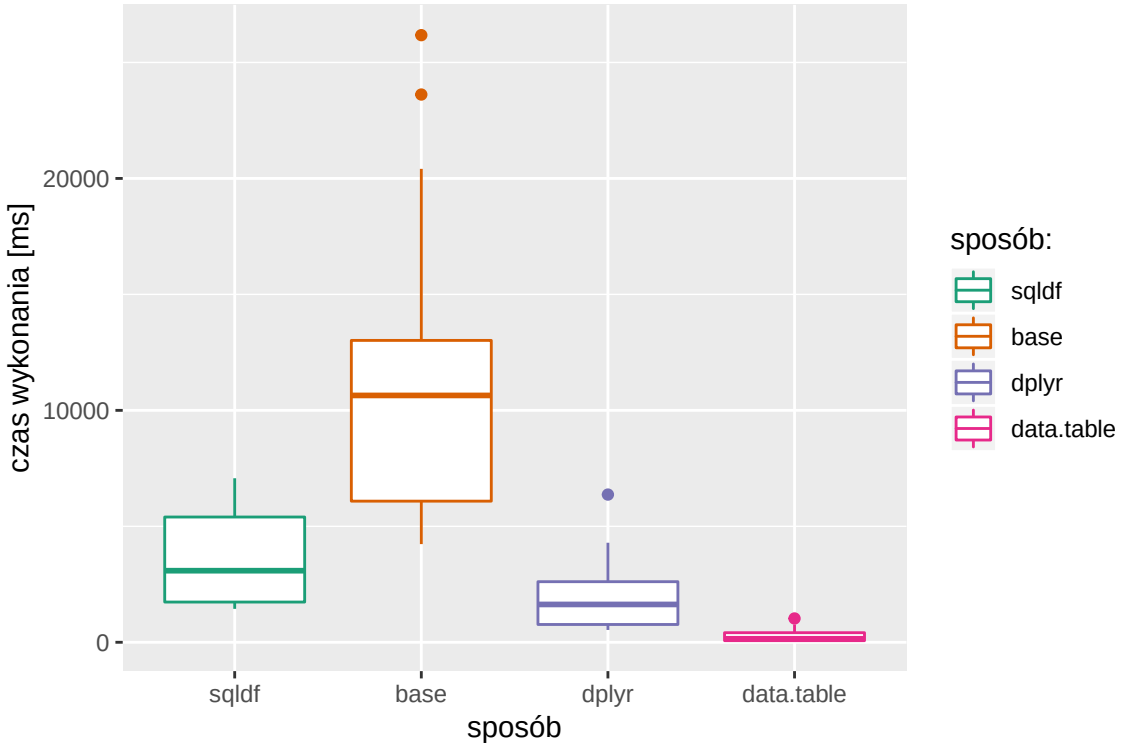
5.3 data.table

```
df_table_5 <- function(Posts, Comments) {
  stopifnot(is.data.table(Posts))
  stopifnot(is.data.table(Comments))

  # Dla każdego postu, dla każdego udzielającego się pod nim
  # w komentarzach użytkownika, liczymy sumę Scorów jego komentarzy.
  CmtTotScr <- Comments[, .(CommentsTotalScore = sum(Score)),
                        by = .(PostId, UserId)]

  Questions <- Posts[PostTypeId == 1]

  # Dopasowujemy do każdego pytania sumę Scorów wszystkich komentarzy
  # jego autora pod tym pytaniem.
  Results <- merge(Questions, CmtTotScr,
                   by.x = c("Id", "OwnerUserId"),
                   by.y = c("PostId", "UserId"))

  # # Inny sposób na operację powyższą.
  # # Nie ma istotnych różnic w szybkości działania
  # setkey(Questions, Id, OwnerUserId)
  # setkey(CmtTotScr, PostId, UserId)
  # Results <- CmtTotScr[Questions]
  # Results <- Results[!is.na(CommentsTotalScore)]
  # # END inny sposób

  # Sortujemy, wybieramy kolumny i wybieramy pierwsze 10.
  setorder(Results, -CommentsTotalScore)[1:10, .(Title, CommentsTotalScore)]
}
```

5.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_5 <- microbenchmark::microbenchmark(
  sqldf = df_sql_5(Posts, Comments),
  base = df_base_5(Posts, Comments),
  dplyr = df_dplyr_5(Posts, Comments),
  data.table = df_table_5(DT_Posts, DT_Comments),
  times = benchmark_times
)
```

