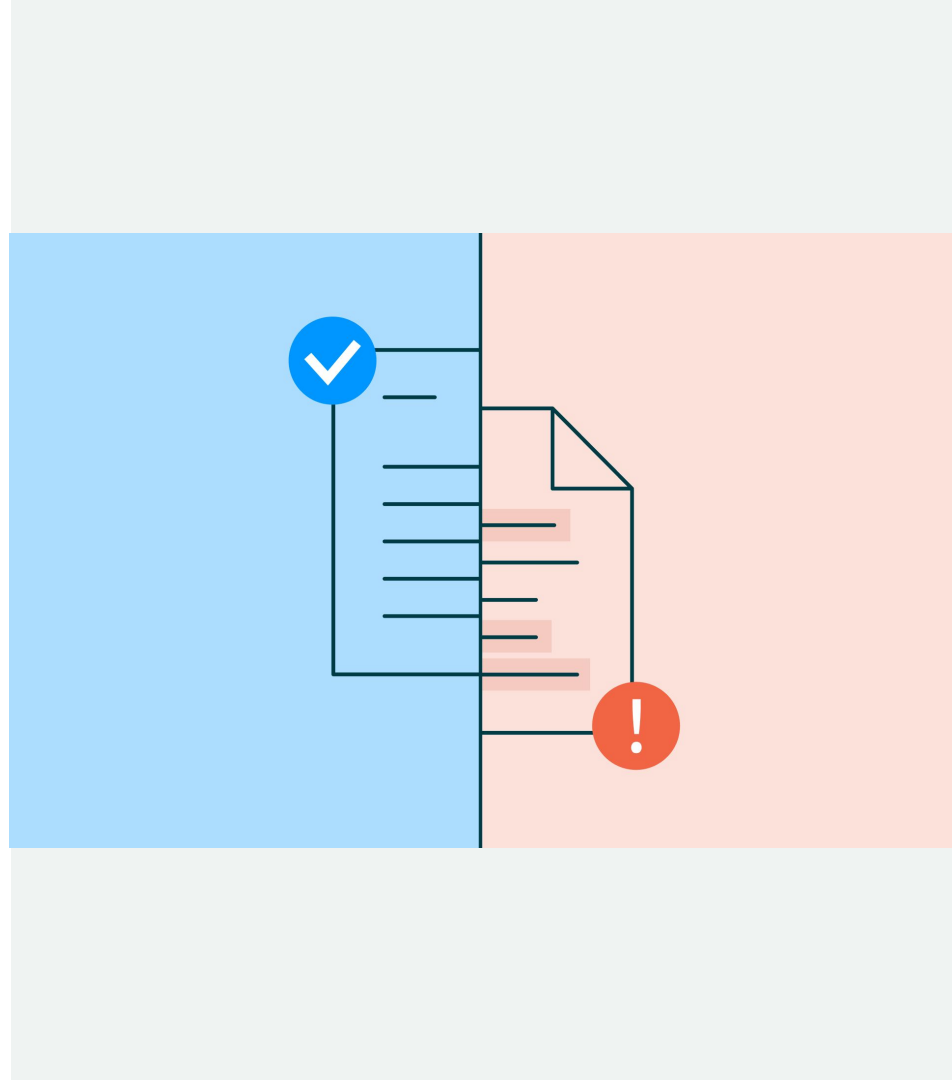


Plagiarism Detector

APAN 5210 Python for Data Analysis Final Project

Team 6

Herbert Gonzalez, Kyle Ongko, Michael Popper,
Nickolas Amarta Tan & Nicole Francolini

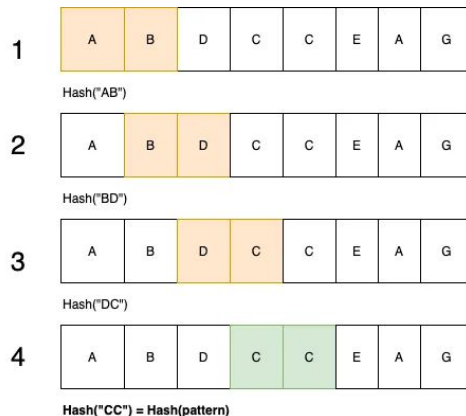


Algorithm #1

- Rabin-Karp algorithm of comparing strings
- Makes use of hash tables using the rolling hash technique
- The hash function maps data of an arbitrary size to a fixed value, while the rolling hash allows the algorithm to calculate a hash value quickly
- The algorithm allows for the size of the n-gram you want to compare.
- The similarity score is calculated as:

