Praca projektowa nr 2

Sebastian Deręgowski

12 maja 2020r.

Spośród czterech funkcji najbardziej podobały mi się funkcje *dplyr*, były bardzo intuicyjne i łatwo się do nich konwertowało zapytania SQLa. Najmniej przepadałem za funkcjami *data.table*, których składnia była skomplikowana i przysporzyła mi sporo trudności.

Jeśli chodzi o analizę wydajności, to na pewno najmniej optymalna jest funkcja *sqldf*. Funkcje bazowe sprawdzały się dla małych wolumenów danych, ale przy większych były zdecydowanie wolniejsze. Biorąc to wszystko pod uwagę osobiście dla mnie najlepszym pakietem do obróbki danych wydaje się być *dplyr*.

Przy sprawdzaniu zgodności wyników używałem jedynie funkcji $anti_join$, z racji na to, że funkcja all_equal zwracała zazwyczaj wiele różnic, które nieestetycznie prezentowałyby się w raporcie. Różnice te zazwyczaj dotyczyły różnicy klas obiektów i były to błędy stricte estetyczne, a nie rzeczowe. Zdaję też sobie sprawę z tego, że implementacja poszczególnych funkcji może nie być optymalna. Wynika to głównie z tego, że pisałem je często równolegle metodą prób i błędów, a na koniec nie ujednolicałem ich do jednej postaci. Mimo to mam nadzieję, że zasadniczo wszystko jest w porządku.

Zadanie 1

Opis

Funkcje wyświetlają zadane cztery kolumny z ramki Posts i filtrują po tych rekordach, które spełniają trzy określone warunki.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_1(Posts),df_base_1(Posts))
```

```
## Joining, by = c("Title", "Score", "ViewCount", "FavoriteCount")
```

```
## [1] Title         Score         ViewCount     FavoriteCount
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_1(Posts),df_dplyr_1(Posts))
```

```
## Joining, by = c("Title", "Score", "ViewCount", "FavoriteCount")
```

```
## [1] Title         Score         ViewCount     FavoriteCount
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_1(Posts),df_table_1(Posts))
```

```
## Joining, by = c("Title", "Score", "ViewCount", "FavoriteCount")
```

```
## [1] Title         Score         ViewCount     FavoriteCount
## <0 rows> (or 0-length row.names)
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_1(Posts),
  base = df_base_1(Posts),
  dplyr = df_dplyr_1(Posts),
  table = df_dplyr_1(Posts)
)
```

```
## Unit: milliseconds
##   expr      min        lq       mean    median        uq      max neval
##  sqldf 130.8781 133.56560 138.123902 136.26520 140.29430 187.3433   100
##   base   1.6158   1.87085   3.316621   1.92575   2.02475  42.7899   100
##  dplyr   4.1211   4.60220   6.526648   4.98310   5.38190  56.0118   100
##  table   4.1187   4.66545   5.421276   4.93740   5.24995  42.4773   100
```

Zadanie 2

Opis

Funkcje ładują zbiór Tags, następnie złączają go ze zbiorami Posts i Users na podstawie wspólnych wartości poszczególnych kolumn, a następnie wyświetlają niektóre kolumny filtrując dodatkowo wiersze, w których ID użytkownika jest różne od -1 i sortując malejąco po wybranej kolumnie.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_2(Tags,Posts,Users),df_base_2(Tags,Posts,Users))
```

```
## Joining, by = c("TagName", "Count", "OwnerUserId", "Age", "Location", "DisplayName")
```

```
## [1] TagName     Count       OwnerUserId Age         Location    DisplayName
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_2(Tags,Posts,Users),df_dplyr_2(Tags,Posts,Users))
```

```
## Joining, by = c("TagName", "Count", "OwnerUserId", "Age", "Location", "DisplayName")
```

```
## [1] TagName     Count       OwnerUserId Age         Location    DisplayName
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_2(Tags,Posts,Users),df_table_2(Tags,Posts,Users))
```

```
## Joining, by = c("TagName", "Count", "OwnerUserId", "Age", "Location", "DisplayName")
```

```
## [1] TagName     Count       OwnerUserId Age         Location    DisplayName
## <0 rows> (or 0-length row.names)
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_2(Tags,Posts,Users),
  base = df_base_2(Tags,Posts,Users),
  dplyr = df_dplyr_2(Tags,Posts,Users),
  table = df_table_2(Tags,Posts,Users)
)
```

```
## Unit: milliseconds
##   expr      min        lq      mean   median        uq      max neval
##  sqldf 240.7595 245.14160 251.27957 247.4417 253.76445 291.3377   100
##   base  14.7235  15.29240  19.13968  15.6545  16.21825 136.6483   100
##  dplyr  28.6409  30.98615  33.02166  32.5214  34.04470  49.4704   100
##  table  16.7592  18.58285  25.00555  19.4219  21.29135  61.4509   100
```