```
Unit: milliseconds
       expr  min     lq      mean median      uq   max neval
      sqldf 1442 1734.5  3714.833   3087  5520.0  7073    24
       base 4234 6012.5 11155.375  10643 13157.5 26178    24
      dplyr  532  758.5  1924.250   1631  2652.0  6369    24
 data.table   63   70.0   279.000    151   433.5  1027    24
```



Rysunek 5.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 5.

6 Zadanie 6

Odznaki o `Class`=1 to złote odznaki.

Wartościową odznaką nazwiemy złotą odznakę, która do tej pory została przyznana między 2 a 10 razy (włącznie).

Zadanie polega na znalezieniu użytkowników, którzy zostali odznaczeni wartościową odznaką.

Zwraca nastepujące informacje o tych użytkownikach:
Id, wyświetlaną nazwę, reputację, wiek i lokalizację.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

6.1 base R

```
df_base_6 <- function(Badges, Users) {
  stopifnot(is.data.frame(Badges))
  stopifnot(is.data.frame(Users))

  # Wybieramy nazwy Wartościowych Odznak.
  BadgesClass1 <- Badges[Badges$Class == 1, ]
  BadgesNameCount <- aggregate(BadgesClass1["Id"],
                               by = BadgesClass1["Name"],
                               FUN = length)
  ValuableBadgesNames <- BadgesNameCount[(BadgesNameCount$Id) >= 2
                                         & (BadgesNameCount$Id <= 10),
                                         "Name"]

  # Wybieramy Id użytkowników, którzy dostali Wartościową Odznakę.
  ValuableBadges <- BadgesClass1[BadgesClass1$Name %in% ValuableBadgesNames,
                                 c("Name", "UserId")]

  # Wybieramy informacje o odznaczonych użytkownikach.
  Results <- merge(Users, ValuableBadges,
                   by.x = "Id", by.y = "UserId")
  Results <- Results[, c("Id", "DisplayName", "Reputation", "Age",
                         "Location")]

  # Wybieramy unikalne wartości.
  unique(Results)
}
```

6.2 dplyr

```r
df_dplyr_6 <- function(Badges, Users) {
  stopifnot(is.data.frame(Badges))
  stopifnot(is.data.frame(Users))

  # w zadaniu interesują nas jedynie odznaki klasy 1
  BadgesClass1 <- Badges %>%
    filter(Class == 1)

  # ## Sposób robiący to jak sql
  # # Znajdujemy nazwy odznak, które zostały przyznane między 2, a 8 razy.
  # ValuableBadgesVector <- (BadgesClass1 %>%
  #   group_by(Name) %>%
  #   tally() %>%
  #   filter(2 <= n, n <= 8)
  #   )£Name
  #
  # # Wybieramy wartościowe odznaki.
  # ValuableBadges <- BadgesClass1 %>%
  #   filter(Name %in% ValuableBadgesVector) %>%
  #   select(Name, UserId)
  # ## END sposób robiący to jak sql

  ## Prostszy sposób
  ## Trochę szybszy. Przy nim funkcja działa średnio 13ms, a przy tym, co
  ## robi, jak SQL 16ms. Max i min czasy działania nie różnią się istotnie.
  ValuableBadges <- BadgesClass1 %>%
    group_by(Name) %>%
    # Zliczamy liczbę przyznań oznak o konkretnej nazwie.
    tally() %>%
    # Wybieramy nazwy tych, które zostały przyznane między 2, a 10 razy.
    filter(2 <= n, n <= 10) %>%
    # Znajdujemy UserId odznak o tych nazwach.
    inner_join(BadgesClass1, by = "Name") %>%
    select(Name, UserId)
  ## END prostszy sposób

