Data Curation Air Quality

December 2, 2023

1 Motivation

Our choice of the air quality project stems from its direct impact on public health, environmental significance, and interdisciplinary nature. Addressing data challenges in air quality monitoring, our project aims to innovate with technology, bridge research gaps, and foster collaboration for meaningful policy and decision-making impact.

2 Data Quality Issues

Inconsistent Date Formats: Both datasets had inconsistent date formats. In one data set the dates were in datetime format while in another it was in string format and the representation was different as well.

Missing Values: Addressing missing values is paramount. Whether due to equipment malfunctions or other reasons, imputing or handling these gaps is essential for a comprehensive dataset. Our curation focuses on mitigating the impact of missing data points.

Outliers / Bad Measurements: Outliers or inaccuracies in air quality measurements are common. Our curation process identifies and handles these outliers, ensuring data integrity and the reliability of insights derived from the dataset. By addressing these challenges head-on, our curation strategy aims to enhance the quality and reliability of air quality datasets, paving the way for more accurate analyses and informed decision-making.

3 Data Integration Issues

Creating Mappings: To harmonize disparate datasets, we create mappings between corresponding columns. This facilitates seamless integration and ensures accurate alignment of data points.

Standardizing Column Names: Standardizing column names is crucial for consistency. Whether it's 'date,' 'location,' or other variables, uniformity simplifies the integration process and fosters clarity in the combined dataset.

Handling Different Data Types: Air quality datasets may exhibit variations in data types among columns. Our integration process includes thoughtful conversion and standardization to ensure uniformity in data representation.

Handling Duplicate Rows: Duplicate rows can introduce biases and inaccuracies. Our integration strategy involves addressing and resolving duplicate entries to maintain data integrity and avoid overrepresentation.

4 About the Data Sources

Here's a brief description of each column based on common data elements found in air quality datasets:

Unique ID: This likely represents a unique identifier for each record in the dataset. Having zero missing values here is crucial because it means every record can be distinctly identified. Indicator ID: This could refer to a code or number that uniquely identifies a specific air quality indicator being measured, such as levels of a particular pollutant. Name: This column might contain the names of the locations or monitoring stations where air quality data has been collected.

Measure: This likely indicates the actual measurements of air quality, which could be expressed in parts per million or another relevant unit.

Measure Info: This might provide additional details about the measurements, such as the methods used to collect them or the standards against which they are being assessed.

Geo Type Name: This could refer to the type of geographical unit used in the dataset, such as 'city,' 'county,' or 'region.'

Geo Join ID: This may be an identifier used to link or 'join' this dataset with other geographic information, potentially in a spatial database or a Geographic Information System (GIS).

Geo Place Name: This column likely contains the names of geographic places associated with each data record, which might require entity resolution as seen in previous slides.

Time Period: This could represent the time frame over which the data was collected, such as a specific year or range of dates.

Start Date: This presumably indicates the starting date of the measurement period for each record.

Data Value: This might contain the air quality data values themselves, representing the concentration of pollutants or air quality index (AQI) readings.

Message: This column could contain messages or notes related to the data, such as qualifiers or explanations for anomalies.

The count next to each column name indicates the number of missing values. In this output, 'Geo Place Name', 'Data Value', and 'Message' columns have missing data, with 'Message' having the most missing entries. This information is crucial for understanding the completeness of your dataset and determining the next steps in the data cleaning process. It's important to decide how to handle these missing values, whether it's through imputation, deletion, or some other method of data correction.

[1]:
```python
import pandas as pd
import numpy as np
import difflib
```

[2]:
```python
data = pd.read_csv('Air_Quality.csv', names = ['Unique_ID', 'Indicator_ID',
 ↪'Name', 'Measure', 'Measure_Info', 'Geo_Type_Name', 'Geo_Join_ID',
 ↪'Geo_Place_Name', 'Time_Period', 'Start_Date', 'Data_Value','Message'])
```

[3]:
```python
data.head()
```

[3]:

```
    Unique_ID  Indicator_ID                    Name  Measure  Measure_Info  \
0  Unique ID  Indicator ID                    Name  Measure  Measure Info
1     172653           375  Nitrogen dioxide (NO2)     Mean           ppb
2     172585           375  Nitrogen dioxide (NO2)     Mean           ppb
3     336637           375  Nitrogen dioxide (NO2)     Mean           ppb
4     336622           375  Nitrogen dioxide (NO2)     Mean           ppb

    Geo_Type_Name  Geo_Join_ID                     Geo_Place_Name  \
0  Geo Type Name  Geo Join ID                     Geo Place Name
1          UHF34          203  Bedford Stuyvesant - Crown Heights
2          UHF34          203  Bedford Stuyvesant - Crown Heights
3          UHF34          204                      East New York
4          UHF34          103                 Fordham - Bronx Pk

           Time_Period  Start_Date  Data_Value  Message
0          Time Period  Start_Date  Data Value  Message
1  Annual Average 2011  12/01/2010        25.3      NaN
2  Annual Average 2009  12/01/2008       26.93      NaN
3  Annual Average 2015  01/01/2015       19.09      NaN
4  Annual Average 2015  01/01/2015       19.76      NaN
```

[4]:

```python
# data = vizierdb.get_data_frame('air_quality')

missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 Unique_ID            0
Indicator_ID         0
Name                 0
Measure              0
Measure_Info         0
Geo_Type_Name        0
Geo_Join_ID          0
Geo_Place_Name       0
Time_Period          0
Start_Date           0
Data_Value           0
Message          16218
dtype: int64
```

[5]:

```python
#handling missing value
del data["Message"]
```

[6]:

```python
#duplicates
duplicate_rows = data[data.duplicated()]
print("Duplicate Rows:\n", duplicate_rows)
data = data.drop_duplicates()
```

```
Duplicate Rows:
 Empty DataFrame
Columns: [Unique_ID, Indicator_ID, Name, Measure, Measure_Info, Geo_Type_Name,
Geo_Join_ID, Geo_Place_Name, Time_Period, Start_Date, Data_Value]
Index: []
```

The line of code you see below—data['Geo_Place_Name'].unique()—is a command in pandas, which is a Python library for data manipulation and analysis. What this function does is extract all the unique values from the 'Geo_Place_Name' column within our dataset.