$$P = \frac{2 * SH}{THA + THB} * 100\%$$

Rabin Karp Algorithm



Top 5 Results Highest Similarity %

File1	File2	Similarity
0101.ipynb	0102.ipynb	97.090652
0102.ipynb	0103.ipynb	96.991921
0101.ipynb	0103.ipynb	96.991921
0109.ipynb	0123.ipynb	58.408668
0107.ipynb	0130.ipynb	49.699812

Algorithm #2

- Utilizes the **SequenceMatcher** from the difflib library to compute the similarity ratio between two cells:
 - SequenceMatcher **compares pairs of sequences**, usually strings. It has the ability to measure the similarity between two sequences of strings.
- Extracts code source from each cell in a DataFrame and calculates the similarity scores between different code cells within each notebook, then storing these scores in a matrix:
 - The **.ratio() method** will compute the **similarity ratio between two sequences** selected, returning a value between 0 (no similarity) and 1 (identical similarity).

```
def similarity_calc(cell1, cell2):  
    return SequenceMatcher(None, cell1, cell2).ratio()
```

.ratio() Calculation:

Calculates the ratio as: $\text{ratio} = 2.0 * M / T$,
where M = matches, T = total number of
elements in both sequences

Final Output:

	File1	File2	Similarity2
0	0101.ipynb	0102.ipynb	1.00
1	0101.ipynb	0103.ipynb	1.00
2	0101.ipynb	0104.ipynb	1.00
3	0101.ipynb	0105.ipynb	0.89
4	0101.ipynb	0106.ipynb	1.00
..	...	...	...
430	0127.ipynb	0129.ipynb	1.00
431	0127.ipynb	0130.ipynb	1.00
432	0128.ipynb	0129.ipynb	1.00
433	0128.ipynb	0130.ipynb	1.00
434	0129.ipynb	0130.ipynb	1.00

[435 rows x 3 columns]

Algorithm #3 - Line Comparison using SequenceMatcher

- difflib.SequenceMatcher
- Comparing lines instead of characters
- Broke down the notebooks into lines, ignoring leading and trailing whitespaces for a more granular comparison
- Calculated Similarity Score between each pair of notebooks based on the ratio of matching lines

Summary of Output

	File1	File2	Similarity
0	0101.ipynb	0102.ipynb	100.00
1	0101.ipynb	0103.ipynb	99.69
2	0101.ipynb	0104.ipynb	67.04
3	0101.ipynb	0105.ipynb	50.51
4	0101.ipynb	0106.ipynb	63.25
..	...	...	...
430	0127.ipynb	0129.ipynb	59.34
431	0127.ipynb	0130.ipynb	27.16
432	0128.ipynb	0129.ipynb	60.48
433	0128.ipynb	0130.ipynb	40.29
434	0129.ipynb	0130.ipynb	39.21

[435 rows x 3 columns]

Top 5 Results Highest Similarity %

	File1	File2	Similarity
0	0101.ipynb	0102.ipynb	100.00
29	0102.ipynb	0103.ipynb	99.69
1	0101.ipynb	0103.ipynb	99.69
217	0109.ipynb	0123.ipynb	90.13
256	0111.ipynb	0123.ipynb	82.19

Top 5 Results Lowest Similarity %

	File1	File2	Similarity
356	0117.ipynb	0130.ipynb	9.30
317	0115.ipynb	0118.ipynb	9.58
191	0108.ipynb	0118.ipynb	16.78
244	0110.ipynb	0130.ipynb	19.01
346	0117.ipynb	0120.ipynb	19.91

Algorithm #4 - rapidfuzz

Algorithm:

- **Levenshtein (edit) distance:** measures the similarity between two strings or sequences, **minimum edit distance** at bottom right corner
- **Creates matrix:** doc1 x doc2 lengths of the two sequences
- **Insertion, deletion, substitution:** fill cells with values of the edit distance of the substrings of the strings
- **Similarity score:** [0,1] score range, based on the formula $1/(1+L.distance)$
- Increasing Levenshtein distance will indicate a lower similarity score between the two sequences

