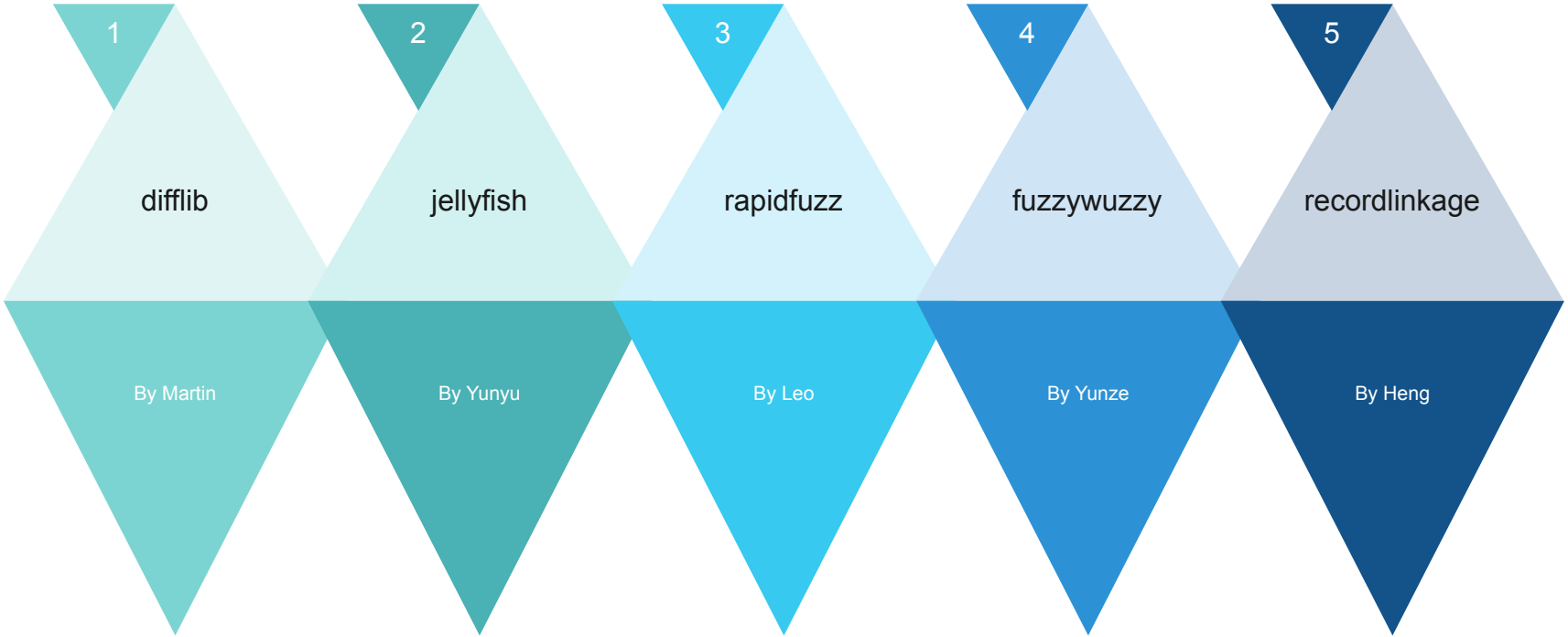
5210 Presentation Assignment

Group 9



By Martin Ng

- A Python library offering tools to compare sequences.
- It contains functions to find differences and calculate similarity
- Using algorithms like SequenceMatcher
- Yield a result of 23k matched pairs in the given dataset. The tested minimum threshold is 0.6
- More conservative match approach: longer contiguous matching sequences
  => fewer matches if the matches are not long or contiguous

```python
def fuzzy_match_with_difflib(left_df, right_df, threshold=0.80):
    results = []
    common_blocks = set(left_df['block_key']).intersection(set(right_df['block_key']))

    for block in common_blocks:
        left_block = left_df[left_df['block_key'] == block]
        right_block = right_df[right_df['block_key'] == block]
        for _, left_row in left_block.iterrows():
            best_match = None
            best_score = threshold
            for _, right_row in right_block.iterrows():
                score = SequenceMatcher(None, left_row['combined'], right_row['combined']).ratio()
                if score > best_score:
                    best_score = score
                    best_match = right_row['business_id']

            if best_match:
                match_data = {
                    'left_id': left_row['entity_id'],
                    'right_id': best_match,
                    'match_score': best_score
                }
                results.append(match_data)

    return pd.DataFrame(results)
```

