Exercise1

The cutoff point ¶

In this exercise, and throughout this chapter, you'll be working with the restaurants DataFrame which has data on various restaurants. Your ultimate goal is to create a restaurant recommendation engine, but you need to first clean your data.

This version of restaurants has been collected from many sources, where the cuisine_type column is riddled with typos, and should contain only italian, american and asian cuisine types. There are so many unique categories that remapping them manually isn't scalable, and it's best to use string similarity instead.

Before doing so, you want to establish the cutoff point for the similarity score using the fuzzywuzzy's process.extract() function by finding the similarity score of the most distant typo of each category.

Instructions

- Import process from fuzzywuzzy.
- Store the unique cuisine_types into unique_types.
- Calculate the similarity of 'asian', 'american', and 'italian' to all possible cuisine_types using process.extract(), while returning all possible matches.

In [ ]:
```
# Import process from fuzzywuzzy
____

# Store the unique values of cuisine_type in unique_types
unique_types = ____

# Calculate similarity of 'asian' to all values of unique_types
print(process.____('____', ____, limit = len(____)))

# Calculate similarity of 'american' to all values of unique_types
print(____('____', ____, ____))

# Calculate similarity of 'italian' to all values of unique_types
print(____)


#_________________________#
#Solutions

# Import process from fuzzywuzzy
from fuzzywuzzy import process

# Store the unique values of cuisine_type in unique_types
unique_types = restaurants['cuisine_type'].unique()

# Calculate similarity of 'asian' to all values of unique_types
print(process.extract('asian', unique_types, limit = len(unique_types)))

# Calculate similarity of 'american' to all values of unique_types
print(process.extract('american', unique_types, limit = len(unique_types)))

# Calculate similarity of 'italian' to all values of unique_types
print(process.extract('italian', unique_types, limit = len(unique_types)))
```

Exercise2

Remapping categories II

In the last exercise, you determined that the distance cutoff point for remapping typos of 'american', 'asian', and 'italian' cuisine types stored in the cuisine_type column should be 80.

In this exercise, you're going to put it all together by finding matches with similarity scores equal to or higher than 80 by using fuzywuzzy.process's extract() function, for each correct cuisine type, and replacing these matches with it. Remember, when comparing a string with an array of strings using process.extract(), the output is a list of tuples where each is formatted like:

(closest match, similarity score, index of match) The restaurants DataFrame is in your environment, and you have access to a categories list containing the correct cuisine types ('italian', 'asian', and 'american').

Instructions

- Return all of the unique values in the cuisine_type column of restaurants.

Okay! Looks like you will need to use some string matching to correct these misspellings!

- As a first step, create a list of all possible matches, comparing 'italian' with the restaurant types listed in the cuisine_type column.

Now you're getting somewhere! Now you can iterate through matches to reassign similar entries.

- Within the for loop, use an if statement to check whether the similarity score in each match is greater than or equal to 80.
- If it is, use .loc to select rows where cuisine_type in restaurants is equal to the current match (which is the first element of match), and reassign them to be 'italian'.

Finally, you'll adapt your code to work with every restaurant type in categories.

- Using the variable cuisine to iterate through categories, embed your code from the previous step in an outer for loop.
- Inspect the final result. This has been done for you.

In [ ]:
```
# Inspect the unique values of the cuisine_type column
print(____)

# Create a list of matches, comparing 'italian' with the cuisine_type column
matches = ____

# Inspect the first 5 matches
print(matches[0:5])

# Create a list of matches, comparing 'italian' with the cuisine_type column
matches = process.extract('italian', restaurants['cuisine_type'], limit=len(restaurants.cuisine_type))

# Iterate through the list of matches to italian
for match in matches:
  # Check whether the similarity score is greater than or equal to 80
      :
    # Select all rows where the cuisine_type is spelled this way, and set them to the correct cuisine
    ____

# Iterate through categories
for cuisine in ____:
  # Create a list of matches, comparing cuisine with the cuisine_type column
  matches = process.extract(____, restaurants['cuisine_type'], limit=len(restaurants.cuisine_type))

  # Iterate through the list of matches
  for match in matches:
     # Check whether the similarity score is greater than or equal to 80
    if match[1] >= 80:
      # If it is, select all rows where the cuisine_type is spelled this way, and set them to the correct cuisine
      restaurants.loc[restaurants['cuisine_type'] == match[0]] = ____

# Inspect the final result
print(restaurants['cuisine_type'].unique())


#_____________________#
#Solutions

# Inspect the unique values of the cuisine_type column
print(restaurants['cuisine_type'].unique())

# Create a list of matches, comparing 'italian' with the cuisine_type column
matches = process.extract('italian', restaurants['cuisine_type'], limit=len(restaurants.cuisine_type))

# Inspect the first 5 matches
print(matches[0:5])

# Create a list of matches, comparing 'italian' with the cuisine_type column
matches = process.extract('italian', restaurants['cuisine_type'], limit=len(restaurants.cuisine_type))

# Iterate through the list of matches to italian
for match in matches:
  # Check whether the similarity score is greater than or equal to 80
  if match[1] >= 80:
    # Select all rows where the cuisine_type is spelled this way, and set them to the correct cuisine
    restaurants.loc[restaurants['cuisine_type'] == match[0]] = 'italian'

# Iterate through categories
for cuisine in categories:
  # Create a list of matches, comparing cuisine with the cuisine_type column
  matches = process.extract(cuisine, restaurants['cuisine_type'], limit=len(restaurants.cuisine_type))

  # Iterate through the list of matches
  for match in matches:
     # Check whether the similarity score is greater than or equal to 80
    if match[1] >= 80:
      # If it is, select all rows where the cuisine_type is spelled this way, and set them to the correct cuisine
      restaurants.loc[restaurants['cuisine_type'] == match[0]] = cuisine

# Inspect the final result
print(restaurants['cuisine_type'].unique())
```

Exercise3

Pairs of restaurants

In the last lesson, you cleaned the restaurants dataset to make it ready for building a restaurants recommendation engine. You have a new DataFrame named restaurants_new with new restaurants to train your model on, that's been scraped from a new data source.

