CS2006: Python 2
Dataset Analysis With Pandas

180000269, 180008859, 180021978 Tutor: Stephen Linton

May 07, 2020

Overview

In this practical we are asked to use Python Data Analysis Library (Pandas) to analyse Wikipedia citation data of four languages: German, English, Chinese and Ukrainian.

The raw data of each language is presented in a TSV (tab-separated values) file.

Before starting detailed analysis of each language, verification and preprocessing of the raw data are very important first steps, which can be executed in cleanData.py.

After raw data has been read in correctly and cleaned, initial descriptive analysis is performed in analysis.py. Finally, additional analytics are performed to explore the data further.

Design

Pre-processing raw data is split into two distinct functions, checkData to check if the data is valid and refineData to refine the data further. Prior to analysis, refine data is saved to a new .tsv file.

Analytics take form in modular functions, using pandas dataframe as parameter, and results of analysis are outputted in a series of print statements.

Extension work has been designed similarly as modular functions, runs in series after initial analysis.

Base Specifications

The minimum requirements of this project were to implement the parts which were currently undefined to complete

We were specifically asked for the following features:

- Check consistency of initial raw data
- Refine data in case of inconsistencies before analysis
- Output total number of records
- Output range of dates represented in the data
- Output a table with number of records for each identifier type

Base Requirements

We fully completed all the requirements of the base practical.

Each subsection for the additional requirements includes a bulleted list of extra features that fit within that difficulty level.

NB: As some requirements build on others, the features listed as the difficulty goes up may overlap with those listed earlier to make sure that all the features are mentioned, especially if they were specific to a suggested requirement difficulty.

Easy, Medium, and Hard Requirements

Easy

- Output a table with percentage of records for each identifier type
- Calculate an average number of days since citation appeared on Wikipedia
- Output a table with the number of citations from arXiv by the year of their appearance.

Medium

- Find an average number of citations per page
- Find first ten pages citing the largest number of sources
- Output a table with the number of citations from Zenodo by the year of their appearance

Hard

- Find first ten most highly cited sources
- Output a table with the number and percentage of citations appearing on Wikipedia by the number of years since their appearance
- Analyse and optimise the performance of different steps of your analysis (see Evaluation section below)

Evaluation

Parsing timestamp column to date_time format rather than object format has helped to optimise further analysis, because it is arduous and memory-inefficent to work with date_time as strings.

While initially we parse timestamp column to datatime during the pre-processing stage, we realized that after some research there is a more efficient way to parse ISO 8601 format in the read_csv stage. In fact, one of the parameter of read_csv, parse_dates, specially mentions that it has a fast route for ISO 8601 formatted data.

Other important high performance Pandas function usage includes groupby and value_counts. On the one hand, groupby has been efficient in operating data relating one column to other columns. On the other hand, value_counts focuses on finding frequencies of values in a single column.

Moreover, Pandas .apply() function, which is a powerful replacement of iterative functions, has been used in Hard extensions.

If any illegal values in the data can not be removed or amended without compromising the rest of the data the checkData function prints an error message and exits the program.

The refineData function removes and illegal and duplicate values to ensure there are not inconsistencies with the analysis.

Overall, we are very happy with the progress and completeness of our solution to both the base and additional requirements. Our .apply() takes another function as its input and applies it along an axis of a DataFrame. In such cases where we are passing functions, a lambda is often convenient to package everything together in an optimised manner.

Conclusion

To begin with there was a lack of structure and people worked on the project individually and occasionally pushed to github. We didn't begin our work for several weeks, and once we did begin work, it was in disparate intervals, such that no one in our group was really able to work on the project at the same time due to timezones and individual scheduling of time.

We would have liked to be able to implement all the extensions, but we didn't have the time to make sure everyone would be able to do a similar amount of work while still completing all the extensions.

We would have liked to generate a better way of displaying the data, as textual output is only so good at showing results, especially when the results are so inclined to being displayed via graphical displays such as bar graphs.

Provenance

We did not source any code from anywhere and wrote all the code submitted ourselves. We did use examples and guides to figure out our code structure, but no code was not written by us.

Citations

values_counts function and parameter usage
- Used some parameters such as normalize and ascending to sort resultant dataframe
read_csv documentation of Pandas
- Used to parse datetime at input stage, optimise performance
use pandas to parse a column to datetime
- Used to analyse timestamp column efficiently
understand python datetime
- Used to extract year, date from timestamp column
Pandas high performance functions
- Referenced to analyse optimisation when using groupby and value_counts functions