  # Wyciągamy informacje o znalezionych użytkownikach
  Users %>%
    inner_join(ValuableBadges, by = c("Id" = "UserId")) %>%
    select(Id, DisplayName, Reputation, Age, Location) %>%
    distinct()
}
```

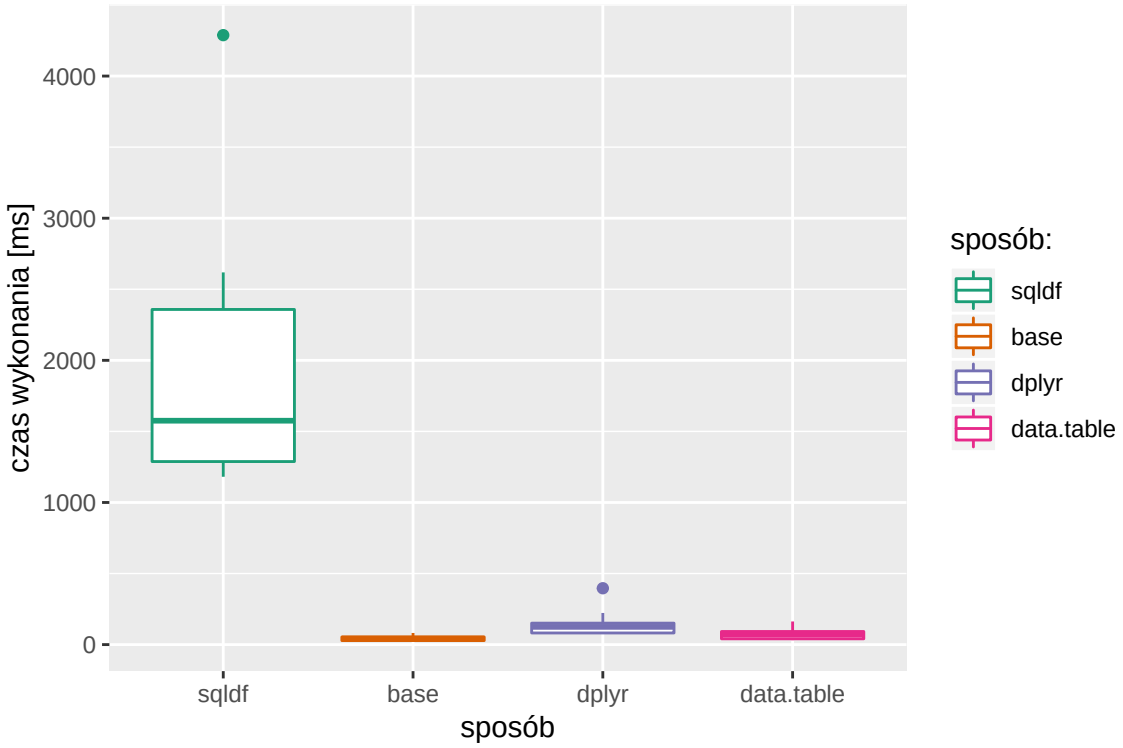
6.3 data.table

```
df_table_6 <- function(Badges, Users) {
  stopifnot(is.data.table(Badges))
  stopifnot(is.data.table(Users))

  # Liczymy, ile razy wystąpiła każda nazwa odznaki.
  BadgeNameOccurences <- Badges[Class == 1, .N, by = Name]

  # Wybieramy nazwy wartościowych odznak.
  ValuableBadgesVector <- BadgeNameOccurences[2 <= N & N <= 10]$Name

  # Dla wszystkich przyznanych wartościowych odznak znajdujemy
  # UserId użytkownika, któremu zostały przynane.
  ValuableBadges <- Badges[Class == 1 & Name %in% ValuableBadgesVector,
                           .(Name, UserId)]

  # Znajdujemy informacje o tych użytkownikach, co mieli przyznane
  # wartościowe odznaki
  Results <- merge(Users, ValuableBadges, by.x = "Id", by.y = "UserId")

  # Wybieramy interesujące nas kolumny i usuwamy powtórzenia
  unique(Results[, .(Id, DisplayName, Reputation, Age, Location)])
}
```

6.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_6 <- microbenchmark::microbenchmark(
  sqldf = df_sql_6(Badges, Users),
  base = df_base_6(Badges, Users),
  dplyr = df_dplyr_6(Badges, Users),
  data.table = df_table_6(DT_Badges, DT_Users),
  times = benchmark_times
)
```

```
Unit: milliseconds
       expr  min     lq       mean median     uq  max neval
      sqldf 1181 1287.0 1854.00000 1575.5 2368.0 4288    24
       base   27   28.0   44.58333   42.0   54.0   82    24
      dplyr   77   81.5  133.04167  125.0  151.5  396    24
 data.table   31   40.0   75.29167   70.5   93.0  163    24
```



Rysunek 6.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 6.

