Data Engineering – Project 3: Data cleaning

March 30, 2023

1   Introduction

Welcome to the third project, which will be devoted to data cleaning.

As before, they will require that you load a dataset and process it as required, saving the indicated file. This time, we will focus on loading datasets and some basic data cleaning operations like type casting, value filtering and mapping, as well as using categories and encodings.

Note: Please do not submit your project for grading until the announcement that it is ready.

2   Exercises

2.1   Exercise 1: Load data

There are three homogenous JSON files called:

- `proj3_data1.json`,
- `proj3_data2.json`,
- `proj3_data3.json`.

Load all these files into one big DataFrame, with an index containing a consistent sequence of numbers, e.g. $0, 1, 2, ...$. Keep the original column structure.

Save the DataFrame to `ex01_all_data.json`.

2.2   Exercise 2: Missing values

Produce a CSV file containing column names and the number of missing values in these columns. Only include the columns which have any missing values, e.g.:

```
doors,2
fuel_consumption,4
```

Save the file as `ex01_no_nulls.csv`.

2.3   Exercise 3: Applying functions

The file `proj3_params.json` contains a dictionary with certain parameters which will be important in the following exercises. Load the file so that it is available later on.

Add a column called `description` to the DataFrame you created in the previous exercise. It should contain the values from columns listed as the `concat_columns` parameter, separated with spaces.

E.g., if `concat_columns` is `['make', 'model', 'engine']`, for the following record:

- make: Audi,
- model: A3,
- engine: 1.5 TFSI,

the `description` column should contain the string "Audi A3 1.5 TFSI".

Save the DataFrame as `ex03_descriptions.json`.

2.4 Exercise 4: Joining datasets

Load the file `proj3_more_data.json`.

Perform a join between the DataFrame you obtained in the previous exercise and that loaded from the new file, using the column indicated as the `join_column` parameter as the join column. Make sure the new DataFrame contains all rows from the DataFrame created in ex. 3, even those that do not have a corresponding record from the new one.

Save the new DataFrame as `ex04_joined.json`.

2.5 Exercise 5: Iterating over DataFrames

Create separate JSON file for each record in your DataFrame obtained in the previous exercise. The JSON record should include all fields apart from `description`.

The file name for each should be based on the value of `description` and follow the `ex05_(description).json` pattern. The value of `description` should be converted to lowercase, with spaces replaced with underscores.

For example, for the record with „Audi A3 1.5 TFSI" as the `description`, the file name should be `ex05_audi_a3_1.5_tfsi.json`.

Now repeat the following task, but make sure that columns listed in the `int_columns` parameter are saved as integers, not floats. Also make sure that the JSON file does not contain `NaN` values, as they are not allowed by the specification (only `null` is, and that should be used).

Save the files in a similar way as before, but now use the `ex05_int_(description).json` pattern; e.g., for the example above, the file name should be `ex03_int_audi_a3_1.5_tfsi.json`.

2.6 Exercise 6: Aggregation

The `aggregations` parameter contains the aggregations which should be performed for the data, in the form of tuples, where the first element is the column name and the second element is the aggregation function, e.g.:

```
('displacement', 'min'),
('displacement', 'max'),
('fuel_consumption', 'mean')
```

Create a JSON file called `ex06_aggregations.json` which contains the aggregated values from your DataFrame, in the form of a dictionare, where the keys are composed from the aggregation function and the column name, joined with an underscore, e.g.:

```
{
    "min_displacement": 875.0,
    "max_displacement": 3000.0,
    "mean_fuel_consumption": 6.414285714285714
}
```

2.7 Exercise 7: Grouping

Group the values from your DataFrame by the column indicated as the `grouping_column` parameter. Create a DataFrame which contains the mean values, for all groups and all numerical columns, but only for groups containing more than 1 record.

Save the result as `ex07_groups.csv`, including the header and the index.

2.8 Exercise 8: Reshaping data

Create a DataFrame which contains rows for the column indicated by the `pivot_index` parameter, columns for all values found in the `pivot_columns` parameter, and values taken from the column indicated as the `pivot_values` parameter.

If the original DataFrame contains more than one record for a given index/column combination, the maximum value should be used.

An example result could look like this:

make	diesel	gasoline	hybrid
Audi	nan	6	nan
Ford	nan	10.2	nan
Opel	nan	5.5	nan
Peugeot	nan	5.6	nan
Porsche	nan	11.8	nan
Renault	6.5	nan	nan
Volkswagen	6	5	nan
Volvo	nan	nan	6.6

(Please note that your DataFrame will have a MultiIndex header, which is not displayed in the table above.)

Save the DataFrame as `ex08_pivot.pkl`.

Transfrorm your DataFrame, so that, for every row, all of the columns are expanded to several rows – i.e., transform a *wide* DataFrame to a *long* one. Please use the list of columns in the `id_vars` parameter as record identifiers.

For instance, if `id_vars` contains `['make', 'model']`, the set of rows for a single record could look like this:

	make	model	variable	value
0	Audi	A3	body_type	hatchback
1	Audi	A3	doors	5.0

	make	model	variable	value
2	Audi	A3	top_speed	220
3	Audi	A3	acceleration	7.2
4	Audi	A3	fuel_consumption	6.0
5	Audi	A3	engine	1.5 TFSI
6	Audi	A3	description	Audi A3 1.5 TFSI
7	Audi	A3	displacement	1498.0
8	Audi	A3	horsepower	150.0
9	Audi	A3	fuel_type	gasoline
10	Audi	A3	cylinders	4.0
11	Audi	A3	emissions_class	Euro 6d

Save the result as `ex08_melt.csv`. Include the header, but *not* the index.

Load the `proj3_statistics.csv` file, which contains columns where two parameters are separated by an underscore character, like this:

Country	Audi_2019	Audi_2020	BMW_2019	BMW_2020
Poland	12	14	21	25
Germany	24	26	31	35
France	20	22	29	33
Spain	16	18	25	29
Italy	18	20	27	31

Assume that:

- the first column contains a grouping variable (in the example above, `Country`),
- the *prefixes* of the column labels are values from the column in your DataFrame indicated by the `pivot_index` parameter,

Transform the loaded data, so that it only has columns for the *prefixes*, and the *suffixes* form part of a MultiIndex with the original first column, e.g.:

	Audi	BMW	Volkswagen	Renault
('Poland', 2019)	12	21	32	22
('Germany', 2019)	24	31	44	17
('France', 2019)	20	29	36	28
('Spain', 2019)	16	25	40	19
('Italy', 2019)	18	27	48	31
('Poland', 2020)	14	25	36	20
('Germany', 2020)	26	35	48	15
('France', 2020)	22	33	40	26
('Spain', 2020)	18	29	44	17
('Italy', 2020)	20	31	52	29

Save the DataFrame to `ex08_stats.pkl`.