Bring Order to Chaos: Fuzzy Matching for Business Data

Team 22: Ronglu Jiang, Yongze Jiang, Xinyi Wen, Wanghao Ying, Yifan Zhang

01
Exploratory Data Analysis

& Visualization

Explore the Data

01

df.shape

Left data have 98509 rows and right data have 94585 rows.

02

df.dtypes

Left data have zip code as object and right data have postal code as float64.

03

df.head()

Exam both data frame.

04

State Distribution

With pie chart visualization, they have similar distribution.

05

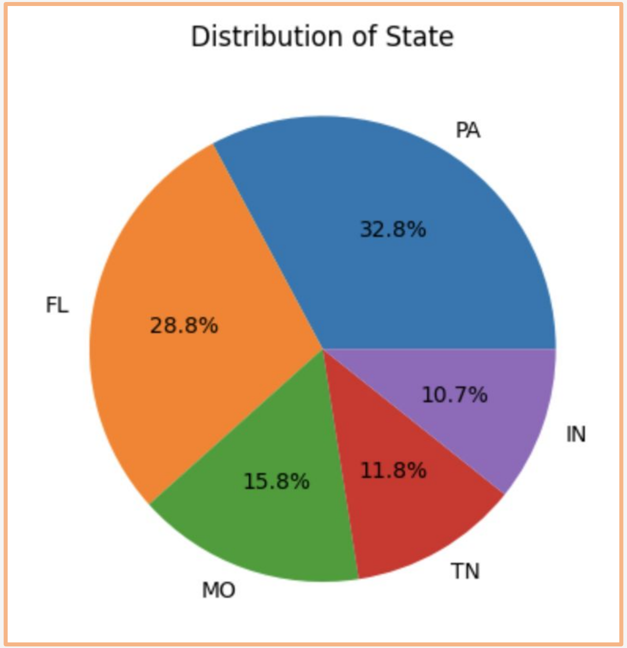
City Distribution

With pie chart visualization, they have similar distribution.