In [7]:

```python
matched_results = fuzzy_match_with_difflib(left_df, right_df,0.6)
print(matched_results)
```

```
       left_id  right_id  match_score
0         1941     77575     0.843750
1        44647     77575     0.620690
2        69578     65157     0.842105
3        85017     65157     0.629213
4         1551     51504     0.948718
...        ...       ...          ...
22951    13437      8755     0.633663
22952    38931     18080     0.666667
22953    51923      8755     0.622222
22954    93260      8755     0.972477
22955    94192     22264     0.754098

[22956 rows x 3 columns]
```

Jellyfish

Calculate string similarity and distance.

By Yunyu Huo

01 Preparation

- Convert relevant columns to the same data type (e.g., string).
- Fill missing values with empty strings.
- Create a common field for comparison (e.g., 5 digits zip code).

02 Function Setting

- Jaro_winkler_similarity: calculate the Jaro-Winkler similarity between two strings.

```
def calculate_similarity(row):
    left_name = row['name_x']
    left_address = row['address_x']
    right_name = row['name_y']
    right_address = row['address_y']

    name_similarity = jellyfish.jaro_winkler_similarity(left_name.lower(), right_name.lower())
    address_similarity = jellyfish.jaro_winkler_similarity(left_address.lower(), right_address.lower())

    return (name_similarity + address_similarity) / 2
```

03 Calculation

04 Output

	entity_id	business_id	similarity_score
424	7	36752	0.823996
2606	2887	40775	0.976190
2863	2887	49464	0.817110
3462	3241	41513	0.857944
3693	3241	49438	0.975000
...	...	...	...
34706430	39137	48887	0.962963
34706661	45570	51316	0.975238
34706730	49626	48119	0.921447
34706746	49626	49649	0.988889
34707154	79058	49649	0.843804

[25659 rows x 3 columns]

Rapid Fuzz

By Leo Zhou

- A MIT licensed C++ written package with a lot of algorithmic improvements on top of Fuzzywuzzy.
- It contains many string_metrics like hamming or jaro_winkler that is not contained in Fuzzywuzzy.
- Using hybrid index blocking for further efficiency enhancement.
- Yield a result of 48k matched pairs in the given dataset. The tested minimum threshold is 0.65.

```python
def fuzzy_match_with_rapidfuzz(left_df, right_df, threshold=85):
    results = []
    # Identify common blocks to minimize comparisons
    common_blocks = set(left_df['block_key']).intersection(set(right_df['block_key']))

    # Perform matching within each common block
    for block in common_blocks:
        left_block = left_df[left_df['block_key'] == block]
        right_block = right_df[right_df['block_key'] == block]
        for _, left_row in left_block.iterrows():
            # Using RapidFuzz to find the best match in the right block
            best_match = process.extractOne(
                left_row['combined'],
                {idx: row['combined'] for idx, row in right_block.iterrows()},
                scorer=fuzz.WRatio,
                score_cutoff=threshold
            )
            if best_match:
                # Accessing details of the best match
                match_data = {
                    'left_dataset': left_row['entity_id'],
                    'right_dataset': right_block.loc[best_match[2]]['business_id'],
                    'confidence_score': best_match[1]
                }
                results.append(match_data)

    return pd.DataFrame(results)
```

In [9]:
```python
from rapidfuzz import process, fuzz

# Execute fuzzy matching
matched_results = fuzzy_match_with_rapidfuzz(left_df, right_df)
```