Zadanie 3

Opis

Funkcje tworzą ramkę danych RelatedTab powstałą w wyniku wyselekcjonowania i zmienienia nazw niektórym kolumnom zbioru PostLinks. Następnie do tej ramki dołączana jest ramka Posts na podstawie wspólnych wartości poszczególnych kolumn. Następnie filtrowane są tylko te rekordy, których PostTypeId jest równy 1, a całość sortowana jest malejąco po kolumnie NumLinks.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_3(PostLinks,Posts),df_base_3(PostLinks,Posts))

## Joining, by = c("Title", "NumLinks")

## [1] Title    NumLinks
## <0 rows> (or 0-length row.names)

anti_join(df_sql_3(PostLinks,Posts),df_dplyr_3(PostLinks,Posts))

## Joining, by = c("Title", "NumLinks")

## [1] Title    NumLinks
## <0 rows> (or 0-length row.names)

anti_join(df_sql_3(PostLinks,Posts),df_table_3(PostLinks,Posts))

## Joining, by = c("Title", "NumLinks")

## [1] Title    NumLinks
## <0 rows> (or 0-length row.names)

microbenchmark::microbenchmark(
  sqldf = df_sql_3(PostLinks,Posts),
  base = df_base_3(PostLinks,Posts),
  dplyr = df_dplyr_3(PostLinks,Posts),
  table = df_table_3(PostLinks,Posts)
)

## Unit: milliseconds
##   expr      min        lq      mean    median        uq      max neval
##  sqldf 154.3970 158.24660 161.16482 160.26830 162.53600 187.4743   100
##   base  64.7715  65.96795  75.87064  66.82900  93.56340 187.9765   100
##  dplyr  21.0538  22.82720  25.16781  23.53575  24.56940  52.4959   100
##  table  11.7168  15.39540  20.53027  16.02490  16.60935  47.5170   100
```

Zadanie 4

Opis

Funkcje najpierw tworzą ramkę ValuableBadges, która powstała w wyniku przefiltrowania ramki Badges po konkretnych warunkach i wybraniu z niej dwóch kolumn. Następnie dołączona jest do niej ramka Users na podstawie zgodności wartości w zadanej kolumnie, a następnie wyselekcjonowane są wybrane kolumny i wyświetlane są rekordy o unikatowych wartościach.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_4(Users,Badges),df_dplyr_4(Users,Badges))
```

```
## Joining, by = c("Id", "DisplayName", "Reputation", "Age", "Location")
```

```
## [1] Id          DisplayName Reputation  Age         Location
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_4(Users,Badges),df_base_4(Users,Badges))
```

```
## Joining, by = c("Id", "DisplayName", "Reputation", "Age", "Location")
```

```
## [1] Id          DisplayName Reputation  Age         Location
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_4(Users,Badges),df_table_4(Users,Badges))
```

```
## Joining, by = c("Id", "DisplayName", "Reputation", "Age", "Location")
```

```
## [1] Id          DisplayName Reputation  Age         Location
## <0 rows> (or 0-length row.names)
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_4(Users,Badges),
  base = df_base_4(Users,Badges),
  dplyr = df_dplyr_4(Users,Badges),
  table = df_table_4(Users,Badges)
)
```

```
## Unit: milliseconds
##   expr      min        lq      mean    median        uq      max neval
##  sqldf 193.3737 195.44180 198.97242 197.26410 199.15330 225.8433   100
##   base   6.0193   6.33695  12.30917   6.52060   7.24885  38.5612   100
##  dplyr   9.3792  10.27945  12.46155  10.58355  11.09855  49.3097   100
##  table  11.0630  13.07500  17.95687  13.82735  15.00580  47.1367   100
```

