Rush 00 – Piscine Python for Data Science

MovieLens Analytics

Summary: This rush will help you to strengthen the skills acquired in the previous days.

Contents

Chapter I

Preamble

Why do we like movies? What makes them so attractive to us?

Even though the movie is a relatively modern thing for humanity, it has a pretty old mechanism inside – it is the story.

People have loved stories since ancient times. Think about them as a universal container that effectively transfers some useful information from a source to a person. By sparkling emotions and imagination in us, it establishes a good connection and packages information in a way that can be easily consumed by a human being. Stories were crucial for surviving to our ancestors. Stories contain the personal experience that can be applied to your life. For example, you may discover that some areas around your village are pretty dangerous. Or there are some really good places to gather mushrooms.

Our attention to stories has survived through the centuries. If a speaker starts their presentation by telling a story, they catch our attention. We love books. We love music and songs. We love movies.

How can you use stories in data science? Good reports have elements of storytelling. Try to tell a story by your analysis.

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further we use `Python 3` as the only correct version of Python.
- The python files for python exercises (d01, d02, d03) must have a block in the end: `if __name__ == '__main__'`.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google.
- Remember to discuss on the Intra Piscine forum.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Specific instructions of the day

- No code in the global scope. Use functions!
- Any exception not caught will invalidate the work, even in the event of an error that was asked you to test.
- The interpreter to use is `Python 3`.
- Any built-in function is allowed.
- You can import the following libraries: `os`, `sys`, `urllib`, `requests`, `beautifulsoup`, `json`, `pytest`, `collections`, `functools`.
- Use `Jupyter Notebook` for creating the report.

Chapter IV

Mandatory part

In this rush, you are going to work on your own analytical report. You will analyze data from the MovieLens database. By the end of the rush, you will have two files: `movielens_analysis.py` and `movielens_report.ipynb`. In the first file, you will need to create your own module with classes and methods. In the second file, you will create the report itself using only your module.

Module

Remember that the goal of the rush is to strengthen your skills. Try to use as much as you can from what you have learned from the previous days.

1. Use the data from `D04E05`. If you have already removed it, redownload it, please.

2. Read the `README.txt` very carefully. Focus on the file structures.

3. In your module, you will need to create 4 classes corresponding to 4 files from the data and 1 class for testing.

4. The classes and methods below are obligatory but you can add to them whatever suits your needs.

a. Class Ratings

```python
class Ratings:
    """
    Analyzing data from ratings.csv
    """
    def __init__(self, path_to_the_file):
        """
        Put here any fields that you think you will need.
        """
    class Movies:
        def dist_by_year(self):
            """
            The method returns a dict where the keys are years and the values are counts.
            Sort it by years ascendingly. You need to extract years from timestamps.
            """
            return ratings_by_year

        def dist_by_rating(self):
            """
            The method returns a dict where the keys are ratings and the values are counts.
            Sort it by ratings ascendingly.
            """
            return ratings_distribution

        def top_by_num_of_ratings(self, n):
            """
            The method returns top-n movies by the number of ratings.
            It is a dict where the keys are movie titles and the values are numbers.
            Sort it by numbers descendingly.
            """
            return top_movies

        def top_by_ratings(self, n, metric=average):
            """
            The method returns top-n movies by the average or median of the ratings.
            It is a dict where the keys are movie titles and the values are metric values.
            Sort it by metric descendingly.
            The values should be rounded to 2 decimals.
            """
            return top_movies

        def top_controversial(self, n):
            """
            The method returns top-n movies by the variance of the ratings.
            It is a dict where the keys are movie titles and the values are the variances.
            Sort it by variance descendingly.
            The values should be rounded to 2 decimals.
            """
            return top_movies

    class Users:
        """
        In this class, three methods should work.
        The 1st returns the distribution of users by the number of ratings made by them.
        The 2nd returns the distribution of users by average or median ratings made by them.
        The 3rd returns top-n users with the biggest variance of their ratings.
        """
```

