Common data operations: MySQL, tidyverse/dplyr, and data.table

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Learning objectives:

- Familiarize you with the most common data operations on social and digital media data
- Introduce you to two packages to wrangle with data: dplyr (which has a very "clean" and easy-to-use syntax, but is slow on big data sets), and data.table (which is a bit messier to write up, but shows extremely good performance on large datasets)
- Show you alternatives - if they are available - using the SQL syntax

Prerequisites:

- access to the Research in Social Media SQL server.
- install the following packages: install.packages(c('RMySQL','data.table','tidyverse'))
- download the cheat sheets
  - dplyr and tidyr: https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
  - data.table
  - SQL

Let's now initialize the packages.

```
library(RMySQL) # loads the MySQL package
library(data.table) # loads the data.table package
library(dplyr) # loads the dplyr library
library(tidyr) # loads the tidyr library
```

. . . and load some data from the classes' MySQL server. Recall that the database holds a copy of daily data on the Top 200 Charts, as shown on https://spotifycharts.com.

```
database = dbConnect(MySQL(), user = 'student', password='rsm!2020',
                 dbname='rsm', host = 'sql.tilburg-digital.com')

dbListTables(database) # shows a list with tables in the database
```

```
## [1] "artists"  "echonest" "plays"    "tracks"
```

Let us also store a copy of the data in actual variables in R, so that we can use them with `dplyr` and `data.table`.

```
# Let's define a function to load some sample data from the SQL database server (the first X rows)
get_sample_data = function(tablename) {
  rs = dbSendQuery(database, paste0("SELECT * FROM ", tablename, " LIMIT 1000"))
  data = fetch(rs, n=-1)
}

# Let's define a function that will ease running queries on our MySQL database.
run_query = function(query) {
  rs= dbSendQuery(database, query) # runs the query
```

```
  data = fetch(rs, n=-1) # fetches n rows of the result (here: -1 indicates to fetch ALL rows)
  return(data) # returns the result back to the main program
}

artists = get_sample_data("artists")
tracks = get_sample_data("tracks")
plays = get_sample_data("plays")
```

```
## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 4 imported as
## numeric
```

```
# To use the data with data.table, let us convert them to "data.table".
dt_artists = data.table(artists)
dt_tracks = data.table(tracks)
dt_plays = data.table(plays)

# To use the data with dplyr, let's convert them to a so-called "tibble":
df_artists= tbl_df(artists)
df_tracks = tbl_df(tracks)
df_plays = tbl_df(plays)
```

1. Filtering/querying/subsetting

The first set of operations we're going to review are simple "filters", which sometimes are also referred to as querying ("searching"), or subsetting ("showing only a part of what is there").

There are two types of filters: filtering for particular rows, and filtering for particular columns.

Rows

Let's start querying for some observations in rows. For example, we could "filter" for all artists that have the name Adele.

SQL

```
run_query('SELECT * FROM artists WHERE artists.name = "Adele"')
```

```
##    id  name
## 1 130 Adele
```

dplyr

To use the `dplyr` syntax, we have to run our commands on the `df_` tables saved above.

```
filter(df_artists, name == 'Adele')
```

```
## # A tibble: 1 x 2
##      id name
##   <int> <chr>
## 1   130 Adele
```

data.table

To use the `data.table` syntax, we have to run our commands on the `dt_` tables savbed above.

```
dt_artists[name == 'Adele']
```

```
##     id  name
## 1: 130 Adele
```

Columns

Let's now proceed with "filtering" for particular columns. For example, we can only show the "name" column from the artist table, but this time, we show only the first 10 results.

Note that we will now combine the different commands in one code cell.

```
# SQL
run_query('SELECT name FROM artists LIMIT 10')
```

```
##                   name
## 1                  ???
## 2         #TocoParaVos
## 3               $hirak
## 4              $igmund
## 5          $uicideBoy$
## 6         'Til Tuesday
## 7             (G)I-DLE
## 8               *NSYNC
## 9                 -M-
## 10 04 Limited Sazabys
```

```
# dplyr/tidyverse
select(df_artists, name) # selects everything
```

```
## # A tibble: 1,000 x 1
##    name
##    <chr>
##  1 " ???"
##  2 "#TocoParaVos"
##  3 "$hirak"
##  4 "$igmund"
##  5 "$uicideBoy$"
##  6 "'Til Tuesday"
##  7 "(G)I-DLE"
##  8 "*NSYNC"
##  9 "-M-"
## 10 "04 Limited Sazabys"
## # ... with 990 more rows
```

```
select(df_artists, name) %>% top_n(10) # first 10 rows
```

```
## Selecting by name
```

```
## # A tibble: 10 x 1
##    name
##    <chr>
##  1 Ateyaba
##  2 Athena
##  3 Atip
##  4 Atitude 67
##  5 Atiye
##  6 ATkel
##  7 ATL
##  8 Atlantic Starr
##  9 Atlas Genius
## 10 Atle
```

```
# data.table
dt_artists[, c('name')] # selects everything
```

```
##                 name
##    1:            ???
##    2:   #TocoParaVos
##    3:         $hirak
##    4:        $igmund
##    5:    $uicideBoy$
##   ---
##  996:          ATkel
##  997:            ATL
##  998: Atlantic Starr
##  999:   Atlas Genius
## 1000:           Atle
```

```
dt_artists[, c('name')][1:10] # shows first 10 rows
```

```
##                   name
##  1:                ???
##  2:       #TocoParaVos
##  3:             $hirak
##  4:            $igmund
##  5:        $uicideBoy$
##  6:       'Til Tuesday
##  7:           (G)I-DLE
##  8:             *NSYNC
##  9:               -M-
## 10: 04 Limited Sazabys
```

