

# Language-Agnostic Data Frame Wrangling Exercises

Marek Gagolewski [marek@gagolewski.com]

Last update: 2018-05-02

Copyright © 2017-2018, Marek Gagolewski. All rights reserved. Do not redistribute.

---

All the datasets referenced below can be downloaded from <http://gagolewski.com/resources/>.

80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson, *Exploratory data mining and data cleaning*, Wiley, 2003)

1. In the `flights` table from the `nycflights13` database, select all columns between `year` and `day` (inclusive), without referring to column numbers explicitly (the code should work if we randomly rearrange the columns too). Then select all columns except those between `year` and `day` (inclusive).
2. In `tips`, select data on male customers where total bill was in `[10, 20]`.
3. In `tips`, select data from Saturday and Sunday with `tip > $5`.
4. In `winequality-all`, leave out only white wines. Partition the resulting data frame randomly into two data frames: `wines_train` (80% of the rows) and `wines_test` (remaining 20%).
5. Given `fueleconomy::vehicles`, convert the values in `cty` (city-) and `hwy` (highway-fuel economy – given in mpg) to l/100 km. Then add new columns `z_cty` and `z_hwy`, that give z-scores of `cty` and `hwy` (i.e., standardize these columns). Moreover, add two other columns `z_cty_rel` and `z_hwy_rel`, which denote the corresponding z-scores relative to (grouped by) `class`.
6. Consider the `nycflights13` database which consists of 5 tables: `flights`, `weather`, `planes`, `airlines`, `airports`. Fetch the results equivalent to the following SQL queries.
  1. `SELECT DISTINCT engine FROM planes`
  2. `SELECT DISTINCT type, engine FROM planes`
  3. `SELECT COUNT(*), engine FROM planes GROUP BY engine`
  4. `SELECT COUNT(*), engine, type FROM planes GROUP BY engine, type`
  5. `SELECT MIN(year), AVG(year), MAX(year), engine, manufacturer FROM planes GROUP BY engine, manufacturer`
  6. `SELECT * FROM planes WHERE speed IS NOT NULL`
  7. `SELECT tailnum FROM planes WHERE seats BETWEEN 150 AND 190 AND year >= 2012`
  8. `SELECT * FROM planes WHERE manufacturer IN ("BOEING", "AIRBUS", "EMBRAER") AND seats>390`
  9. `SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER BY year ASC, seats DESC`
  10. `SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER BY seats DESC, year ASC`
  11. `SELECT manufacturer, COUNT(*) FROM planes WHERE seats > 200 GROUP BY manufacturer`
  12. `SELECT manufacturer, COUNT(*) FROM planes GROUP BY manufacturer HAVING COUNT(*) > 10`