You've already cleaned the cuisine_type and city columns using the techniques learned throughout the course. However you saw duplicates with typos in restaurants names that require record linkage instead of joins with restaurants.

In this exercise, you will perform the first step in record linkage and generate possible pairs of rows between restaurants and restaurants_new. Both DataFrames, pandas and recordlinkage are in your environment.

Instructions

- Instantiate an indexing object by using the Index() function from recordlinkage.
- Block your pairing on cuisine_type by using indexer's' .block() method.
- Generate pairs by indexing restaurants and restaurants_new in that order.

In [ ]:
```
# Create an indexer and object and find possible pairs
indexer = ____

# Block pairing on cuisine_type
indexer.____(____)

# Generate pairs
pairs = indexer.____(____, ____)
```

```
#_________________________#
#Solutions

# Create an indexer and object and find possible pairs
indexer = recordlinkage.Index()

# Block pairing on cuisine_type
indexer.block('cuisine_type')

# Generate pairs
pairs = indexer.index(restaurants, restaurants_new)
```

Exercise4

Similar restaurants

In the last exercise, you generated pairs between restaurants and restaurants_new in an effort to cleanly merge both DataFrames using record linkage.

When performing record linkage, there are different types of matching you can perform between different columns of your DataFrames, including exact matches, string similarities, and more.

Now that your pairs have been generated and stored in pairs, you will find exact matches in the city and cuisine_type columns between each pair, and similar strings for each pair in the rest_name column. Both DataFrames, pandas and recordlinkage are in your environment.

Instructions

- Instantiate a comparison object using the recordlinkage.Compare() function.
- Use the appropriate comp_cl method to find exact matches between the city and cuisine_type columns of both DataFrames.
- Use the appropriate comp_cl method to find similar strings with a 0.8 similarity threshold in the rest_name column of both DataFrames.
- Compute the comparison of the pairs by using the .compute() method of comp_cl.

In [ ]:
```
# Create a comparison object
comp_cl = ____

# Find exact matches on city, cuisine_types
comp_cl.____('____', '____', label='city')
comp_cl.____('____', '____', label = 'cuisine_type')

# Find similar matches of rest_name
comp_cl.____('____', '____', label='name', ____ = ____)

# Get potential matches and print
potential_matches = comp_cl.____(pairs, ____, ____)
print(potential_matches)


#_________________________#
#Solutions

# Create a comparison object
comp_cl = recordlinkage.Compare()

# Find exact matches on city, cuisine_types
comp_cl.exact('city', 'city', label='city')
comp_cl.exact('cuisine_type', 'cuisine_type', label='cuisine_type')

# Find similar matches of rest_name
comp_cl.string('rest_name', 'rest_name', label='name', threshold = 0.8)

# Get potential matches and print
potential_matches = comp_cl.compute(pairs, restaurants, restaurants_new)
print(potential_matches)
```

Exercise5

Linking them together!

In the last lesson, you've finished the bulk of the work on your effort to link restaurants and restaurants_new. You've generated the different pairs of potentially matching rows, searched for exact matches between the cuisine_type and city columns, but compared for similar strings in the rest_name column. You stored the DataFrame containing the scores in potential_matches.

Now it's finally time to link both DataFrames. You will do so by first extracting all row indices of restaurants_new that are matching across the columns mentioned above from potential_matches. Then you will subset restaurants_new on these indices, then append the non-duplicate values to restaurants. All DataFrames are in your environment, alongside pandas imported as pd.

Instructions

- Isolate instances of potential_matches where the row sum is above or equal to 3 by using the .sum() method.
- Extract the second column index from matches, which represents row indices of matching record from restaurants_new by using the .get_level_values() method.
- Subset restaurants_new for rows that are not in matching_indices.
- Append non_dup to restaurants.

In [ ]:
```
# Isolate potential matches with row sum >=3
matches = ____[____.___(____) >= ____]

# Get values of second column index of matches
matching_indices = matches.____.____(____)

# Subset restaurants_new based on non-duplicate values
non_dup = ____[~restaurants_new.index.____(____)]

# Append non_dup to restaurants
full_restaurants = restaurants.____(____)
print(full_restaurants)


#_____________________#
#Solutions

# Isolate potential matches with row sum >=3
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]

# Get values of second column index of matches
matching_indices = matches.index.get_level_values(1)

# Subset restaurants_new based on non-duplicate values
non_dup = restaurants_new[~restaurants_new.index.isin(matching_indices)]

# Append non_dup to restaurants
full_restaurants = restaurants.append(non_dup)
print(full_restaurants)
```