CS2006 P2: Dataset analysis with Pandas
Team 21 Group Report

Overview

In this practical, we have been given datasets consisting of information on Wikipedia pages and their citations across four languages. Each record contains information that links the Wikipedia page and the citation source as well as information on when said citation was added to the Wikipedia page and other such information. We have been tasked with creating a program that must examine the datasets and remove any inconsistencies, building a 'refined' data set that can be further analysed. Inconsistencies in the original datasets include duplications, corrupted or unreadable files, unexpected variable values and discrepancies in logic. Once the refined dataset has been produced, the number of records, the range of data and a table showing the number of records for each identifier type within the data set should all be shown in the output. Code must also be modular to encourage reusability. This must all be done using Python code so that data analysis can be carried out automatically and can be replicated with ease. We must use Pandas to meet these requirements, rather than implementing our own procedures.

All basic requirements have been met and all additional requirements have been met as well, including the final hard requirement, which had no concrete idea to implement but has been discussed in this report.

Design and Implementation

Basic 1 – cleaning the data

The first part of this practical was performing quality control on the data in order to produce a clean copy that could then be used for further analysis. We began by getting rid of any rows that had missing values, and duplicate rows. Next, we needed to check that each column was storing the correct data:

page_id = int

page_title = utf-8

rev_id = int

timestamp = datetime

type = pmid/pmcid/doi/isbn/arxiv

id = utf-8

To check that each column was storing the correct data, I created a list of datatypes and then checked them against their expected datatype. Then, if the column was storing the wrong datatype, I did a regex check to get rid of any incorrect datatypes before converting it to the correct datatype. With the id column, I had to do a bit more work as there were many different formats the id could take, and this was dependent on the type column. I needed to find a way that filtered out any values that were the wrong structure for that specific type without making it too slow. Eventually I decided that the best way

to go about this was to split the Data Frame up into groups depending on the type, and then use regex to filter out any ids that did not fit the structure for that type. I found it quite difficult working out what the correct format was for each id, however these are what I came up with:

pmid = number between 1-8 digits

pmcid = 7 digit number

doi = a prefix/suffix where the prefix is 10.NNNN, where NNNN is a number greater than 1000, and then any number of further .numbers and the suffix is just any valid unicode

isbn = a 13 digit number starting with 978/979 or a 10 digit number, where the first of the 10 digits signifies the language, however I decided not to do a check for this as I felt it would cause problems as, for example, the dewiki dataset contains both isbns starting with 0/1 (for English) and 3 (for German)

arxiv = either subject followed by an optional .class (2 letters long) then /YYMMNumber or YYMM.Number

Once I had filtered the individual groups, I then joined them back together.

Having finished cleaning the data, I decided that it would be useful to print what percentage of rows had been removed, in this way we could garner whether we were getting rid of too much data and thus change our filtering techniques accordingly. Using this I realised that we were removing quite a lot of data, so I looked at where the majority was being removed using jupyter lab and I noticed that it was in the isbn filtering. Therefore, I checked if I could see anything by looking at the isbns in some of the smaller data sets and noticed that a lot of them ended in X. Research proved that this is a check digit, and so I decided to allow it. This helped to reduce the amount being filtered out to around 1.5% on average in the datasets.

The regular expressions used to match the ID types were refined throughout development and were finalised using an online regular expression matcher with real ID values to ensure that the regular expressions matched for valid IDs only.

To allow the now consistent data to be written to a new file, initially a second command line argument was given to specify this new file name. However, to better match the examples in the specification, it was decided that the clean data would be written to a file in the form ‘filename_consistent.tsv’ before the data would be analysed. Repeated analysis of the same data could therefore be sped up and bypass the consistency checking stage by determining whether the file ends in ‘consistent’.

Basic 2 – Initial descriptive analysis

The initial descriptive analysis proved simple to implement, with pandas methods being available for each of the four elements. The total record count is the length of the data frame index, and the maximum and minimum timestamp values are easily retrieved using 'max' and 'min' on the timestamp column. A new dataframe could be created with the length of the index for each specific ID type using the ‘loc’ method to retrieve each. This was then printed to fulfill the final data point.

Easy 1 - Output a table with the percentage of records for each identifier type

For this requirement, it was a simple case of taking the number of records for each id type and dividing that by the total number of records. Then simply multiplying the result will give the percentage. However, the slight issue here came with rounding the percentages as they were to umpteen decimal places to begin with. This only took a small amount of time to fix however as a simple round function could be applied to the percentages to get all results to two decimal places.

Easy 2 - Calculate an average number of days since citation appeared on Wikipedia, as on March 1, 2018

For this requirement, I had to get to grips with how datetimes worked with pandas. I created a new datetime for the 1st March 2018 00:00:00 and then I subtracted all the values in the timestamp column from it in order to get the duration. I decided to make it include March 1st by adding 1, since this was the case in Excel. I put all the results into a new column called “duration” and I printed the mean.