13. `SELECT manufacturer, COUNT(*) FROM planes WHERE seats > 200 GROUP BY manufacturer HAVING COUNT(*) > 10`
  14. `SELECT manufacturer, COUNT(*) AS howmany FROM planes GROUP BY manufacturer ORDER BY howmany DESC LIMIT 5`
  15. `SELECT * FROM flights LEFT JOIN planes ON flights.tailnum=planes.tailnum`
  16. `SELECT planes.*, airlines.* FROM  
 (SELECT DISTINCT carrier, tailnum FROM flights) AS cartail  
 INNER JOIN planes ON cartail.tailnum=planes.tailnum  
 INNER JOIN airlines ON cartail.carrier=airlines.carrier`
  17. `SELECT flights2.*, weather2.atemp, weather2.ahumid, weather2.apressure FROM  
 (SELECT * FROM flights WHERE origin='EWR') AS flights2  
 LEFT JOIN  
 (SELECT year, month, day, AVG(temp) AS atemp,  
 AVG(humid) AS ahumid, AVG(pressure) AS apressure  
 FROM weather WHERE origin='EWR' GROUP BY year, month, day) AS weather2  
 ON flights2.year=weather2.year  
 AND flights2.month=weather2.month  
 AND flights2.day=weather2.day`
7. With the weather data frame from `nycflights13`:
- Convert temperature to Celsius.
  - Compute daily mean temperatures for the JFK airport. If some hourly temperature measurements is missing, linearly interpolate between the preceding and following non-missing data, e.g., a temperature sequence of `[..., 10, NaN, NaN, 40, ...]` should be transformed to `[..., 10, 20, 30, 40, ...]`.
  - Present the daily mean temperatures on a plot. The x-axis labels should be human-readable and intuitive.
  - Choose days with greater mean temperature than in the preceding day.
  - Find 5 hottest days.
8. Let  $A = \text{some\_birth\_dates1}$ ,  $B = \text{some\_birth\_dates2}$ , and  $C = \text{some\_birth\_dates3}$ . Assume the `name` column can be used as the primary key, i.e., it uniquely identifies each row in the data frame. Determine  $A \cup B$  (union),  $A \cup B \cup C$ ,  $A \cap B$  (intersection),  $A \cap C$ ,  $A \setminus B$  (difference).
9. Determine the union, intersection, and symmetric difference of the `some_wines1` and `some_wines2` datasets. Assume that no single column can be used as primary key.
10. Stack (melt) the `world_phones` dataset. In other words, convert:

##		N.Amer	Europe	Asia	S.Amer	Oceania	Africa	Mid.Amer
## 1951	45939	21574	2876	1815	1646	89	555	
## 1956	60423	29990	4708	2568	2366	1411	733	
## 1957	64721	32510	5230	2695	2526	1546	773	
## 1958	68484	35218	6662	2845	2691	1663	836	
## 1959	71799	37598	6856	3000	2868	1769	911	
## 1960	76036	40341	8220	3145	3054	1905	1008	
## 1961	79831	43173	9053	3338	3224	2005	1076	

to:

##	year	region	value
## 1	1951	N.Amer	45939
## 2	1956	N.Amer	60423
## 3	1957	N.Amer	64721
## 4	1958	N.Amer	68484

```
## 5 1959 N.Amer 71799
## 6 1960 N.Amer 76036
## 7 1961 N.Amer 79831
## 8 1951 Europe 21574
## 9 1956 Europe 29990
## 10 1957 Europe 32510
## 11 1958 Europe 35218
## ...
## 46 1958 Mid.Amer 836
## 47 1959 Mid.Amer 911
## 48 1960 Mid.Amer 1008
## 49 1961 Mid.Amer 1076
```

On a side note, here we deal with data from a *fully crossed design* experiment (all combinations of two grouping variables are present).

11. Unstack (cast) the molten `world_phones` dataset.
12. Unstack the molten dataset but first remove 5 random rows and then randomly permute all the remaining rows (*incomplete cross design*).
13. Unstack the `titanic` dataset. Represent it as a 4-dimensional matrix. Compute the sums of observations over different axes and their combinations.
14. Consider the `nasaweather_glaciers` data frame. All glaciers are assigned 11/12-character unique identifiers. The ID number is assigned to the glacier as defined by the WGMS convention that forms the glacier ID number by combining the following five elements. Extract all of them and store them as independent columns in the data frame.
  - 2-character political unit
  - 1-digit continent code
  - 4-character drainage code
  - 2-digit free position code
  - 2-3-digit local glacier code
15. Consider the `nycflights13_weather` data frame. Create the `time_hour` column (ignore the existing one) of type date-time based on `year`, `month`, `day`, and `hour`. Note that Eastern Standard Time (EST) is 5 hours behind UTC and that Eastern Daylight Time (EDT) is used in the summer.
16. Take the above data frame. Convert the `time_hour` column to plain text. Use the ISO 8601 `date-T-time-timezoneshift` format like `2017-01-17T13:29:24+00:00`.
17. Consider the `birth_dates` data set. Write a function that returns the names of people which – at a given date – have already been born but are less than 16 years of age.
18. Cleanse the `warsaw_weather` dataframe.