7. Cleaning Data in Python

Chapter 1 - Common data problems

Categorical and text data can often be some of the messiest parts of a dataset due to their unstructured nature. In this chapter, you'll learn how to fix whitespace and capitalization inconsistencies in category labels, collapse multiple categories into one, and reformat strings for consistency.

Link for reference

In [ ]:
```
#importing libraries
import pandas as pd
import datetime as dt

# Common path prefix
common_path = "C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course DataCa

# Paths for the variables with double backslashes
airlines = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\airlines_final.csv')
banking = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\banking_dirty.csv')
restaurant = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2.csv')
restaurant_dirty = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2_dirty.csv')
ride_sharing = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\ride_sharing_new.csv')

print(airlines.head(6))
```

```
   Unnamed: 0    id        day      airline        destination    dest_region  \
0           0  1351    Tuesday  UNITED INTL             KANSAI           Asia
1           1   373     Friday       ALASKA  SAN JOSE DEL CABO  Canada/Mexico
2           2  2820   Thursday        DELTA        LOS ANGELES        West US
3           3  1157    Tuesday    SOUTHWEST        LOS ANGELES        West US
4           4  2992  Wednesday     AMERICAN              MIAMI        East US
5           5   634   Thursday       ALASKA            NEWARK        East US

  dest_size boarding_area   dept_time  wait_min     cleanliness  \
0       Hub  Gates 91-102  2018-12-31     115.0           Clean
1     Small   Gates 50-59  2018-12-31     135.0           Clean
2       Hub   Gates 40-48  2018-12-31      70.0         Average
3       Hub   Gates 20-39  2018-12-31     190.0           Clean
4       Hub   Gates 50-59  2018-12-31     559.0  Somewhat clean
5       Hub   Gates 50-59  2018-12-31     140.0  Somewhat clean

          safety        satisfaction
0        Neutral      Very satisfied
1      Very safe      Very satisfied
2  Somewhat safe             Neutral
3      Very safe  Somewhat satsified
4      Very safe  Somewhat satsified
5      Very safe      Very satisfied
```

Numeric data or ... ?

In [ ]:

```python
# Print the information of ride_sharing
print(ride_sharing.info())

# Print summary statistics of user_type column
print(ride_sharing['user_type'].describe())

# Convert user_type from integer to category
ride_sharing['user_type_cat'] = ride_sharing['user_type'].astype('category')

# Write an assert statement confirming the change
assert ride_sharing['user_type_cat'].dtype == 'category'

# Print new summary statistics
print(ride_sharing['user_type_cat'].describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25760 entries, 0 to 25759
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Unnamed: 0       25760 non-null  int64
 1   duration         25760 non-null  object
 2   station_A_id     25760 non-null  int64
 3   station_A_name   25760 non-null  object
 4   station_B_id     25760 non-null  int64
 5   station_B_name   25760 non-null  object
 6   bike_id          25760 non-null  int64
 7   user_type        25760 non-null  int64
 8   user_birth_year  25760 non-null  int64
 9   user_gender      25760 non-null  object
dtypes: int64(6), object(4)
memory usage: 2.0+ MB
None
count    25760.000000
mean         2.008385
std          0.704541
min          1.000000
25%          2.000000
50%          2.000000
75%          3.000000
max          3.000000
Name: user_type, dtype: float64
count     25760
unique        3
top           2
freq      12972
Name: user_type_cat, dtype: int64
```

Summing strings and concatenating numbers

In [ ]:

```python
# Strip duration of minutes
ride_sharing['duration_trim'] = ride_sharing['duration'].str.strip('minutes')

# Convert duration to integer
ride_sharing['duration_time'] = ride_sharing['duration_trim'].astype('int')

# Write an assert statement making sure of conversion
assert ride_sharing['duration_time'].dtype == 'int'

# Print formed columns and calculate average ride duration
print(ride_sharing[['duration','duration_trim','duration_time']])
print(ride_sharing['duration_time'].mean())
```

```
         duration duration_trim  duration_time
0      12 minutes           12              12
1      24 minutes           24              24
2       8 minutes            8               8
3       4 minutes            4               4
4      11 minutes           11              11
...           ...          ...             ...
25755  11 minutes           11              11
25756  10 minutes           10              10
25757  14 minutes           14              14
25758  14 minutes           14              14
25759  29 minutes           29              29

[25760 rows x 3 columns]
11.389052795031056
```