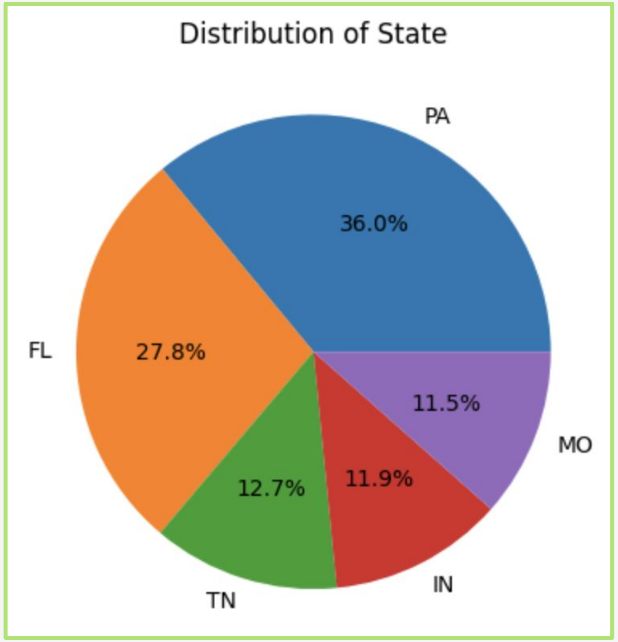
06

Zip Code Distribution by States

State Distribution

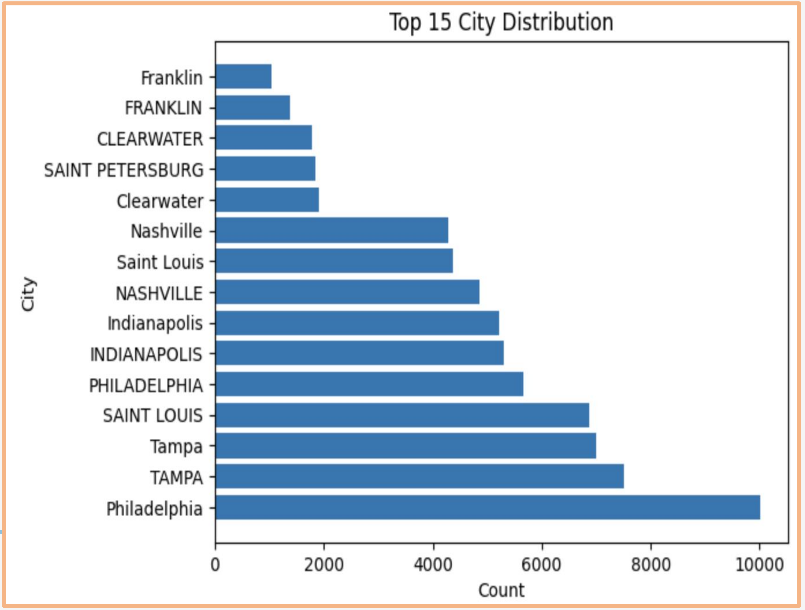


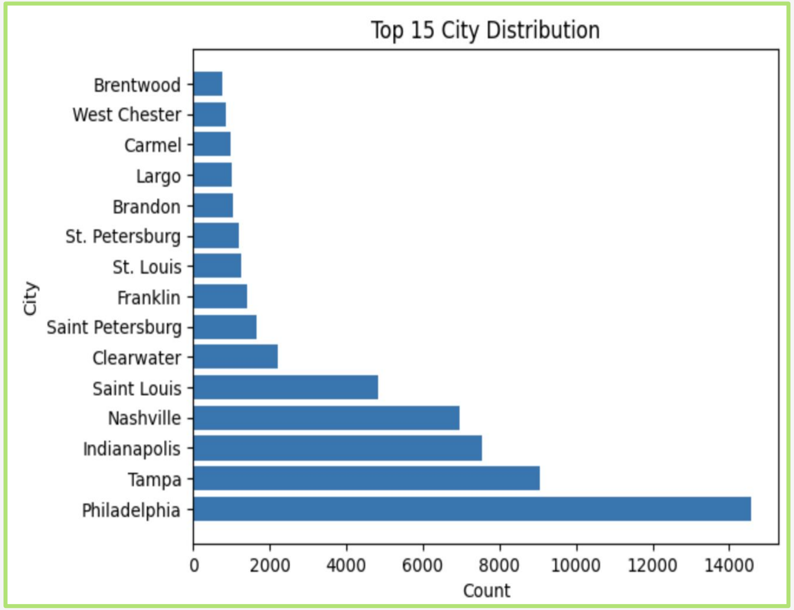
Left



Right

Top 15 City Distribution

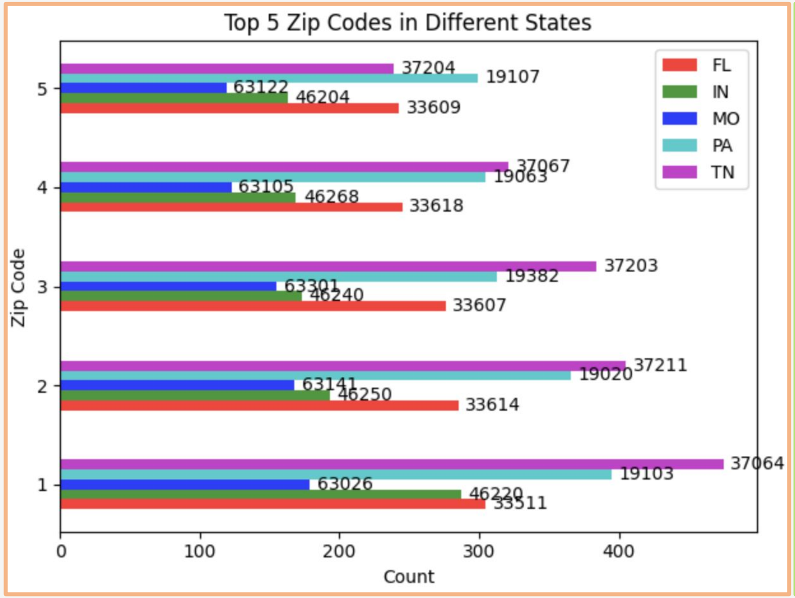




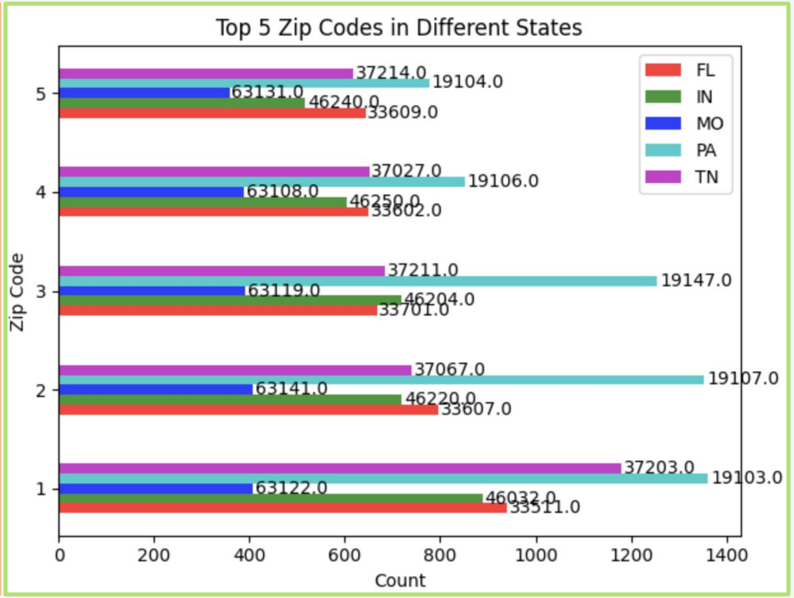
Left

Right

Top 15 City Distribution



Left



Right

02

Data Preprocessing

Data Cleaning & Standardization

Data manipulation processes

1: data standardization

2: abbreviations to full name

3: deal with city data entry inconsistencies

Data Standardization

For this step, we do the all characters lowercase process,
removing punctuations and removing trailing white spaces
[we did this process for name, address and city columns]

```
# function: lowercase, remove punctuation, trailing strip
def standarlize(df, column):
    # lower case
    df[column] = df[column].str.lower()
    # remove punctuation
    df[column] = df[column].str.replace(r'[^\w\s]+', '')
    # remove trailing whitespace
    df[column] = df[column].str.strip()
```

Replace abbreviations in address to full names

For this step, we define a function that with a dictionary of abbreviations, to convert the all the abbreviated words into full expressions

```
def abb_to_full(df, column):
    # create a dictionary of abbreviations and corresponding full name
    abbreviations = {
        r'\bave\b': 'avenue',
        r'\bblvd\b': 'boulevard',
        r'\bcir\b': 'circle',
        r'\bct\b': 'court',
        r'\bexpy\b': 'expressway',
        r'\bfwy\b': 'freeway',
        r'\bln\b': 'lane',
        r'\bpky\b': 'parkway',
        r'\brd\b': 'road',
        r'\bsq\b': 'square',
        r'\bst\b': 'street',
        r'\bste\b': 'suite',
        r'\btpke\b': 'turnpike',
        r'\bn\b': 'north',
        r'\be\b': 'east',
        r'\bs\b': 'south',
        r'\bw\b': 'west',
        r'\bne\b': 'northeast',
        r'\bse\b': 'southeast',
        r'\bsw\b': 'southwest',
        r'\bnw\b': 'northwest'
    }
    df[column] = df[column].replace(abbreviations, regex=True)
```

Data Entry Inconsistencies

Firstly, we determine the unique name of the city in dataset, and we found out that 'st louis', 'st petersburg' need to be changed;