Zadanie 5

Opis

Funkcje tworzą dwie ramki danych - UpVotesTab i DownVotesTab, obie powstałe w wyniku wyselekcjonowania odpowiednich rekordów z ramki Votes. Obie ramki zestawione są następnie razem ze sobą na podstawie wspólnego PostId i wyselekcjonowane po konkretnych kolumnach, dodatkowo puste wartości zostają zastąpione zerami.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_5(Votes),df_base_5(Votes))
```

```
## Joining, by = c("PostId", "UpVotes", "DownVotes")
```

```
## [1] PostId    UpVotes   DownVotes
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_5(Votes),df_dplyr_5(Votes))
```

```
## Joining, by = c("PostId", "UpVotes", "DownVotes")
```

```
## [1] PostId    UpVotes   DownVotes
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_5(Votes),df_table_5(Votes))
```

```
## Joining, by = c("PostId", "UpVotes", "DownVotes")
```

```
## [1] PostId    UpVotes   DownVotes
## <0 rows> (or 0-length row.names)
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_5(Votes),
  base = df_base_5(Votes),
  dplyr = df_dplyr_5(Votes),
  table = df_dplyr_5(Votes)
)
```

```
## Unit: milliseconds
##   expr       min        lq      mean    median        uq       max neval
##  sqldf  596.8741  627.3860  662.8915  646.7055  678.0223 1014.2524   100
##   base 1200.9190 1305.2728 1370.6233 1356.3228 1422.0358 1712.0094   100
##  dplyr  119.9389  141.8112  168.8460  160.7601  184.6311  299.1800   100
##  table  118.6782  143.7310  165.9446  161.3700  179.0509  300.4497   100
```

Zadanie 6

Opis

Funkcje dwukrotnie powtarzają algorytm funkcji z zadania nr 5, tworząc dwie ramki danych zawierające odpowiednio UpVotes i DownVotes. Następnie są one łączone w jedną wynikową ramkę, a różnice wartości UpVotes i DownVotes zostają wyświetlone jako Votes.

Sprawdzenie poprawności i porównanie czasów

```
anti_join(df_sql_6(Votes),df_base_6(Votes))
```

```
## Joining, by = c("PostId", "Votes")
```

```
## [1] PostId Votes
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_6(Votes),df_dplyr_6(Votes))
```

```
## Joining, by = c("PostId", "Votes")
```

```
## [1] PostId Votes
## <0 rows> (or 0-length row.names)
```

```
anti_join(df_sql_6(Votes),df_table_6(Votes))
```

```
## Joining, by = c("PostId", "Votes")
```

```
## [1] PostId Votes
## <0 rows> (or 0-length row.names)
```

```
microbenchmark::microbenchmark(
  sqldf = df_sql_6(Votes),
  base = df_base_6(Votes),
  dplyr = df_dplyr_6(Votes),
  table = df_dplyr_6(Votes)
)
```

```
## Unit: milliseconds
##   expr       min        lq      mean    median        uq       max neval
##  sqldf  841.1008  854.9329  888.3397  879.5542  910.0575 1047.3757   100
##   base 1231.1473 1306.2824 1347.9683 1348.3703 1385.1603 1569.8673   100
##  dplyr  261.0657  297.0381  330.2237  321.2543  335.6067  481.8990   100
##  table  259.2686  292.6527  323.8524  318.6314  344.5122  471.0094   100
```