Tire size constraints

In [ ]:
```
# Convert tire_sizes to integer
ride_sharing['tire_sizes'] = ride_sharing['tire_sizes'].astype('int')

# Set all values above 27 to 27
ride_sharing.loc[ride_sharing['tire_sizes'] > 27, 'tire_sizes'] = 27

# Reconvert tire_sizes back to categorical
ride_sharing['tire_sizes'] = ride_sharing['tire_sizes'].astype('category')

# Print tire size description
print(ride_sharing['tire_sizes'].head())
```

Back to the future

In [ ]:
```
#import
import pandas as pd
import datetime as dt

import pandas as pd
import datetime as dt

# Convert ride_date to date
ride_sharing['ride_dt'] = pd.to_datetime(ride_sharing['ride_date'])

# Save today's date
today = pd.to_datetime(dt.date.today())

# Set all in the future to today's date
ride_sharing.loc[ride_sharing['ride_dt'] > today, 'ride_dt'] = today
```

```
# Print maximum of ride_dt column
print(ride_sharing['ride_dt'].max())
```

Finding duplicates

In [ ]:
```
# Find duplicates
duplicates = ride_sharing.duplicated(subset='ride_id', keep=False)

# Sort your duplicated rides
duplicated_rides = ride_sharing[duplicates].sort_values('ride_id')

# Print relevant columns of duplicated_rides
print(duplicated_rides[['ride_id','duration','user_birth_year']])
```

Treating duplicates

In [ ]:
```
# Drop complete duplicates from ride_sharing
ride_dup = ride_sharing.drop_duplicates()

# Create statistics dictionary for aggregation function
statistics = {'user_birth_year': 'min', 'duration': 'mean'}

# Group by ride_id and compute new statistics
ride_unique = ride_dup.groupby('ride_id').agg(statistics).reset_index()

# Find duplicated values again
duplicates = ride_unique.duplicated(subset = 'ride_id', keep = False)
duplicated_rides = ride_unique[duplicates == True]

# Assert duplicates are processed
assert duplicated_rides.shape[0] == 0
```

In [ ]:
```

```

Chapter 2 - Text and categorical data problems

Categorical and text data can often be some of the messiest parts of a dataset due to their unstructured nature. In this chapter, you'll learn how to fix whitespace and capitalization inconsistencies in category labels, collapse multiple categories into one, and reformat strings for consistency.

Link for reference

In [ ]:
```
#importing libraries
import pandas as pd
import datetime as dt
```

```
# Common path prefix
common_path = "C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course DataCa

# Paths for the variables with double backslashes
airlines = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\airlines_final.csv')
banking = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\banking_dirty.csv')
restaurant = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2.csv')
restaurant_dirty = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2_dirty.csv')
ride_sharing = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\ride_sharing_new.csv')

print(airlines.head(3))
```

```
   Unnamed: 0    id       day      airline        destination    dest_region  \
0           0  1351   Tuesday  UNITED INTL             KANSAI           Asia
1           1   373    Friday       ALASKA  SAN JOSE DEL CABO  Canada/Mexico
2           2  2820  Thursday        DELTA        LOS ANGELES        West US

  dest_size boarding_area   dept_time  wait_min cleanliness         safety  \
0       Hub  Gates 91-102  2018-12-31     115.0       Clean        Neutral
1     Small   Gates 50-59  2018-12-31     135.0       Clean      Very safe
2       Hub   Gates 40-48  2018-12-31      70.0     Average  Somewhat safe

     satisfaction
0  Very satisfied
1  Very satisfied
2         Neutral
```