2. Reshaping data

In wrangling with data, you frequently need to change the format of your data to "fit" the input requirements of some analysis methods you would like to use.

For example, let's retrieve some data the number of plays for Adele and Drake across all countries. At this stage, let's not worry too much about the query to retrieve that data (it's quite complicated). Instead, let's focus on the reshaping operations later.

```
bycountry = run_query("SELECT artists.name as name, country, SUM(streams) as total_plays FROM plays LEF
```

```
## Warning in .local(conn, statement, ...): Decimal MySQL column 2 imported as
## numeric
```

```
head(bycountry) # shows first six rows of result - Adele is there
```

```
##    name country total_plays
## 1 Adele      ar    19008266
## 2 Adele      at     3880581
## 3 Adele      au    52394668
## 4 Adele      be    14362457
## 5 Adele      bg      160013
## 6 Adele      bo      365184
```

```
tail(bycountry) # shows last six rows of result - Drake is there
```

```
##      name country total_plays
```

```
## 116 Drake      th     3432758
## 117 Drake      tr    47913723
## 118 Drake      tw    14578546
## 119 Drake      us  5740749631
## 120 Drake      uy     3004290
## 121 Drake      vn      556825
```

From long to wide

The data we're seeing is saved in the so-called "long" format - we have many rows to store the result of Adele and Drake, for each country.

Now let's suppose we wanted to prepare an overview for a music manager on the performance of these two artists across all countries. Further, let's suppose the manager is interested in learning in which country any of these two artists wasn't making the Top 200 at all!

We can give a quick answer to this by converting the "long" format to the so-called "wide" (which basically means: many columns) format.

Watch this! (Note we're going to convert the result from above to tibbles and data.tables first)

```
df_bycountry = tbl_df(bycountry)
dt_bycountry = data.table(bycountry)

# tidyverse/dplyr
result1 = spread(df_bycountry, country, total_plays)
result1
```

```
## # A tibble: 2 x 62
##   name      ar     at     au     be     bg     bo     br     ca     ch     cl
##   <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Adele 1.90e7 3.88e6 5.24e7 1.44e7 160013 3.65e5 5.62e7 3.33e7 9.16e6 1.54e7
## 2 Drake 4.99e7 1.77e7 3.73e8 4.70e7 967910 2.95e6 2.32e8 6.54e8 4.09e7 6.11e7
## # ... with 51 more variables: co <dbl>, cr <dbl>, cy <dbl>, cz <dbl>, de <dbl>,
## #   dk <dbl>, do <dbl>, ec <dbl>, ee <dbl>, es <dbl>, fi <dbl>, fr <dbl>,
## #   gb <dbl>, gr <dbl>, gt <dbl>, hk <dbl>, hn <dbl>, hu <dbl>, id <dbl>,
## #   ie <dbl>, il <dbl>, is <dbl>, it <dbl>, jp <dbl>, lt <dbl>, lu <dbl>,
## #   lv <dbl>, mt <dbl>, mx <dbl>, my <dbl>, ni <dbl>, nl <dbl>, no <dbl>,
## #   nz <dbl>, pa <dbl>, pe <dbl>, ph <dbl>, pl <dbl>, pt <dbl>, py <dbl>,
## #   ro <dbl>, se <dbl>, sg <dbl>, sk <dbl>, sv <dbl>, th <dbl>, tr <dbl>,
## #   tw <dbl>, us <dbl>, uy <dbl>, vn <dbl>
```

```
# data.table
result2 = dcast(dt_bycountry, name~country, value.var='total_plays')
result2
```

```
##     name       ar       at        au       be     bg      bo        br
## 1: Adele 19008266  3880581  52394668 14362457 160013  365184  56155124
## 2: Drake 49931955 17743237 372982497 47010442 967910 2953007 232142448
##           ca       ch       cl       co       cr    cy      cz        de
## 1:  33260300  9163207 15408229 12130779  8465703 16180 2956355  57850369
## 2: 654086247 40890658 61139387 35613761 22692854 82516 9064946 343009280
##           dk       do       ec      ee        es       fi        fr         gb
## 1:  23680490  1558688  2881672  478838  42898066 10389074  17872035  147393684
## 2: 106447172 11978584 10538562 2583835 135401801 37647293 155238124 1035361207
##         gr      gt       hk      hn      hu       id       ie      il      is
## 1:  894783 1170657  5589452  891552 1824451 17360916 12259353   26603  751725
```

```
## 2: 9208863 7501238 11935528 6535679 8729258 34195234 69540229 4875074 7861147
##           it      jp      lt     lu      lv     mt        mx       my     ni
## 1:  24600039  383196  611429  74855  574232 233574  69231917 14014592 180895
## 2: 113913687 8546331 3399787 444274 3509961 909267 275768406 26622964 781399
##           nl       no       nz      pa       pe        ph       pl       pt
## 1:  48434000 43796672 18731362  885138  5696515  40351708 13438508  6448729
## 2: 227503357 94745136 81536699 7355619 26245797 126106766 37272065 43150940
##         py      ro        se       sg      sk      sv      th       tr       tw
## 1:  657011      NA  91612868 15369718  452077  538775    9828 10690389  8385285
## 2: 4560931 3308516 196275811 34789093 3021548 4180983 3432758 47913723 14578546
##            us      uy     vn
## 1:  323100434 1057208 211490
## 2: 5740749631 3004290 556825
```

