PADPy 2020/2021

Praca domowa nr 2 (max. = 25 p.)

Termin oddania pracy: 21.12.2020, godz. 10:00.

Prace domowe należy przesłać za pośrednictwem platformy `Moodle` – **jedno archiwum `.zip`**[1] o nazwie typu `Nazwisko_Imie_NrAlbumu_Nick_pd2.zip`. W archiwum znajdować się powinien jeden katalog, `Nazwisko_Imie_NrAlbumu_Nick_pd2`, dopiero w którym umieszczone zostaną następujące pliki:

- `Nazwisko_Imie_NrAlbumu_Nick_pd2.ipynb` (rozwiązanie zadań)
- `Nazwisko_Imie_NrAlbumu_Nick_pd2.html` (ściągnięta wersja powyższego w formacie `html` - zob. `File -> Downlowad as -> html` w notatniku `Jupyter`).

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

  W nazwach plików wynikowych, `Nazwisko_Imie_NrAlbumu_Nick_pd2.zip`, `Nick` oznacza wybrany przez Państwa pseudonim, którego będziemy używać do publikowania wyników (inny niż nazwa użytkownika na platformie Github).

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

Przykładowe wywołanie — ładowanie zbioru `Tags`:

```
import pandas as pd
import numpy as np

Tags = pd.read_csv("travel_stackexchange_com/Tags.csv.gz",
                   compression = 'gzip')
Tags.head()
```

Przed przystąpieniem do rozwiązywania zadań zapoznaj się z ww. serwisem oraz znaczeniem poszczególnych kolumn we wspomnianych ramkach danych, zob. http://www.gagolewski.com/resources/data/travel_stackexchange_com/readme.txt.

[1] A więc nie: .rar, .7z itp.

Każdą z ramek danych należy wyeksportować do bazy danych SQLite przy użyciu wywołania metody `to_sql()` w klasie `pandas.DataFrame`. Dokładniej, pracę z bazą danych możemy pzeprowadzić w następujący sposób.

```
import os, os.path
import sqlite3
import tmpfile

# sciezka dostępu do bazy danych:
baza = os.path.join(tempfile.mkdtemp(), 'przyklad.db')
if os.path.isfile(baza): # jesli baza już istneje...
    os.remove(baza)      # ...usuniemy ja

conn = sqlite3.connect(baza)      # połączenie do bazy danych

Badges.to_sql("Badges", conn)     # importujemy ramkę danych do bazy danych
Comments.to_sql("Comments", conn)
PostLinks.to_sql("PostLinks", conn)
Posts.to_sql("Posts", conn)
Tags.to_sql("Tags", conn)
Users.to_sql("Users", conn)
Votes.to_sql("Votes", conn)

#
pd.read_sql_query("""
                  Zaptanie SQL
                  """, conn)
# ...
# rozwiązania zadan
# po skończonej pracy zamykamy połączenie
#
conn.close()
```

W szczególności należy zagwarantować, że w każdym przypadku wynik jest klasy `DataFrame` a nie `Series`.

2 Informacje ogólne

Rozwiąż poniższe zadania przy użyciu wywołań funkcji i metod z pakietu `pandas`. Każdemu z 3 poleceń SQL powinny odpowiadać dwa równoważne sposoby ich implementacji, kolejno:

1. wywołanie `pandas.read_sql_query("""zapytanie SQL""")`;
2. wywołanie ciągu „zwykłych” metod i funkcji z pakietu `pandas` (2 p.).

Upewnij się, że zwracane wyniki są ze sobą tożsame (ewentualnie z dokładnością do permutacji wierszy wynikowych ramek danych), por. np. metodę `.equals()` z pakietu `pandas` (1 p.). W razie potrzeby zaimplementuj własną funkcję sprawdzającą korzystając z funkcji i metod z pakietu `pandas`.

W każdym przypadku należy podać słowną (opisową) interpretację każdego zapytania (1 p.).

Dodatkowo wykorzystaj funkcje z pakietu `timeit` (lub dyrektywę `%timeit` w notatniku `Jupyter`) porównaj czasy wykonania napisanych przez Ciebie wyrażeń (0.5 p.).

Wszystkie rozwiązania umieść w jednym (estetycznie sformatowanym) raporcie `Jupyter`. Za bogate komentarze do kodu, dyskusję i ewentualne rozwiązania alternatywne można otrzymać max. 2.5 p.

3 Zadania do rozwiązania

```sql
--- 1)
SELECT Posts.Title, RelatedTab.NumLinks
FROM
    (SELECT RelatedPostId AS PostId, COUNT(*) AS NumLinks
     FROM PostLinks
     GROUP BY RelatedPostId) AS RelatedTab
JOIN Posts ON RelatedTab.PostId=Posts.Id
WHERE Posts.PostTypeId=1
ORDER BY NumLinks DESC
```

```sql
--- 2)
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
```

```sql
--- 3)
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
```

```sql
--- 4)
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
```

```
--- 5)
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
```