b. Class Tags

```
class Tags:
    """
    Analyzing data from tags.csv
    """
    def __init__(self, path_to_the_file):
        """
        Put here any fields that you think you will need.
        """
    def most_words(self, n):
        """
        The method returns top-n tags with most words inside. It is a dict
        where the keys are tags and the values are the number of words inside the tag.
        Sort it by numbers descendingly.
        """
        return big_tags

    def longest(self, n):
        """
        The method returns top-n longest tags in terms of the number of characters.
        It is a list of the tags. Sort it by numbers descendingly.
        """
        return big_tags

    def most_words_and_longest(self, n):
        """
        The method returns the intersection between top-n tags with most words inside and
        top-n longest tags in terms of the number of characters.
        It is a list of the tags.
        """
        return big_tags

    def most_popular(self, n):
        """
        The method returns the most popular tags.
        It is a dict where the keys are tags and the values are the counts.
        Sort it by counts descendingly.
        """
        return popular_tags

    def tags_with(self, word):
        """
        The method returns all the tags that include the word given as the argument.
        It is a list of the tags. Sort it by tag names alphabetically.
        """
        return tags_with_word
```

c. Class Movies

```
class Movies:
    """
    Analyzing data from movies.csv
    """
    def __init__(self, path_to_the_file):
        """
        Put here any fields that you think you will need.
        """
    def dist_by_release(self):
        """
        The method returns a dict where the keys are years and the values are counts.
        You need to extract years from the titles. Sort it by counts descendingly.
        """
        return release_years
    def dist_by_genres(self):
        """
        The method returns a dict where the keys are genres and the values are counts.
        Sort it by counts descendingly.
        """
        return genres

    def most_genres(self, n):
        """
        The method returns a dict with top-n movies where the keys are movie titles and
        the values are the number of genres of the movie. Sort it by numbers descendingly.
        """
        return movies
```

d. Class Links

```
class Links:
    """
    Analyzing data from links.csv
    """
    def __init__(self, path_to_the_file):
        """
        Put here any fields that you think you will need.
        """

    def get_imdb(list_of_fields):
        """
        The method returns a list of lists [movieId, field1, field2, field3, ...].
        For example, [movieId, Director, Budget, Cumulative Worldwide Gross, Runtime].
        The values should be parsed from the IMDB webpages of the movies.
        Sort it by movieId descendingly.
        """
        return imdb_info

    def top_directors(self, n):
        """
        The method returns a dict where the keys are directors and
        the values are numbers of movies created by them. Sort it by numbers descendingly.
        """
        return directors

    def most_expensive(self, n):
        """
        The method returns a dict with top-n movies where the keys are movie titles and
        the values are their budgets. Sort it by budgets descendingly.
        """
        return budgets

    def most_profitable(self, n):
        """
        The method returns a dict with top-n movies where the keys are movie titles and
        the values are the difference between cumulative worldwide gross and budget.
         Sort it by the difference descendingly.
        """
        return profits

    def longest(self, n):
        """
        The method returns a dict with top-n movies where the keys are movie titles and
        the values are their runtime.
         Sort it by runtime descendingly.
        """
        return runtimes

    def top_cost_per_minute(self, n):
        """
        The method returns a dict with top-n movies where the keys are movie titles and
        the values are the budgets divided by their runtime.
         The values should be rounded to 2 decimals. Sort it by the division descendingly.
        """
        return costs
```

e. Class Tests

Create tests using `PyTest` for each and every method of the classes above. They should check:

1. if the methods return the correct data types,
2. if the lists elements have the correct data types,
3. if the returned data sorted correctly.

Run the tests before going to the next stage of the rush.

Report

Using only the classes and methods from `movielens_analysis.py`, prepare your report. You should do it in Jupyter Notebook. It is a great tool especially if you are a data scientist. It gives you an opportunity to work with the code interactively by launching and relaunching different cells with different values. You do not have to rerun your whole code. Also, you can put in the cells not only code but texts too, which is a great feature for making reports. Install it to your environment.

In this part of the rush, we will give you more freedom. We are not going to define the structure of your report. The goal of the report is to tell us an interesting story about the MovieLens dataset. Find the good structure and the right sequence. The only constraints:

1. you must use each and every method from `movielens_analysis.py` except class `Tests`,
2. every cell in your notebook should contain magic command `%timeit`,

3. all other imports are prohibited as well as using built-in functions. If you need them, put them in your module in advance.

Chapter V

Bonus part

1. Add to the classes more methods that you may find useful and interesting for your report. Do not forget to test them too.

2. Improve the tests. Check the correctness of your calculations as well. Precalculate manually some results and metrics and check if the methods return the correct information if you give them the specific input.

Chapter VI

Turn-in and peer-evaluation

Turn in your work using your git repository, as usual. Only the work that's in your repository will be graded during the evaluation.

During the correction, you will be evaluated on your turn-in (no functions that do all the heavy lifting for you) as well as your ability to present, explain, and justify your choices.