From wide to long

Sometimes, we also just wish to go back from wide to long. Let's use the results from above as input.

```
# tidyverse/dplyr
result1b = gather(result1, 'country', 'total_plays', -name)
result1b
```

```
## # A tibble: 122 x 3
##    name  country total_plays
##    <chr> <chr>         <dbl>
##  1 Adele ar          19008266
##  2 Drake ar          49931955
##  3 Adele at           3880581
##  4 Drake at          17743237
##  5 Adele au          52394668
##  6 Drake au         372982497
##  7 Adele be          14362457
##  8 Drake be          47010442
##  9 Adele bg            160013
## 10 Drake bg            967910
## # ... with 112 more rows
```

```
# data.table
result2b = melt(result2, c('name'), variable.name='country', value.name = 'total_plays')
result2b
```

```
##       name country total_plays
##   1: Adele      ar    19008266
##   2: Drake      ar    49931955
##   3: Adele      at     3880581
##   4: Drake      at    17743237
##   5: Adele      au    52394668
##  ---
## 118: Drake      us  5740749631
## 119: Adele      uy     1057208
## 120: Drake      uy     3004290
## 121: Adele      vn      211490
## 122: Drake      vn      556825
```

Uniting columns into one

We can use code to unite columns into one. For example, let's create an overview about the Top 3 countries in terms of plays for each of the artists. Let's first create an indicator variable for the Top 3, filter on it, and then combine the resulting country names in one column.

```
# tidyverse
# to be added - anybody knows how?

# data.table
setorderv(result2b, c('name', 'total_plays'), order = c(1, -1), na.last=T) # sort in ascending order by
result2b[, rank := 1:.N, by = c('name')] # create rank positions
# filter for ranks
result2b[rank <= 3]
```

```
##     name country total_plays rank
## 1: Adele      us   323100434    1
## 2: Adele      gb   147393684    2
## 3: Adele      se    91612868    3
## 4: Drake      us  5740749631    1
## 5: Drake      gb  1035361207    2
## 6: Drake      ca   654086247    3
```

```
# let's create a variable which combines the country name with the streams (in brackets)
result2b[, country_streams := paste0(country, ' (', total_plays, ')')]

# let's now "concatenate" the top three countries
result3 = result2b[rank <= 3, list(paste0(country_streams, collapse = ', ')),by = c('name')]
```

Separating column values into multiple columns

Now let's suppose we only have access to the comma-separated values in the country_streams column in the example above. How would we separate those values back into individual columns?

Watch how here:

```
# data.table
tmp=result3[, list(strsplit(V1, ', ', fixed=T)[[1]]), by = c('name')]
tmp[, strsplit(V1, ' ')]
```

```
##             V1          V2         V3           V4           V5          V6
## 1:          us          gb         se           us           gb          ca
## 2: (323100434) (147393684) (91612868) (5740749631) (1035361207) (654086247)
```

```
tmp[, country:=sapply(strsplit(V1, ' '), function(x) x[1])]
tmp[, total_streams := sapply(strsplit(V1, ' (', fixed=T), function(x) x[2])]
tmp[, total_streams := as.numeric(gsub(')', '', total_streams))]
tmp[, V1:=NULL]
result4 = tmp
```

3. **Summarizing/aggregating data**
4. **Make new variables**
5. **Combine/join/merge data sets**
6. **Format conversion**