Data Engineering – Project 1: Pandas Basics

March 6, 2025

1 Introduction

Welcome! This is the first set of project exercises for the *Data Engineering* course.

Your job in this course will be to write rather simple programs, according to the instructions and specification presented in each exercise. Most of them will involve loading data from a file, performing some transformations or analytics, and writing the results to some other file.

You should submit your work via the provided Git repository in the Gitlab project that was provided to you. It will then be checked using scripts running on the server (via a CI/CD pipeline), and the results will be added to your repository as a JSON file. Today, in the first project, there is no limit of tries, but these will be limited in the subsequent ones. The number of tries will be always stated in the instructions.

Usually, a file will be provided as sample input for your program, but it is *not* the exact file your program will be tested with. Of course, the test input files will adhere to the description in the exercise, but don't make any other assumption regarding their structure and contents. The test file may change in time, but will always comply with the specification.

You will have 2 weeks, starting with the date of your project classes, to submit your work.

2 Instructions

Below, you will find several exercises with precise specifications. Please write one Python program which performs all the specified tasks in a sequence, producing the desired results.

It is probably a good idea for your program to start its life as a Jupyter Notebook. However, the desired form is a Python program in a `.py` file. In JupyterLab, you can use the *Save and Export Notebook As… > Executable Script* menu option to produce such a file. However, please verify that the file runs correctly, as it is easy to omit a possible mistake – remember that Jupyter kernels maintain their state, and that the cells are not always run sequentially.

After you're done, please commit your solution to your respository as `project01/project01.py`.

3 Technical stuff

Your assignment will be checked using a Docker container from a custom-built image.

Currently, the image uses the following software packages:

- Python version: 3.13.2 (main, Feb 25 2025, 05:25:21) [GCC 12.2.0]

- NumPy version: 2.2.3
- Pandas version: 2.2.3

4 Exercises

4.1 Exercise 1: Column information

3 points

File `proj1_ex01.csv` is a properly formed CSV file, with fields separated using commas (`,`) and with column headers. Load it into a DataFrame.

Create a file called `proj_ex01_fields.json`, which contains information *all* of the columns in the file you read. The file should contain an array of dictionaries with the following items:

- column name (key: `name`),
- percentage of missing values (key: `missing`, values in the range $[0.0, 1.0]$),
- data type as a string with the following values (key: `type`:
  - `int` for integer types,
  - `float` for floating-point types,
  - `other` for all other types.

An example JSON file could look like this:

```
[
    {
        "name": "id",
        "missing": 0.0,
        "type": "int"
    },
    {
        "name": "title",
        "missing": 0.2,
        "type": "other"
    },
    {
        "name": "result",
        "missing": 0.73,
        "type": "float"
    }
]
```

4.2 Exercise 2: Value statistics

2 points

Compute statistics for all columns in your dataframe.

For numeric columns include:

- the count of non-empty values (`count`),
- the average (`mean`),

- the standard deviation (`std`),
- the minimum (`min`) and maximum (`max`) values,
- the the 25th, 50th, and 75th percentiles (attribute names: `25%`, `50%` and `75%`, respectively).

For non-numeric columns include:

- the count of non-empty values (`count`),
- the number of unique values (`unique`),
- the most common value (`top`) and its frequency (number of occurences; `freq`).

Save the result to a JSON file called `proj1_ex02_stats.json` which contains a dictionary at the top level; the keys in the dictionary are column names, and the values are dictionaries with keys as described above, e.g.:

```
{
    "some_number":{
        "count":6.0,
        "mean":-0.5009940002,
        "std":0.8839385203,
        "min":-1.5552904133,
        "25%":-1.2470386925,
        "50%":-0.4162433767,
        "75%":0.1799426841,
        "max":0.5271122589
    },
    "some_string":{
        "count":7,
        "unique":3,
        "top":"good",
        "freq":3,
    }
}
```

In the inner dictionaries, keys with `null` values are allowed, e.g. a dictionary for a numeric column may contain the `unique` and `top` attributes.

4.3 Exercise 3: Column names

5 points

Rename ("normalize") the columns in the dataframe, so that they (sort of) follow the PEP 8 guidelines for variable names.

Apply the following rules:

- keep only characters which belong to the `[A-Za-z0-9_ ]` class (capital and small letters, digits, underscore and space),
- convert all letters to lowercase,
- replace all spaces with underscores (`_`).

Make the changes in your DataFrame persistent.

Save the DataFrame with the new columns to `proj1_ex03_columns.csv` (don't include the index).

4.4 Exercise 4: Output formats

3 points (1 for each format)

Write the data in the DataFrame to various output formats.

Create an MS Excel file called `proj1_ex04_excel.xlsx`, which contains the column headers, but not the index values.

Create a JSON file called `proj1_ex04_json.json`, which contains an array of rows stored as dictionaries, each with the DataFrame columns as keys (and values as values, obviously), e.g.:


```
[
    {
        "one":0.3485539245,
        "two":"-0.14509562920877161",
        "three":"-0.012336991474672475",
        "four":9,
        "five":"red",
        "six":"good",
        "seven":"quarrelsome",
        "eight":"2016-05-26 09:33:42"
    },
    {
        "one":-1.4938530178,
        "two":"0.12436946488785079",
        "three":"1.4611100361038865",
        "four":4,
        "five":"red",
        "six":"bad",
        "seven":"doctor",
        "eight":"2016-12-03 18:55:52"
    }
]
```


Create a pickle file called `proj1_ex04_pickle.pkl` with the DataFrame.

4.5 Exercise 5: Selecting rows and columns

4 points

Load the DataFrame pickled in file `proj1_ex05.pkl`.

Select the following items from the DataFrame:

- the 2nd and 3rd columns (regardless of their names),
- rows whose index values begin with the letter `v`.

Save the result to a Markdown table stored in file `proj1_ex05_table.md`. Include the result, but don't put anything in cells with missing values (i.e. prevent `nan` from being printed there).

4.6 Exercise 6: Flattening data

3 points

Pandas DataFrames are two-dimensional structures. However, data in JSON files often has a hierarchical structure, e.g. objects (dictionaries) are nested within objects.

File `proj1_ex06.json` contains an array with such hierarchical objects (the structure of each array element is the same).

Based on the data in the file, using pd.json_normalize() create a Pandas DataFrame, which contains a flattened version of the data. For nested dictionaries, the column names should have the keys separated using dots (`.`). E.g., for the following entry:

```
[
    {
        "brand": "Audi",
        "name": "Q5",
        "model": 2023,
        "engine": {
            "type": "Diesel",
            "displacement": "2.0L",
            "power": "190 hp",
            "environmental": {
                "euro": 6,
                "filter": "DPF"
            }
        }
    },
]
```

the resulting columns should be:

- `brand`,
- `model`,
- `year`,
- `engine.type`,
- `engine.displacement`,
- `engine.power`,
- `engine.environmental.euro`,
- `engine.environmental.filter`.

Save the resulting DataFrame to pickle file `proj1_ex06_pickle.pkl`.