7 Zadanie 7

Zadanie polega na wybraniu spośród pytań mających 0 nowych (od 2016 roku) głosów typu UpMod, takich, które mają najwięcej starych (sprzed 2016) głosów typu UpMod.

Zwracamy następujące informacje o 10 z tych pytań o największej liczbie starych głosów:
tytuł i liczbę starych głosów typu UpMod,
w kolejności od największej liczby starych głosów.

Zadanie rozwiązuje następujące zapytanie SQL:

```
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
```

Zadanie można rozwiązać prościej niż robi to zapytanie SQL - bez etykietowania starych i nowych głosów napisami *new*/*old*. Ponadto SQL niepotrzebnie oblicza sumaryczną liczbę starych i nowych głosów.

7.1 base R

```
df_base_7 <- function(Posts, Votes) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Votes))

  UpModVotes <- Votes[Votes$VoteTypeId == 2, ]

  # Dzielimy głosy na nowe i stare.
  is.new <- function(x) (format(as.Date(x), "%Y") %in% c("2016", "2017"))
  NewVotes <- UpModVotes[is.new(UpModVotes$CreationDate), ]
  OldVotes <- UpModVotes[!is.new(UpModVotes$CreationDate), ]

  # Zliczamy nowe i stare głosy dla każdego posta.
  NewVotesByPost <- aggregate(NewVotes["Id"],
                              by = NewVotes["PostId"],
                              FUN <- length)
  colnames(NewVotesByPost)[2] <- "NewVotes"
  OldVotesByPost <- aggregate(OldVotes["Id"],
                              by = OldVotes["PostId"],
                              FUN <- length)
  colnames(OldVotesByPost)[2] <- "OldVotes"

  # Łaczymy tabelę starych i nowych głosów, tak, aby zachować
  # wszystkie wpisy o starych głosach.
  VotesByPost <- merge(NewVotesByPost, OldVotesByPost,
                       by = "PostId", all.y = TRUE)

  # Wybieramy te posty, które mają 0 nowych głosów, czyli
  # mają wartość NA w kolumnie NewVotes po złączeniu.
  VotesByPost2 <- VotesByPost[is.na(VotesByPost$NewVotes), ]

  Questions <- Posts[Posts$PostTypeId == 1, ]

  # Dopasowywujemy tytuły do wynikowych postów.
  Q_VBP2 <- merge(Questions, VotesByPost2, by.x = "Id", by.y = "PostId")

  # Wybieramy kolumny, sortujemy i nastepnie wybieramy pierwsze 10
  Results <- Q_VBP2[order(Q_VBP2$OldVotes, decreasing = TRUE),
                    c("Title", "OldVotes")]
  head(Results, 10)
}
```