After our process, we replace for consistency of city columns for saint louis and saint petersburg with minimum ratio of 80;

<u>But there are still too much non standardized city data, so we only replace st to saint</u>

```
left_city = left['city'].unique()
left_city.sort()
left_city
```

```
array(['ardmore', 'ballwin', 'bensalem', 'brandon', 'bryn mawr',
       'clearwater', 'collegeville', 'conshohocken', 'doylestown',
       'exton', 'fenton', 'florissant', 'franklin', 'havertown',
       'indianapolis', 'king of prussia', 'langhorne', 'lansdale',
       'largo', 'levittown', 'lutz', 'malvern', 'media', 'nashville',
       'new port richey', 'newtown', 'norristown', 'north wales',
       'palm harbor', 'philadelphia', 'phoenixville', 'pottstown',
       'saint charles', 'saint louis', 'saint petersburg', 'springfield',
       'st louis', 'st petersburg', 'tampa', 'warminster', 'wayne',
       'west chester', 'willow grove'], dtype=object)
```

'saint louis', 'saint petersburg','st louis', 'st petersburg' need to be changed

There are too many non-standardized city entry with unknown reason, thus we only replace 'st' to 'saint'

```
# check cities contain 'st'
right[right['city'].str.contains(r'\bst\b') == True]['city'].unique()
```

```
array(['st petersburg', 'st louis', 'st pete beach', 'st charles',
       'st davids', 'st louis downtown', 'st ann', 'st pete', 'st  louis',
       'st petersberg', 'st loius', 'st peters', 'st louis county',
       'st petersurg', 'st leo', 'st  charles'], dtype=object)
```

```
# replace 'st' with 'saint'
right['city'] = right['city'].replace( r'\bst\b', 'saint', regex=True)
```

Replace for consistency

```
# use the function we just wrote to replace close matches to "st. louis" with "st. louis"
replace_matches_in_column(df = left, column='city', string_to_match="saint louis", min_ratio = 80)
# use the function we just wrote to replace close matches to "st.petersburg" with "st.petersburg"
replace_matches_in_column(df = left, column='city', string_to_match="saint petersburg", min_ratio = 80)
```

```
# Examine the uniqueness of the cities
left_city = left['city'].unique()
left_city.sort()
left_city
```

```
array(['ardmore', 'ballwin', 'bensalem', 'brandon', 'bryn mawr',
       'clearwater', 'collegeville', 'conshohocken', 'doylestown',
       'exton', 'fenton', 'florissant', 'franklin', 'havertown',
       'indianapolis', 'king of prussia', 'langhorne', 'lansdale',
       'largo', 'levittown', 'lutz', 'malvern', 'media', 'nashville',
       'new port richey', 'newtown', 'norristown', 'north wales',
       'palm harbor', 'philadelphia', 'phoenixville', 'pottstown',
       'saint charles', 'saint louis', 'saint petersburg', 'springfield',
       'tampa', 'warminster', 'wayne', 'west chester', 'willow grove'],
      dtype=object)
```

03

Fuzzy Matching

```python
def find_matches1(left, right,threshold=0.80):
    results = []

    for index1, row1 in left.iterrows():
        for index2, row2 in right.iterrows():

            # Calculate name and address similarity
            name_similarity = fuzz.token_set_ratio(row1["name"], row2["name"])
            address_similarity = fuzz.token_set_ratio(row1["address"], row2["address"])
            city_similarity = fuzz.token_set_ratio(row1["city"], row2["city"])

            # Calculate confidence score
            confidence_score = (name_similarity * 0.4 + address_similarity * 0.4 + city_similarity * 0.2)  / 100

            if confidence_score > threshold:
                results.append((row1["business_id"], row2["entity_id"], confidence_score))

    matches = pd.DataFrame(results, columns=["left", "right", "confidence_score"])
    return matches
```

The ”find_matches1” function helps identify potential matches between records from two dataframes using fuzzy string matching. It considers name, address and city attributes to compute similarity scores and determine if records match based on a confidence score.