Finding consistency

In [ ]:
```
# Print categories DataFrame
print("categories: ")
print("************************************************************")

# Print unique values of survey columns in airlines
print('Cleanliness: ', airlines['cleanliness'].unique(), "\n")
print('Safety: ', airlines['safety'].unique(), "\n")
print('Satisfaction: ', airlines['satisfaction'].unique(), "\n")
```

```
categories:
************************************************************
Cleanliness:  ['Clean' 'Average' 'Somewhat clean' 'Somewhat dirty' 'Dirty']

Safety:  ['Neutral' 'Very safe' 'Somewhat safe' 'Very unsafe' 'Somewhat unsafe']

Satisfaction:  ['Very satisfied' 'Neutral' 'Somewhat satsified' 'Somewhat unsatisfied'
 'Very unsatisfied']
```

Finding consistency 2

In [ ]:
```
# Find the cleanliness category in airlines not in categories
cat_clean = set(airlines['cleanliness']).difference(categories['cleanliness'])


# Find rows with that category
cat_clean_rows = airlines['cleanliness'].isin(cat_clean)

# Print rows with inconsistent category
print(airlines[cat_clean_rows])
```

Finding consistency 3

In [ ]:
```
# Find the cleanliness category in airlines not in categories
cat_clean = set(airlines['cleanliness']).difference(categories['cleanliness'])

# Find rows with that category
cat_clean_rows = airlines['cleanliness'].isin(cat_clean)

# Print rows with inconsistent category
print(airlines[cat_clean_rows])

# Print rows with consistent categories only
print(airlines[~cat_clean_rows])
```

Inconsistent categories

In [ ]:
```
# Print unique values of both columns
print(airlines['dest_region'].unique())
print(airlines['dest_size'].unique())

# Lower dest_region column and then replace "eur" with "europe"
airlines['dest_region'] = airlines['dest_region'].str.lower()
airlines['dest_region'] = airlines['dest_region'].replace({'eur':'europe'})

# Remove white spaces from `dest_size`
airlines['dest_size'] = airlines['dest_size'].str.strip()

# Verify changes have been effected
print(airlines['dest_region'].unique())
print(airlines['dest_size'].unique())
```

Remapping categories

In [ ]:
```
# Create ranges for categories
label_ranges = [0, 60, 180, np.inf]
label_names = ['short', 'medium', 'long']
```

```
# Create wait_type column
airlines['wait_type'] = pd.cut(airlines['wait_min'], bins = label_ranges,
                               labels = label_names)

# Create mappings and replace
mappings = {'Monday':'weekday', 'Tuesday':'weekday', 'Wednesday': 'weekday', 'Thursday': 'weekday', 'Friday': 'weekday',
            'Saturday': 'weekend', 'Sunday': 'weekend'}

airlines['day_week'] = airlines['day'].replace(mappings)
```

Removing titles and taking names

In [ ]:
```
# Replace "Dr." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Dr.","")

# Replace "Mr." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Mr.","")

# Replace "Miss" with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Miss","")

# Replace "Ms." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Ms.","")

# Assert that full_name has no honorifics
assert airlines['full_name'].str.contains('Ms.|Mr.|Miss|Dr.').any() == False
```

Keeping it descriptive

In [ ]:
```
# Store length of each row in survey_response column
resp_length = airlines['survey_response'].str.len()

# Find rows in airlines where resp_length > 40
airlines_survey = airlines[resp_length > 40]

# Assert minimum survey_response length is > 40
assert airlines_survey['survey_response'].str.len().min() > 40

# Print new survey_response column
print(airlines_survey['survey_response'])
```

Chapter 3 - Advanced data problems

In this chapter, you'll dive into more advanced data cleaning problems, such as ensuring that weights are all written in kilograms instead of pounds. You'll also gain invaluable skills that will help you verify that values have been added correctly and that missing values don't negatively impact your

analyses.