7.1.1 rozwiązanie zgodne z SQL

Zdecydowałem się także umiescić rozwiązanie działające w taki sposób, jak SQL. Działa on tylko trochę (średnio o 16%) wolniej, ale jest bardziej skomplikowany i dłuższy.

```
df_base_7_b <- function(Posts, Votes) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Votes))

  # Dla każdego postu liczymy liczbę głosów typu 2 z każdej
  # z kategorii (nowe, stare).

  UpModVotes <- Votes[Votes$VoteTypeId == 2, ]

  # Podmieniamy zawartość kolumny z datą utworzenia na kategorię new/old.
  categorizeByYear <- function(x)
    ifelse(format(as.Date(x), "%Y") %in% c("2016", "2017"),
           "new",
           "old")
  UpModVotes$CreationDate <- categorizeByYear(UpModVotes$CreationDate)
  colnames(UpModVotes)[2] <- "VoteDate"

  # Dla każdego postu liczymy liczbę głosów typu 2 z każdej
  # z kategorii (nowe, stare).
  VotesByAge <- aggregate(UpModVotes["Id"],
                          by = c(UpModVotes["PostId"],
                                 UpModVotes["VoteDate"]),
                          FUN = length)
  colnames(VotesByAge)[3] <- "Total"

  # Obliczamy liczbę wszystkich głosów dla każdego postu.
  VotesByAge2Total <- aggregate(VotesByAge["Total"],
                                by = VotesByAge["PostId"],
                                FUN = sum)

  # Obliczamy liczbę nowych głosów dla każdego postu (mającego nowe głosy).
  VotesByAgeNew <- VotesByAge[VotesByAge$VoteDate == "new", ]
  VotesByAgeNew2 <- aggregate(VotesByAgeNew["Total"],
                              by = VotesByAgeNew["PostId"],
                              FUN = sum)
  colnames(VotesByAgeNew2)[2] <- "NewVotes"

  # Analogicznie obliczamy liczbę starych głosów dla każdego postu.
  VotesByAgeOld <- VotesByAge[VotesByAge$VoteDate == "old", ]
  VotesByAgeOld2 <- aggregate(VotesByAgeOld["Total"],
                              by = VotesByAgeOld["PostId"],
                              FUN = sum)
  colnames(VotesByAgeOld2)[2] <- "OldVotes"

  # Łączymy
  VotesByAgeTotalNew2 <- merge(VotesByAge2Total, VotesByAgeNew2,
```

```
                                by = "PostId", all.x = TRUE)
  VotesByAge2 <- merge(VotesByAgeTotalNew2, VotesByAgeOld2,
                       by = "PostId", all.x = TRUE)

  # Zastępujemy NA zerami.
  na_replace_zero <- function(x) ifelse(is.na(x), 0, x)
  VotesByAge2$NewVotes <- na_replace_zero(VotesByAge2$NewVotes)
  VotesByAge2$OldVotes <- na_replace_zero(VotesByAge2$OldVotes)

  # Wybieramy tylko te rekordy, które mają 0 nowych głosów.
  VotesByAge2 <- VotesByAge2[VotesByAge2$NewVotes == 0, ]

  Questions <- Posts[Posts$PostTypeId == 1, ]

  # Dopasowywujemy do policzonych wartości tytuły pytań.
  Result <- merge(VotesByAge2, Questions,
                  by.x = "PostId", by.y = "Id")[, c("Title", "OldVotes")]

  head(Result[order(Result$OldVotes, decreasing = TRUE), ], 10)
}
```

7.2 dplyr

```r
df_dplyr_7 <- function(Posts, Votes) {
  stopifnot(is.data.frame(Posts))
  stopifnot(is.data.frame(Votes))

  # Interesują nas jedynie głosy typu 2, czyli UpMod.
  UpMods <- Votes %>%
    filter(VoteTypeId == 2)

  is.new <- function(x) (format(as.Date(x), "%Y") %in% c("2016", "2017"))

  NewUpMods <- UpMods %>%
    # Wybieramy nowe głosy
    filter(is.new(CreationDate)) %>%
    # i zliczamy ich liczbę dla każdego postu.
    group_by(PostId) %>%
    summarise(NewVotes = n())

  # Analogicznie do NewUpMods
  OldUpMods <- UpMods %>%
    filter(!is.new(CreationDate)) %>%
    group_by(PostId) %>%
    summarise(OldVotes = n())

  ## Potem będziemy wybierać posty o największej liczbie starych głosów,
  ## załóżmy więc, że te, które mają 0 starych głosów nas nie interesują.
  ## Poza tym SQL i tak nie zwraca takich postów jak usuniemy "LIMIT 10".

  VotesByAge <- NewUpMods %>%
    # Złączamy tabele starych i nowych głosów,
    # zachowując wszystkie PostId starych głosów.
    right_join(OldUpMods, by = "PostId") %>%
    # Wybieramy te posty, które mają 0 nowych głosów,
    # czyli nie było ich w tabeli NewUpMods,
    # czyli w naszej złączonej tabeli ich liczba nowych głosów jest NA.
    filter(is.na(NewVotes))

  Questions <- Posts %>%
    filter(PostTypeId == 1)

  VotesByAge %>%
    # Wybieramy posty będące pytaniami i dopasowywujemy tytuły.
    inner_join(Questions, by = c("PostId" = "Id")) %>%
    select(Title, OldVotes) %>%
    # Wybieramy 10 postów o największej liczbie starych głosów.
    arrange(desc(OldVotes)) %>%
    slice(1:10)
}
```