Approach:

- **fuzz.ratio:** function takes the two notebooks and calculates the similarity score
- **for loop:** to make the function calculate all possible permutations of two documents

	File1	File2	Similarity
0	0101.ipynb	0102.ipynb	100.000000
29	0102.ipynb	0103.ipynb	98.987952
1	0101.ipynb	0103.ipynb	98.987952
217	0109.ipynb	0123.ipynb	95.132177
256	0111.ipynb	0123.ipynb	94.274783

Top 5 - Highest Similarity Score

	File1	File2	Similarity
227	0110.ipynb	0113.ipynb	42.537135
428	0126.ipynb	0130.ipynb	44.767656
186	0108.ipynb	0113.ipynb	45.409833
130	0105.ipynb	0126.ipynb	45.528772
114	0105.ipynb	0110.ipynb	46.355776

Bottom 5 - Lowest Similarity Score

Algorithm #5 - itertools - combinations

Explanation:

- **`load\_notebook` Function:** Reads a Jupyter Notebook file and loads its content as JSON.
- **`extract\_cells` Function:** Extracts and concatenates the content of either code or markdown cells from a notebook.
- **`simple\_similarity` Function:** Computes a basic similarity score between two strings.
- **`compare\_notebooks` Function:** Compares two notebooks based on the similarity of their code and markdown cells.
- **File Comparison:** The script iterates over all combinations of notebook files and compares them, storing the results in a list.
- **Results Visualization:** The results are converted into a pandas DataFrame for easy viewing.

Notes:

- Other algorithms may be more effective to detect plagiarism, this is just an example of using itertools combinations, and the results show a little discrepancy compared to previous algorithms.

	File1	File2	Similarity5
0	0101.ipynb	0102.ipynb	1.000000
1	0101.ipynb	0103.ipynb	0.316135
2	0101.ipynb	0104.ipynb	0.194430
3	0101.ipynb	0105.ipynb	0.198851
4	0101.ipynb	0106.ipynb	0.165007
..	...	...	...
430	0127.ipynb	0129.ipynb	0.529522
431	0127.ipynb	0130.ipynb	0.199376
432	0128.ipynb	0129.ipynb	0.185261
433	0128.ipynb	0130.ipynb	0.191414
434	0129.ipynb	0130.ipynb	0.198217

Meta-heuristic algorithm

- The meta-heuristic algorithm combines all 5 of the algorithms created by our team
- Since each algorithm outputs a similarity score out of 1 or 100, we used an average (rescaled to 100) for a basic meta-heuristic algorithm

Top 5 Results Highest Similarity %

File1	File2	MetaHeuristic_Score
0101.ipynb	0102.ipynb	99.418130
0101.ipynb	0103.ipynb	85.456668
0102.ipynb	0103.ipynb	85.456668
0109.ipynb	0123.ipynb	72.868296
0111.ipynb	0114.ipynb	71.983745

Bibliography

- <https://www.youtube.com/watch?v=IRzC3w2NDg0>
- <https://www.makeuseof.com/python-plagiarism-detector-how-to-build/>
- <https://docs.python.org/3/library/difflib.html>
- <https://github.com/maxbachmann/RapidFuzz/>
- <https://www.geeksforgeeks.org/simple-plagiarism-detector-in-python/amp/>
- <https://stackoverflow.com/questions/74301610/comparing-pandas-dataframe-column-with-list>
- <https://medium.com/@lostandfound2654/extracting-only-cell-codes-from-jupyter-notebooks-a-hand-python-trick-6ff2c948413e>
- <https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc>
- <https://stackoverflow.com/questions/72378391/how-to-get-maximum-similarity-value-between-lists-with-numpy>
- <https://stackoverflow.com/questions/61864248/comparing-each-element-in-series-to-every-other-is-there-a-better-way-than-numpy>
- <https://stackoverflow.com/questions/44905155/nested-loops-avoiding-self-and-reciprocal-comparisons>
- <https://python.plainenglish.io/a-simple-plagiarism-rate-checker-using-rabin-karp-string-matching-algorithm-in-python-e823d29d3f21>

Thank you!