Link for reference

In [ ]:
```
import pandas as pd

# Common path prefix
common_path = "C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course DataCa

# Paths for the variables with double backslashes
airlines = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\airlines_final.csv')
banking = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\banking_dirty.csv')
restaurant = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2.csv')
restaurant_dirty = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2_dirty.csv')
ride_sharing = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\ride_sharing_new.csv')

print(banking.head(2))
```

```
   Unnamed: 0   cust_id  birth_date  Age  acct_amount  inv_amount   fund_A  \
0           0  870A9281  1962-06-09   58     63523.31       51295  30105.0
1           1  166B05B0  1962-12-16   58     38175.46       15050   4995.0

   fund_B  fund_C   fund_D account_opened last_transaction
0  4138.0  1420.0  15632.0       02-09-18         22-02-19
1   938.0  6696.0   2421.0       28-02-19         31-10-18
```

Uniform currencies

In [ ]:
```
# Find values of acct_cur that are equal to 'euro'
acct_eu = banking['acct_cur'] == 'euro'

# Convert acct_amount where it is in euro to dollars
#banking.loc[banking['acct_cur']=='euro', 'acct_amount'] = banking.loc[banking['acct_cur']=='euro', 'acct_amount'] * 1.1
banking.loc[acct_eu, 'acct_amount'] = banking.loc[acct_eu, 'acct_amount'] * 1.1
# Unify acct_cur column by changing 'euro' values to 'dollar'
banking.loc[acct_eu, 'acct_cur'] = 'dollar'

# Assert that only dollar currency remains
assert banking['acct_cur'].unique() == 'dollar'
```

Uniform dates

In [ ]:
```
# Print the header of account_opened
print(banking['account_opened'].head())
```

In [ ]:
```
# Print the header of account_opened
print(banking['account_opened'].head())
```

```
# Convert account_opened to datetime
banking['account_opened'] = pd.to_datetime(banking['account_opened'],
                                           # Infer datetime format
                                           infer_datetime_format = True,
                                           # Return missing value for error
                                           errors = 'coerce')
```

In [ ]:
```
# Print the header of account_opend
print(banking['account_opened'].head())

# Convert account_opened to datetime
banking['account_opened'] = pd.to_datetime(banking['account_opened'],
                                           # Infer datetime format
                                           infer_datetime_format = True,
                                           # Return missing value for error
                                           errors = 'coerce')

# Get year of account opened
banking['acct_year'] = banking['account_opened'].dt.strftime('%Y')

# Print acct_year
print(banking['acct_year'])
```

How's our data integrity?

In [ ]:
```
# Store fund columns to sum against
fund_columns = ['fund_A', 'fund_B', 'fund_C', 'fund_D']

# Find rows where fund_columns row sum == inv_amount
inv_equ = banking[fund_columns].sum(axis=1) == banking['inv_amount']

# Store consistent and inconsistent data
consistent_inv = banking[inv_equ]
inconsistent_inv = banking[~inv_equ]

# Store consistent and inconsistent data
print("Number of inconsistent investments: ", inconsistent_inv.shape[0])
```

How's our data integrity? 2

In [ ]:
```
# Store today's date and find ages
today = dt.date.today()
ages_manual = today.year - banking['birth_date'].dt.year

# Find rows where age column == ages_manual
age_equ = ages_manual == banking['age']
```

```
# Store consistent and inconsistent data
consistent_ages = banking[age_equ]
inconsistent_ages = banking[~age_equ]

# Store consistent and inconsistent data
print("Number of inconsistent ages: ", inconsistent_ages.shape[0])
```

Missing investors

In [ ]:
```
#import
import pandas as pd
import missingno as msno
import matplotlib.pyplot as plt


# Print number of missing values in banking
print(banking.isna().sum())

# Visualize missingness matrix
msno.matrix(banking)
plt.show()

# Isolate missing and non missing values of inv_amount
missing_investors = banking[banking['inv_amount'].isna()]
investors = banking[~banking['inv_amount'].isna()]

# Sort banking by age and visualize
banking_sorted = banking.sort_values('age')
msno.matrix(banking_sorted)
plt.show()
```