Function Running Logic

- Iterating through rows: The function compares each record from the left dataframe with every record in the right dataframe to identify potential matches.

- Calculating similarity scores: FuzzyWuzzy's ”token_set_ratio” method is used to calculate similarity scores for name, address and city fields.

- Computing confidence scores: A weighted average of similarity scores produces a confidence score for each record pair. If the confidence score is above a specified threshold, the pair is considered a match and added to the results list.

Confidence Score Calculation and Weights

```
# Calculate confidence score
confidence_score = (name_similarity * 0.4 + address_similarity * 0.4 + city_similarity * 0.2)  / 100

if confidence_score > threshold:
    results.append((row1["business_id"], row2["entity_id"], confidence_score))
```

- Name and address similarities play a significant role in accurately identifying matches, which is why they are assigned higher weights in the confidence score calculation.

- City, on the other hand, act as supportive criteria in the matching process. It cover broader ranges and are given lower weights to ensure that the matching algorithm effectively screens and identifies the most accurate matches.

Match by State & Zip Code

- Divide left and right datasets by state
- Within a state, get common zip codes of the left and right dataset and number of counts

```
In [132]: def zip_match(left, right):
              left_zip_counts = left['zip_code'].value_counts().reset_index()
              left_zip_counts.columns = ['zip_code', 'count']
              right_zip_counts = right['postal_code'].value_counts().reset_index()
              right_zip_counts.columns = ['zip_code', 'count']
              zip_merged_inner = pd.merge(left_zip_counts, right_zip_counts,
                                          on=['zip_code'],
                                          how='inner')
              return zip_merged_inner
```

```
In [138]: zip_match_TN = zip_match(left_TN, right_TN)
          zip_match_TN.head()
```

Out[138]:

	zip_code	count_x	count_y
0	37203	1339	1145
1	37064	1217	580
2	37211	1078	650
3	37067	937	730
4	37204	631	403

Match by State & Zip Code

- Conduct fuzzy matching between each pair of left & right subsets with the same zip code to reduce execution time
- Concatenate matching results to one data frame and save as a csv file

```
In [197]: def find_match_byzip(zip_match_data, left_data, right_data, filename):
              # create an empty dataframe
              all_results = pd.DataFrame()
              for index,row in zip_match_data.iterrows():
                  left = left_data[left_data['zip_code'] == row['zip_code']]
                  right = right_data[right_data['postal_code'] == row['zip_code']]
                  result = find_matches1(left, right,threshold=0.80)
                  # concatenate the result to the empty DataFrame
                  all_results = pd.concat([all_results, result])
              all_results.to_csv(filename, index=False)
              print(all_results)
              return all_results
```

Match by State & Zip Code

- Apply this function on each state-subset and concatenate the five result files to get the final matching results

We use the same method to get matching results in each state. Each group member took care of one state. Load the five matching result files.

```
In [199]: match_PA = pd.read_csv('matches_PA_byzip.csv')
          match_FL = pd.read_csv('matches_FL_byzip.csv')
          match_MO = pd.read_csv('matches_MO_byzip.csv')
          match_TN = pd.read_csv('matches_TN_byzip.csv')
          match_IN = pd.read_csv('matches_IN_byzip.csv')
```

Concatenate five result files to one dataframe

```
In [200]: all_match = pd.concat([match_PA, match_FL, match_MO, match_TN, match_IN])
```

04
Results

Examine all matching results

```
all_match.shape
```

```
(26265, 3)
```

Number of exact matching

```
len(all_match[all_match['confidence_score'] == 1.0])
```

```
6591
```

Final results with threshold = 0.9

```
df = match_sorted[match_sorted['confidence_score'] > 0.90]
```

```
df.shape
```

```
(10692, 3)
```