Easy 3 - Output a table with the number of citations from arXiv by the year of their appearance

The number of citations with the arXiv ID type was retrieved using the pandas ‘value counts’ method in conjunction with ‘loc’ to retrieve rows exclusively with the arXiv ID type. This created a dataframe with each row representing a year and the number, or ‘value count’, of arXiv IDs with timestamps from that year.

Medium 1 - Find an average number of citations per page

The number of citations for a given page could be retrieved by accessing the ‘value counts’ of the Page ID column, since the number of occurrences of a page ID indicates the number of sources cited by the page, with each row being a page/citation relationship. The average could then be easily calculated with the numpy ‘mean’ method, and was rounded to 2 decimal places for legibility.

Medium 2 - Find first ten pages citing the largest number of sources.

For this requirement, it was a simple case of printing out a table layout and ten records. Before the printing of the records, all duplicates were dropped, and the records were sorted in order of ‘number of sources’, with the most sources first. The ten records at the head of this list were then printed, meaning that the requirement was being met.

Medium 3 - Output a table with the number of citations from Zenodo (https://zenodo.org/) by the year of their appearance

As the hint for this requirement referred to “zenodo” in the doi, I decided to filter out all other rows of other types before checking for zenodo, I then created a new data frame containing only the zenodo citations. Once I had done this I simply had to get the value_counts() for each year in the timestamp column and print it.

Hard 1 - Output a table with the number and percentage of citations appearing on Wikipedia by the number of years since their appearance

For this requirement I had two choices, I could either calculate the time difference and then convert it to years or convert it to years first. I decided to go with the latter option. Once I had done this, I decided to create new data frame containing the years since appearance, and the value_counts(). However, when I ran this and compared it to the Excel results, I noticed there was a difference. Therefore, instead of

doing value_counts() I decided to create a column containing the count and year for every row and then remove the duplicates, this seemed to solve the problem. Once I had this data frame, it was simple to add a third column containing the percentages.

Hard 2 - Find first ten most highly cited sources

The ten most highly cited sources were retrieved using the pandas 'groupby' method to retrieve a group of the ID type and ID columns, since it is only a combination of these that identifies a unique source. This group could then be sorted into descending order of the number of appearances of each type/ID combination using the pandas 'sort_values' method. The first 10 elements of this group would then be the ten most cited sources, so these were printed using the 'head' method with a value of ten.

Testing

Whilst writing the program, I used pandas on jupyter lab to come up with ideas as I found that it was able to process data much faster. I also created a small sample file, called enshort.tsv, which was a randomly selected sample of the enwiki.tsv data file.

Once we had finished writing the program, Microsoft Excel was used to test our methods. This was very useful, as one member of the team has a strong knowledge of dealing with Excel formulas, so the methods written using python and pandas could be quickly replicated in Excel. This was extremely useful in highlighting small errors that would otherwise have gone unnoticed.

Input tests:

```
(base) [redacted]:code [redacted]$ ./analysis.py data/nonexistent.tsv
data/nonexistent.tsv not found
(base) [redacted]:code [redacted]$ ./analysis.py
Usage: analysis.py <file-to-read>
```

Examples

German – dewiki.tsv

```
Removed 0.31% percent of rows due to inconsistencies

--- Initial descriptive analysis ---
Total number of entries: 1643697
Earliest revision date: 2001-12-02 17:08:33
Latest revision date: 2018-03-02 01:26:31
Number of records by ID type:
     ID  Number of entries
   PMID              11925
  PMCID               1895
    DOI             109800
   ISBN            1516233
  ARXIV               3844
```

Chinese – zhwiki.tsv

```
Removed 1.73% percent of rows due to inconsistencies

--- Initial descriptive analysis ---
Total number of entries: 354671
Earliest revision date: 2003-01-03 07:11:49
Latest revision date: 2018-03-02 10:44:29
Number of records by ID type:
    ID  Number of entries
  PMID              26218
 PMCID               4091
   DOI              59228
  ISBN             261179
 ARXIV               3955
```

Ukrainian – ukwiki.tsv

```
Removed 1.58% percent of rows due to inconsistencies

--- Initial descriptive analysis ---
Total number of entries: 221508
Earliest revision date: 2004-04-14 11:49:23
Latest revision date: 2018-03-01 19:25:09
Number of records by ID type:
    ID  Number of entries
  PMID               9304
 PMCID               1389
   DOI              70883
  ISBN             138199
 ARXIV               1733
```

English – enwiki.tsv

```
Removed 1.99% percent of rows due to inconsistencies

--- Initial descriptive analysis ---
Total number of entries: 3719102
Earliest revision date: 2001-12-06 15:08:27
Latest revision date: 2018-03-02 06:32:39
Number of records by ID type:
    ID  Number of entries
  PMID             609848
 PMCID             135471
   DOI            1196386
  ISBN            1740235
 ARXIV              37162
```

Demo of the results of additional analysis using enwiki.tsv:

Output a table with the percentage of records for each identifier type

```
Percentage of records by ID type:
    ID  Percentage of entries
  PMID                 16.40
 PMCID                  3.64
   DOI                 32.17
  ISBN                 46.79
 ARXIV                  1.00
```

Calculate an average number of days since citation appeared on Wikipedia, as on March 1, 2018

```
Average number of days since citation appeared on Wikipedia, as on March 1 2018:
1773.56
```

Output a table with the number of citations from arXiv by the year of their appearance

```
Number of citations from arXiv by year:
2017    7485
2016    6100
2012    5914
2011    4931
2015    3269
2013    2456
2014    1637
2018    1067
2008    1059
2009    1044
2010     941
2007     489
2006     452
2005     224
2004      79
2003      10
2002       5
```

Find an average number of citations per page

```
Average number of citations per page:
4.35
Note - this does not include pages with zero citations (not present in data)
```

Find first ten pages citing the largest number of sources.