Follow the money

In [ ]:
```
# Drop missing values of cust_id
banking_fullid = banking.dropna(subset = ['cust_id'])

# Compute estimated acct_amount
acct_imp = banking_fullid['inv_amount']*5

# Impute missing acct_amount with corresponding acct_imp
banking_imputed = banking_fullid.fillna({'acct_amount':acct_imp})

# Print number of missing values
print(banking_imputed.isna().sum())
```

In [ ]:
```

```

Chapter 4 - Record linkage

Record linkage is a powerful technique used to merge multiple datasets together, used when values have typos or different spellings. In this chapter, you'll learn how to link records by calculating the similarity between strings—you'll then use your new skills to join two restaurant review datasets into one clean master dataset. Other references.

Link for reference

In [ ]:
```
#import last modification
import pandas as pd
import missingno as msno
import matplotlib.pyplot as plt

# Common path prefix
common_path = "C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course DataCa

# Paths for the variables with double backslashes
airlines = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\airlines_final.csv')
banking = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\banking_dirty.csv')
restaurant = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2.csv')
restaurant_dirty = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\restaurants_L2_dirty.csv')
ride_sharing = pd.read_csv(f'{common_path}\\7. Cleaning Data in Python\\datasets\\ride_sharing_new.csv')

print(restaurant_dirty.head(8))
```

```
   Unnamed: 0      name                        addr         city  \
0           0    kokomo            6333 w. third st.           la
1           1    feenix      8358 sunset blvd. west    hollywood
2           2   parkway         510 s. arroyo pkwy .     pasadena
3           3      r-23             923 e. third st.  los angeles
4           4     gumbo            6333 w. third st.           la
5           5    pink's          709 n. la brea ave.           la
6           6  original  875 s. figueroa st. downtown           la
7           7  21 clubs               21 w. 52nd st.      new york

        phone          type
0  2139330773      american
1  2138486677      american
2  8187951001   californian
3  2136877178      japanese
4  2139330358  cajun/creole
5  2139314223      hot dogs
6  2136276879        diners
7  2125827200      american
```

The cutoff point

In [ ]:
```
#install
        #pip install python-Levenshtein thefuzz
#import
        #from thefuzz import process



# Import process from thefuzz
from thefuzz import process #this name "the fuzz" was given randonmly in the course!!!!!

# Store the unique values of cuisine_type in unique_types
unique_types = restaurant_dirty['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)))
```

```
[('asian', 100), ('indonesian', 80), ('californian', 68), ('italian', 67), ('russian', 67), ('american', 62), ('japanese', 54),
('mexican/tex-mex', 54), ('american ( new )', 54), ('mexican', 50), ('fast food', 45), ('middle eastern', 43), ('steakhouses', 4
0), ('pacific new wave', 40), ('pizza', 40), ('diners', 36), ('cajun/creole', 36), ('vietnamese', 36), ('continental', 36), ('sea
food', 33), ('chicken', 33), ('chinese', 33), ('hot dogs', 30), ('hamburgers', 30), ('coffee shops', 30), ('noodle shops', 30),
('southern/soul', 30), ('desserts', 30), ('eclectic', 26), ('coffeebar', 26), ('health food', 22), ('french ( new )', 22), ('deli
s', 20)]
[('american', 100), ('american ( new )', 90), ('mexican', 80), ('mexican/tex-mex', 72), ('asian', 62), ('italian', 53), ('russia
n', 53), ('californian', 53), ('middle eastern', 51), ('southern/soul', 47), ('pacific new wave', 45), ('hamburgers', 44), ('indo
nesian', 44), ('cajun/creole', 42), ('chicken', 40), ('pizza', 40), ('japanese', 38), ('eclectic', 38), ('delis', 36), ('french (
new )', 34), ('vietnamese', 33), ('diners', 29), ('seafood', 27), ('chinese', 27), ('desserts', 25), ('coffeebar', 24), ('steakho
uses', 21), ('health food', 21), ('continental', 21), ('coffee shops', 20), ('noodle shops', 20), ('fast food', 12), ('hot dogs',
0)]
[('italian', 100), ('asian', 67), ('californian', 56), ('continental', 54), ('american', 53), ('indonesian', 47), ('russian', 4
3), ('mexican', 43), ('japanese', 40), ('mexican/tex-mex', 39), ('american ( new )', 39), ('pacific new wave', 39), ('middle east
ern', 38), ('vietnamese', 35), ('delis', 33), ('pizza', 33), ('steakhouses', 33), ('health food', 33), ('diners', 31), ('cajun/cr
eole', 30), ('chicken', 29), ('chinese', 29), ('southern/soul', 28), ('eclectic', 27), ('noodle shops', 22), ('french ( new )', 1
8), ('seafood', 14), ('hot dogs', 13), ('desserts', 13), ('fast food', 12), ('coffeebar', 12), ('hamburgers', 12), ('coffee shop
s', 0)]
```