Review the results and check matching quality

```
match_sorted[10900:10910]
```

	left	right	confidence_score
5370	86451	81764	0.900
3747	3793	29849	0.900
1724	18179	93459	0.900
556	87527	21337	0.900
774	32232	60962	0.900
6526	66615	69368	0.900
4949	57231	19754	0.900
3177	60841	2050	0.900
3168	60020	64165	0.900
282	43066	58150	0.899

```
left[left['business_id'] == 43066]
```

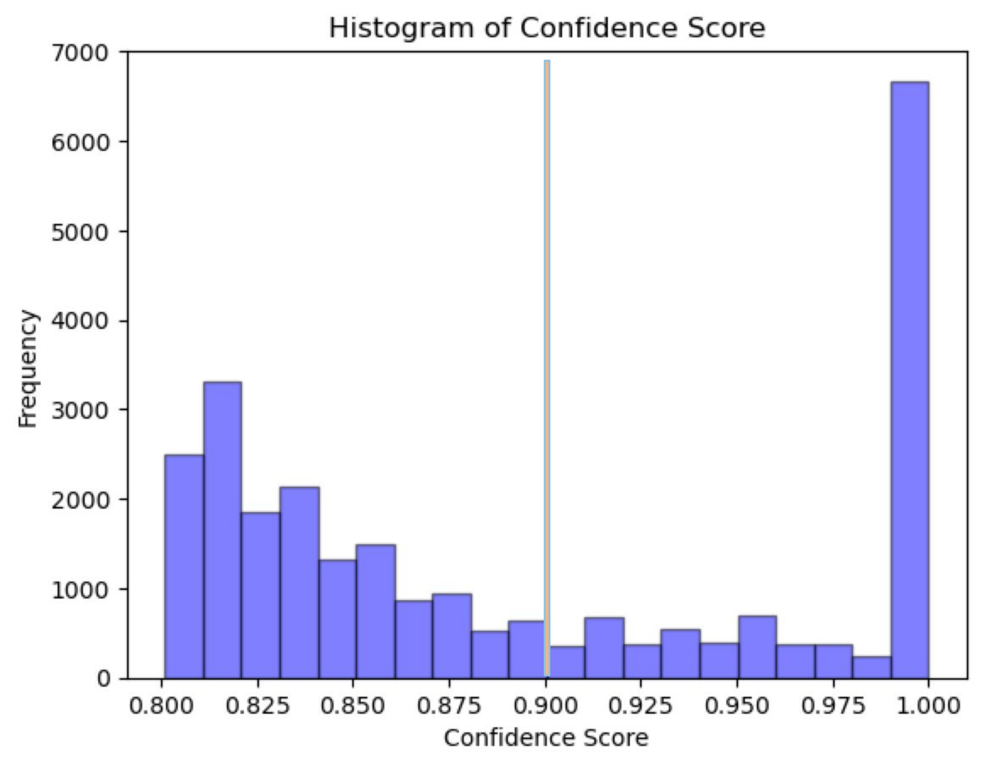
	business_id	name	address	city	state	zip_code
43065	43066	allstar moving llc	333 leffingwell avenue ste 127	saint louis	MO	63122

```
right[right['entity_id'] == 58150]
```

	entity_id	name	address	city	state	postal_code
58149	58150	all star moving llc	333 leffingwell avenue ste 127	kirkwood	MO	63122

Visualize matching results

- Choose confidence score = 0.90 as a threshold for acceptable matchings



Github Links

Yifan Zhang (yz4388): https://github.com/YifanZhang0522/5210.git

Wanghao Ying (wy2416): https://github.com/ArsWying/APAN5210-Python-Project.git

Yongze Jiang (yj2730):https://github.com/EvisudDream/Yongze-Jiang.git

Ronglu Jiang (rj2662): https://github.com/jocilulu/PROJECT-5400

Xinyi Wen(xw2897):https://github.com/wenxinyiiiiiiiii/Rainyy/blob/main/project_final.html

Thanks!