First ten pages citing the largest number of sources:

Page id	Page title	Number of sources cited
52206193	2017 in paleontology	895
47597082	2016 in paleontology	752
36315057	Induced stem cells	720
43792556	2015 in paleontology	627
9236	Evolution	607
9238638	Haplogroup T-M184	568
18914017	Alzheimer's disease	558
156964	MicroRNA	512
11664498	List of sequenced bacterial genomes	511
4501641	Stress granule	510

Output a table with the number of citations from Zenodo (https://zenodo.org/) by the year of their appearance

Number of citations from Zenodo (https://zenodo.org/) by the year of their appearance

2016	217
2017	129
2015	86
2018	22
2013	1

Output a table with the number and percentage of citations appearing on Wikipedia by the number of years since their appearance

Number and percentage of citations appearing on Wikipedia by the number of years since their appearance:

Years since	Number of citations	Percentage of citations
2	98109	2.64
3	483496	13.00
4	434029	11.67
5	427598	11.50
6	366572	9.86
7	303141	8.15
8	267173	7.18
9	266994	7.18
10	282301	7.59
11	261834	7.04
12	259446	6.98
13	217693	5.85
14	36735	0.99
15	10904	0.29
16	2380	0.06
17	561	0.02
18	135	0.00
19	1	0.00

Find first ten most highly cited sources

```
10 most cited sources:
type   id
isbn   9781921496325                  4769
pmid   12477932                       4702
doi    10.1073/pnas.242603899         4702
isbn   1904994105                     4667
doi    10.1101/gr.2596504             3387
pmid   15489334                       3387
doi    10.1051/0004-6361:20078357     2895
arxiv  0708.1752                      2891
isbn   0646119176                     2736
pmid   14702039                       2212
```

Analysis of Performance

Hard 3 - Analyse and optimise the performance of different steps of your analysis; summarise your experience in the form of recommendations included in the report.

The code for this solution is split into three files – a core 'analysis.py' main file that runs the components of the solution, as well as 'consistency.py', containing all methods to ensure consistency in data, and 'analysisMethods.py', containing all the methods for initial and additional analysis, as well as methods combining these to streamline running all analysis methods at once. This improves the modularity of the code, makes it easier to maintain, and aids reusabilty, allowing components to be reused on other projects more easily in the future.

The complete analysis unsurprisingly runs more slowly on larger files containing more data, with most time spent reading in each line of data. While this unfortunately cannot be optimised further, the naming convention used when generating a new file containing only consistent data means that there will be no need to re-check consistency if the user is reusing an already consistent data file, improving performance. Avoiding iteration in the analysis methods is key to optimising performance, and this has been avoided in all but one case – the second hard requirement, which uses the 'group by' functionality in pandas, which internally uses iteration to zip two columns together. While a faster version of this analysis function was investigated, it was decided to use this method despite its slower performance due to the emphasis on using the functions supplied with pandas where possible, since this was the ideal application of the 'group by' method.

In 'consistency.py', each inconsistency is checked for and dealt with separately inside of one function, then the file's other function is called which adds the refined record to the refined record structure. This again aids with the code's modularity which is essential for the code to be **reusable**, thus we highly recommend this as that is an essential goal of this practical.

Running this Solution

The complete solution is run by running the file 'analysis.py', which requires a path tsv data file as a command line argument. The file can therefore be run as follows:

python3 analysis.py <path-to-file>

The solution can also be run as an executable by first running 'chmod +x analysis.py' before running the solution as follows:

./analysis.py <path-to-file>

The output of the program can be sent to a file as follows:

./analysis.py <data-file> > <output-file>

Evaluation and Summary

Overall, our program demonstrates: **correctness** since its results are equivalent to those found when using the data with excel, **repeatability** since the data only needs to be cleaned once and then you can run the process on the cleaned data to skip the cleaning, **replicability** since we are all getting the same results on our different individual machines, **reproducibility** since it worked when we created a new data set that was much smaller but had the same set of metrics, **reusability** since a number of the components can be easily modified to test different things, for example, the arXiv by year function only needs a small change for it to become pmid by year etc.

This project has taught us a lot about python, and its usefulness as a language; it has also strengthened our knowledge on regexes. If we had more time, we would've liked to have maybe done a little more checking on the file, for example, asking what would happen if we tried to read a text file into the program; however, overall we are happy with the results and pleased that we managed to implement all the requirements.