Remapping categories II

In [ ]:
```
# Inspect the unique values of the cuisine_type column
print(restaurant_dirty['type'].unique())
```

```
['american' 'californian' 'japanese' 'cajun/creole' 'hot dogs' 'diners'
 'delis' 'hamburgers' 'seafood' 'italian' 'coffee shops' 'russian'
 'steakhouses' 'mexican/tex-mex' 'noodle shops' 'mexican' 'middle eastern'
 'asian' 'vietnamese' 'health food' 'american ( new )' 'pacific new wave'
 'indonesian' 'eclectic' 'chicken' 'fast food' 'southern/soul' 'coffeebar'
 'continental' 'french ( new )' 'desserts' 'chinese' 'pizza']
```

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

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

```
[('italian', 100, 14), ('italian', 100, 21), ('italian', 100, 47), ('italian', 100, 57), ('italian', 100, 73)]
```

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

# 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
    restaurant_dirty.loc[restaurant_dirty['type'] == match[0], 'type'] = 'italian'
```

In [ ]:
```
# List of predefined categories
categories = ['Asian', 'American', 'Italian', 'Mexican', 'French']


# Iterate through categories
for cuisine in categories:
    # Create a list of matches, comparing cuisine with the cuisine_type column
    matches = process.extract(cuisine, restaurant_dirty['type'], limit=len(restaurant_dirty.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
            restaurant_dirty.loc[restaurant_dirty['type'] == match[0]] = cuisine

# Inspect the final result
restaurant_dirty['type'].unique()
```

```
C:\Users\yeiso\AppData\Local\Temp\ipykernel_38696\3801345375.py:15: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise in a future error of pandas. Value 'Asian' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  restaurant_dirty.loc[restaurant_dirty['type'] == match[0]] = cuisine
```

Out[ ]:
```
array(['Mexican', 'californian', 'japanese', 'cajun/creole', 'hot dogs',
       'diners', 'delis', 'hamburgers', 'seafood', 'Italian',
       'coffee shops', 'russian', 'steakhouses', 'noodle shops',
       'middle eastern', 'Asian', 'vietnamese', 'health food',
       'pacific new wave', 'eclectic', 'chicken', 'fast food',
       'southern/soul', 'coffeebar', 'continental', 'French', 'desserts',
       'chinese', 'pizza'], dtype=object)
```

Pairs of restaurants

In [ ]:
```
# first install this package
#pip install recordlinkage

# Import the required library
import recordlinkage

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

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

# Generate pairs
pairs = indexer.index(restaurant_dirty, restaurant)
```

Similar restaurants

In [ ]:
```
# Create a comparison object
comp_cl = recordlinkage.Compare()
```

In [ ]:
```
# 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)
```

Out[ ]:
```
<Compare>
```

In [ ]:
```
# Get potential matches and print
potential_matches = comp_cl.compute(pairs, restaurant_dirty, restaurant)
print(potential_matches)
```

Linking them together!

In [ ]:
```
# 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 = restaurant_dirty[~restaurant_dirty.index.isin(matching_indices)]

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