DPRPy 2021/2022

Homework assigment no. 1 (max. = 35 p.)

Maximum grade: 35 p.

Deadline: **06.12.2021, 06.00** (28 days == 4 weeks).

Homework should be sent via the `Moodle` platform - **one archive `.zip`**[1] named

`Last-name_First-name_assgment_1.zip`

(one directory inside: `Last-name_First-name_assgment_1`), in which the following files will be placed:

- `Last-name_First-name_assgment_1.Rmd` (report prepared with `Markdown` / `knitr` containing task solutions, comments, etc.),
- `Last-name_First-name_assgment_1.html` (compiled to HTML version of the above).

1 Data description

We are working on a simplified dump of anonymised data from the website https://travel.stackexchange.com/ (by the way: full data set is available at https://archive.org/details/stackexchange), which consists of the following data frames:

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

Before starting to solve the problems familiarize yourself with the said service and data sets structure (e.g. what information individual columns represent), see https://archive.org/27/items/stackexchange/readme.txt.

Example: loading the set `Tags`:

```r
options(stringsAsFactors=FALSE)
# if files are saved at "pd1/travel_stackexchange_com/" directory
Tags <- read.csv("pd1/travel_stackexchange/Tags.csv.gz")
head(Tags)
```

2 Tasks description

Solve the following tasks using base functions calls and those provided by the `dplyr` and`data.table` packages - you will learn them on your own; their documentation (and tutorials) is easy to find online. Each of the **5**

[1]So not: .rar, .7z etc.

SQL queries should have four implementations in `R`:

1. `sqldf::sqldf()` – reference solution;
2. only base functions (1.5 p.);
3. `dplyr` (1.5 p.);
4. `data.table` (1.5 p.).

Make sure that the obtained results are equivalent (possibly with row permutation accuracy; up to 1 p. for each task). You can propose a function that implements relevant tests (e.g. based on `compare::compare()` or `dplyr::all_equal()`) - the results of such comparisions should be included in the final report. In addition, compare the execution times written by you in each case using one call to `microbenchmark :: microbenchmark ()` (0.5 p.), e.g .:

```
microbenchmark::microbenchmark(
   sqldf=sqldf_solution,
   base=base_functions_solution,
   dplyr=dplyr_solutions,
   data.table=datatable_solution
)
```

In addition, in each case, it is necessary to provide "intuitive" interpretation of each query (0.5 p.).

Put all solutions in one (nicely formatted) `knitr` / `Markdown` report. For rich code comments, discussion and possible alternative solutions you can obtained max. 2.5 p.

The solutons code should be included in the report.

3 SQL queries

```
--- 1)
SELECT
    Name,
    COUNT(*) AS Number,
    MIN(Class) AS BestClass
FROM Badges
GROUP BY Name
ORDER BY Number DESC
LIMIT 10
```

Important note: This query has been modify (26.11.2021). If you done this task in previous form. i.e.,

```
SELECT
    Name,
    COUNT(*) AS Number,
    MIN(Class) AS BestClass
FROM Badges
GROUP BY Name
ORDER BY Number DESC
LIMIT 10
```

it's fine and the max number of points will be granted (provided that solution is correct).

```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
    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
LIMIT 10
```

```sql
--- 4)
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
ORDER BY Year ASC
```

```sql
--- 5)
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 '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 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
```