PDU 2021/2022

Praca domowa nr 2 (max. = 40 p.)

Maksymalna ocena: 40 p.

Prace domowe należy przesłać za pośrednictwem platformy `Moodle` – **jedno archiwum `.zip`**[1] o nazwie typu `Nazwisko_Imie_PD2.zip`. W archiwum znajdować się powinien jeden katalog, `Nazwisko_Imie_PD2`, dopiero w którym umieszczone zostaną następujące pliki:

- `Nazwisko_Imie_PD2.R` (rozwiązania zadań)
- `Nazwisko_Imie_PD2.Rmd` (kod źródłowy raportu wraz z rozwiązaniami)
- `Nazwisko_Imie_PD2.html` (skompilowana wersja powyższego).

Nazwy plików nie powinny zawierać polskich liter diakrytyzowanych (przekształć $ą \rightarrow a$ itd.).

1 Zbiory danych

Będziemy pracować na uproszczonym zrzucie zanonimizowanych danych z serwisu https://travel.stackexchange.com/ (na marginesie: pełen zbiór danych dostępny jest pod adresem https://archive.org/details/stackexchange), który składa się z następujących ramek danych:

- Badges.csv.gz
- Comments.csv.gz
- PostLinks.csv.gz
- Posts.csv.gz
- Tags.csv.gz
- Users.csv.gz
- Votes.csv.gz

Przed przystąpieniem do rozwiązywania zadań zapoznaj się z ww. serwisem oraz znaczeniem poszczególnych kolumn we wspomnianych ramkach danych, zob. https://ia600107.us.archive.org/27/items/stackexchange/readme.txt

Przykładowe wywołanie — ładowanie zbioru `Tags`:

```
options(stringsAsFactors=FALSE)
# ww. pliki pobralismy do katalogu travel_stackexchange_com/
Tags <- read.csv("travel_stackexchange_com/Tags.csv.gz")
head(Tags)
```

Uwaga: Nazwy ramek danych po wczytaniu zbiorów powinny wyglądać następująco: `Badges`, `Comments`, `Tags`, `Posts`, `Users`, `Votes`, `PostLinks`.

[1] A więc nie: .rar, .7z itp.

2 Informacje ogólne

Rozwiąż poniższe zadania przy użyciu wywołań funkcji bazowych oraz tych, które udostępniają pakiety `dplyr` oraz `data.table` – nauczysz się ich samodzielnie; ich dokumentację znajdziesz łatwo w internecie. Każdemu z **5** poleceń SQL powinny odpowiadać cztery równoważne sposoby ich implementacji w R, kolejno:

1. `sqldf::sqldf()`;
2. tylko funkcje bazowe;
3. `dplyr`;
4. `data.table`.

Każde z zadań powinno być rozwiązane za pomocą funkcji:

1. `df_sql_i(df1, df2, ...)` ,
2. `df_base_i()` ,
3. `df_dplyr_i()` ,
4. `df_table_i()` ,

gdzie `i` to numer zadania, a `df1`, `df2`, ... potrzebne ramki danych (np. `df_sql_1(Users, Posts)`)

W raporcie koniecznie:

- upewnij się, że zwracane wyniki są ze sobą tożsame (ewentualnie z dokładnością do permutacji wierszy wynikowych ramek danych, zob. np. funkcję `dplyr::all_equal` lub `compare::compare`);
- podaj słowną interpretację (tzn. intuicyjne – „dla laika" – tłumaczenie) każdego zapytania;
- w każdym przypadku porównaj czasy wykonania napisanych przez Ciebie wyrażeń przy użyciu jednego wywołania `microbenchmark::microbenchmark()`, np.:

```
microbenchmark::microbenchmark(
   sqldf = df_sql_i(df1, df2, ...),
   base = df_base_i(df1, df2, ...),
   dplyr = df_dplyr_i(df1, df2, ...),
   data.table = df_table(df1, df2, ...)
```

Wszystkie rozwiązania umieść w jednym (estetycznie sformatowanym) raporcie knitr/Markdown.

Łączna ocena każdego z 5. zadań to 7 pkt (za poszczególne komponenty umieszczone w pliku `.R` oraz raporcie, tj. rozwiązanie i sprawdzenie równoważności wyników, pomiar i ocena czasu wykonania, opis słowny zapytań, komentarze) oraz 5 pkt za ogólną postać raportu (umieszczenie krótkiego wstępu, podsumowania, formatowanie dokumentu oraz wstawek `R`-a itp.).

3 Zadania do rozwiązania

```
--- 1)
SELECT Count, TagName
FROM Tags
WHERE Count > 1000
ORDER BY Count DESC
```

```sql
--- 2)
SELECT Location, COUNT(*) AS Count
FROM (
    SELECT Posts.OwnerUserId, Users.Id, Users.Location
    FROM Users
    JOIN Posts ON Users.Id = Posts.OwnerUserId
)
WHERE Location NOT IN ('')
GROUP BY Location
ORDER BY Count DESC
LIMIT 10
```

```sql
--- 3)
SELECT Year, SUM(Number) AS TotalNumber
FROM (
    SELECT
        Name,
        COUNT(*) AS Number,
        STRFTIME('%Y', Badges.Date) AS Year
    FROM Badges
    WHERE Class = 1
    GROUP BY Name, Year
)
GROUP BY Year
ORDER BY TotalNumber
```

```sql
--- 4)
SELECT
    Users.AccountId,
    Users.DisplayName,
    Users.Location,
    AVG(PostAuth.AnswersCount) as AverageAnswersCount
FROM
(
    SELECT
        AnsCount.AnswersCount,
        Posts.Id,
        Posts.OwnerUserId
    FROM (
            SELECT Posts.ParentId, COUNT(*) AS AnswersCount
            FROM Posts
            WHERE Posts.PostTypeId = 2
            GROUP BY Posts.ParentId
          ) AS AnsCount
    JOIN Posts ON Posts.Id = AnsCount.ParentId
) AS PostAuth
JOIN Users ON Users.AccountId=PostAuth.OwnerUserId
GROUP BY OwnerUserId
ORDER BY AverageAnswersCount DESC, AccountId ASC
LIMIT 10
```

```sql
--- 5)
SELECT Posts.Title, Posts.Id,
       STRFTIME('%Y-%m-%d', Posts.CreationDate) AS Date,
       VotesByAge.Votes
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 '2021' THEN 'new'
                    WHEN '2020' THEN 'new'
                    ELSE 'old'
                    END VoteDate,
                COUNT(*) AS Total
            FROM Votes
            WHERE VoteTypeId IN (1, 2, 5)
            GROUP BY PostId, VoteDate
        ) AS VotesDates
        GROUP BY VotesDates.PostId
        HAVING NewVotes > OldVotes
) AS VotesByAge ON Posts.Id = VotesByAge.PostId
WHERE Title NOT IN ('')
ORDER BY Votes DESC
LIMIT 10
```