In [10]:
```python
print(matched_results)
```

```
       left_dataset  right_dataset  confidence_score
0              8703          20197         85.500000
1             19631          20197         85.500000
2             73038          20197         85.500000
3             85862          20197         85.500000
4               903          63344         85.500000
...             ...            ...               ...
47616          1115          23915         95.000000
47617         17529           4609         85.500000
47618         59493          22503         95.238095
47619         92447          12152         96.721311
47620         61857          38198         85.500000

[47621 rows x 3 columns]
```

Fuzzywuzzy

By Yunze Dou

Results

	source_entity_id	target_entity_id	confidence_score
0	78610	39955	86
1	93224	45062	86
2	15947	51530	86
3	24194	55870	92
4	25391	52741	86
...	...	...	...
47666	70735	51776	86
47667	74843	51776	86
47668	88163	51776	86
47669	90698	51776	86
47670	93905	72958	86

[47671 rows x 3 columns]

Imports and Exploration

Data Cleaning

Fuzzy Matching

- loads two datasets, left_df and right_df
- uses the head() method to preview the first few rows of each dataset, allowing the user to get a sense of the data structure and contents.
- Renaming zip_code to postal_code in right_df to match left_df
- Dropping Columns: categories, city, state, size are removed from both dataframes
- converts text to lowercase, removes leading and trailing

```
def perform_fuzzy_matching(source_df, target_df, threshold=85):
    """
    Perform fuzzy matching between two dataframes using specified blocking keys to reduce the comparison space.

    Args:
        source_df (DataFrame): Source dataframe.
        target_df (DataFrame): Target dataframe.
        threshold (int): The similarity score threshold to determine a valid match.

    Returns:
        DataFrame: A dataframe containing the matching results with entity IDs and scores.
    """
    match_results = []
    common_blocks = set(source_df['block_key']).intersection(target_df['block_key'])

    for block in common_blocks:
        source_block_df = source_df[source_df['block_key'] == block]
        target_block_df = target_df[target_df['block_key'] == block]

        for _, source_row in source_block_df.iterrows():
            best_match_idx, best_match_score = find_best_match(
                query=source_row['combined'],
                candidate_choices={idx: row['combined'] for idx, row in target_block_df.iterrows()},
                minimum_score=threshold
            )

            if best_match_score:
                match_data = {
                    'source_entity_id': source_row['entity_id'],
                    'target_entity_id': target_block_df.loc[best_match_idx]['business_id'],
                    'confidence_score': best_match_score
                }
                match_results.append(match_data)

    return pd.DataFrame(match_results)
```

recordlinkage

By Heng Zhu

```
def preprocess_data(df):
    for column in df.columns:
        if df[column].dtype == 'object':
            df[column] = df[column].str.replace(r'[^a-zA-Z0-9]', '', regex=True).str.lower()
    return df
```

```
left['address'] = left['address'].str.cat(left[['city', 'state', 'zip_code']], sep=' ')
right['address'] = right['address'].str.cat(right[['city', 'state', 'zip_code']], sep=' ')
```

```
left_common['block_key'] = left_common['name'].str[:5]
right_common['block_key'] = right_common['name'].str[:5]
```

```
comparer = recordlinkage.Compare()
comparer.string('name', 'name', method='jarowinkler', label='name_similarity')
comparer.string('address', 'address', method='jarowinkler', label='address_similarity')
```

```
matched.rename(columns={'entity_id':'left_dataset','business_id':'right_dataset','confidence_score':'confidence_score'})
```

		left_dataset	right_dataset	confidence_score
	81911	7	81911	0.812385
7	**82926**	7	82926	0.895760
	84020	7	84020	0.893032
3260	**82926**	3260	82926	0.809016
	83734	3260	83734	0.808502
...	...	...	...	...
94033	**79875**	94033	79875	0.994118
94034	**72786**	94034	72786	0.967469
94036	**53965**	94036	53965	0.953301
94167	**69598**	94167	69598	0.960116
94538	**79362**	94538	79362	0.946438

33298 rows × 3 columns

Thank you for listening