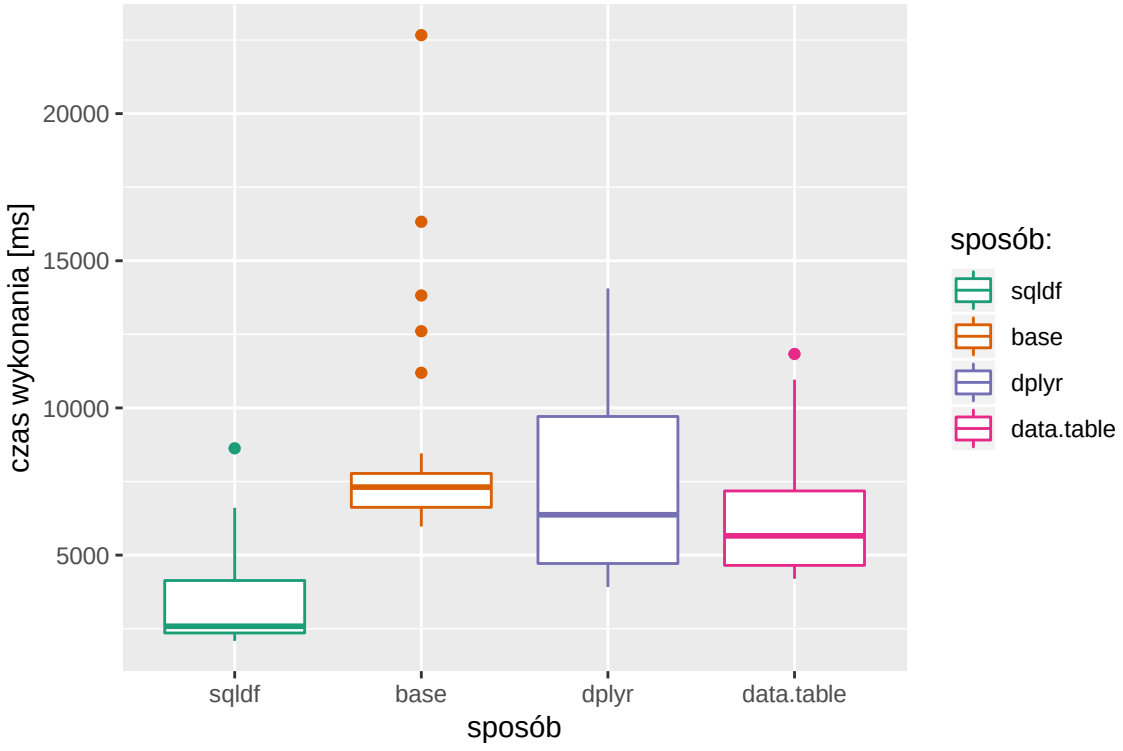
7.3 data.table

```
df_table_7 <- function(Posts, Votes) {
  stopifnot(is.data.table(Posts))
  stopifnot(is.data.table(Votes))

  is.new <- function(x) (format(as.Date(x), "%Y") %in% c("2016", "2017"))

  # Wybieramy nowe głosy typu UpMod i zliczamy je dla każdego postu.
  NewUpMods <- Votes[VoteTypeId == 2 & is.new(CreationDate),
                     .(NewVotes = .N),
                     by = PostId]

  # Analogicznie stare głosy.
  OldUpMods <- Votes[VoteTypeId == 2 & !is.new(CreationDate),
                     .(OldVotes = .N),
                     by = PostId]

  # Dopasowywujemy do liczby starych głosów zebranych przez post
  # liczbę nowych i wybieramy jedynie takie, które nie dostały żadnych
  # nowych głosów, czyli po złączeniu mają wartość NA w kolumnie NewVotes.
  VotesByAge2 <- (NewUpMods[OldUpMods, on = "PostId"])[is.na(NewVotes)]

  Questions <- Posts[PostTypeId == 1]

  # Spośród wybranych wczesniej postów wybieramy pytania
  # i dopasowywujemy informacje o nich.
  Results <- merge(VotesByAge2, Questions, by.x = "PostId", by.y = "Id")
  # Wybieramy jedynie interesujące nas kolumny.
  Results <- Results[, .(Title, OldVotes)]

  # Sortujemy i wybieramy pierwsze 10.
  setorder(Results, -OldVotes)[1:10]
}
```

7.4 Porównanie czasów wykonywania rozwiązań zadania

```
benchmark_7 <- microbenchmark::microbenchmark(
  sqldf = df_sql_7(Posts, Votes),
  base = df_base_7(Posts, Votes),
  dplyr = df_dplyr_7(Posts, Votes),
  data.table = df_table_7(DT_Posts, DT_Votes),
  times = benchmark_times
)
```

```
Unit: seconds
       expr   min     lq     mean median     uq    max neval
      sqldf 2.087 2.3520 3.377917 2.5825 4.2300  8.629    24
       base 5.972 6.6140 8.716000 7.3100 8.0010 22.666    24
      dplyr 3.913 4.7115 7.423125 6.3705 9.9330 14.060    24
 data.table 4.197 4.6500 6.319292 5.6540 7.2595 11.832    24
```



Rysunek 7.1: Wykres pudełkowy porównujący czasy działania różnych sposobów rozwiązania zadania nr 7.

8 Podsumowanie

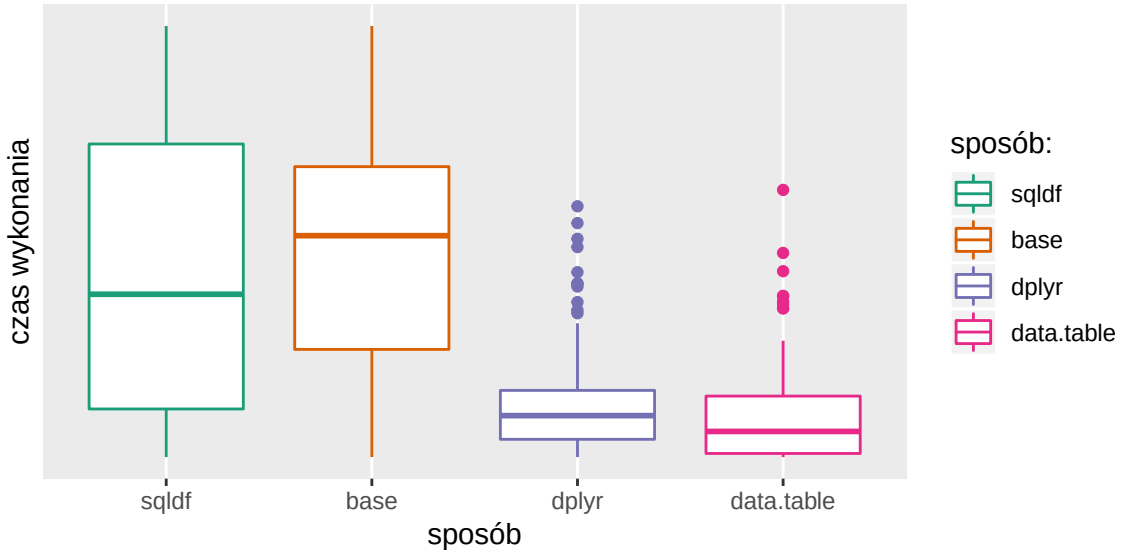
Prawie w każdym zadaniu rozwiązanie za pomocą tylko bazowego R okazało się najwolniejsze. Najszybsze zazwyczaj okazuje się użycie `data.table`, jednak pakiet `dplyr` jest niewiele wolniejszy.

Dla mnie osobiście w `dplyr` pisze się najwygodniej. Kod jest też najbardziej czytelny. Operator `%>%` pozwala łatwo uniknąć długich linii kodu. Pozwala też pisać łatwy do zrozumienia kod spełniający zależność 1 linijka = 1 operacja.

Aby jeszcze lepiej porównać czasy działania, możemy zebrać znormalizowane wewnątrz każdego zadania wyniki wszystkich benchmarków:

```
normalize <- function(benchmark) {
  benchmark %>%
    mutate(time = (time - min(time)) / (max(time) - min(time)))
}

all_benchmarks <- normalize(benchmark_1) %>%
  rbind(normalize(benchmark_2)) %>%
  rbind(normalize(benchmark_3)) %>%
  rbind(normalize(benchmark_4)) %>%
  rbind(normalize(benchmark_5)) %>%
  rbind(normalize(benchmark_6)) %>%
  rbind(normalize(benchmark_7))

class(all_benchmarks) <- c("microbenchmark", "data.frame")
```

I zilustrować ogólne czasy wykonywania za pomocą wykresu:



Rysunek 8.1: Porównanie czasów działania różnych sposobów rozwiązania wszystkich zadań, znormalizowanych wewnątrz grupy każdego zadania