Why is this important? Well, it's not uncommon to find discrepancies in categorical data. For instance, the same place might be entered into the database as 'Central Park' in one entry and 'central park' in another. To the human eye, they are clearly the same, but to a computer algorithm, they would be treated as two separate entities. This can lead to skewed data analysis and unreliable outcomes. By identifying these unique values, we can assess the extent of the inconsistency. The next step, which follows this identification process, is to standardize these values to ensure uniformity across our dataset. This may involve converting all text to a consistent case, removing leading and trailing spaces, or even more complex string-matching methods to consolidate similar names.

Through this essential step, we enhance the quality of our dataset and lay a strong foundation for accurate and reliable analysis. As we move forward, you will see how this step integrates into the larger data cleaning workflow that we've established for our project.

[7]:
```python
# Check unique values for categorical inconsistency
print("Unique Values in 'Geo_Place_Name':\n", data['Geo_Place_Name'].unique())
#addressing inconsistencies
data['Geo_Place_Name'] = data['Geo_Place_Name'].str.lower().str.strip()
print("Unique Values in 'Geo_Place_Name':\n", data['Geo_Place_Name'].unique())
```

```
Unique Values in 'Geo_Place_Name':
 ['Geo Place Name' 'Bedford Stuyvesant - Crown Heights' 'East New York'
 'Fordham - Bronx Pk' 'Pelham - Throgs Neck' 'Chelsea-Village'
 'Borough Park' 'High Bridge - Morrisania' 'Bensonhurst - Bay Ridge'
 'Coney Island - Sheepshead Bay' 'Rockaways'
 'Mott Haven and Melrose (CD1)' 'Financial District (CD1)'
 'Greenwich Village and Soho (CD2)' 'Woodside and Sunnyside (CD2)'
 'Greenpoint' 'Kingsbridge - Riverdale' 'Northeast Bronx' 'West Queens'
 'Washington Heights' 'Hunts Point - Mott Haven'
 'East Flatbush - Flatbush' 'Canarsie - Flatlands' 'Southwest Queens'
 'Morrisania and Crotona (CD3)' 'Lower East Side and Chinatown (CD3)'
 'Central Harlem - Morningside Heights' 'Downtown - Heights - Slope'
 'Bronx' 'Williamsburg - Bushwick' 'Northern SI' 'Port Richmond'
 'Upper East Side (CD8)' 'Central Harlem (CD10)'
 'Washington Heights and Inwood (CD12)'
 'Bay Ridge and Dyker Heights (CD10)' 'Borough Park (CD12)'
 'Flushing - Clearview' 'Lower Manhattan' 'Southeast Queens' 'East Harlem'
 'Upper West Side' 'Upper East Side-Gramercy'
 'Hunts Point and Longwood (CD2)' 'Brooklyn' 'Jamaica' 'Sunset Park'
 'Upper West Side (CD7)' 'Bayside Little Neck-Fresh Meadows'
```

```
 'Rockaway and Broad Channel (CD14)' 'Staten Island'
 'Long Island City - Astoria' 'South Bronx' 'Coney Island (CD13)'
 'Queens Village (CD13)' 'Queens' 'Throgs Neck and Co-op City (CD10)'
 'Williamsbridge and Baychester (CD12)' 'Jamaica and Hollis (CD12)'
 'Southern SI' 'Crotona -Tremont' 'Clinton and Chelsea (CD4)'
 'Stuyvesant Town and Turtle Bay (CD6)' 'Belmont and East Tremont (CD6)'
 'Riverdale and Fieldston (CD8)' 'Upper East Side' 'Fresh Meadows'
 'Fordham and University Heights (CD5)' 'Midtown (CD5)'
 'Chelsea - Clinton' 'St. George and Stapleton (CD1)' 'Manhattan'
 'South Beach and Willowbrook (CD2)' 'Greenpoint and Williamsburg (CD1)'
 'Highbridge and Concourse (CD4)' 'Gramercy Park - Murray Hill'
 'Greenwich Village - SoHo' 'East Flatbush (CD17)'
 'Flatlands and Canarsie (CD18)' 'Elmhurst and Corona (CD4)'
 'South Ozone Park and Howard Beach (CD10)'
 'Rego Park and Forest Hills (CD6)' 'Hillcrest and Fresh Meadows (CD8)'
 'Park Slope and Carroll Gardens (CD6)'
 'Crown Heights and Prospect Heights (CD8)'
 'Kingsbridge Heights and Bedford (CD7)' 'South Beach - Tottenville'
 'Flatbush and Midwood (CD14)' 'Flushing and Whitestone (CD7)'
 'Bedford Stuyvesant (CD3)' 'Bushwick (CD4)'
 'Tottenville and Great Kills (CD3)' 'Jackson Heights (CD3)'
 'Union Square - Lower East Side' 'Long Island City and Astoria (CD1)'
 'Fort Greene and Brooklyn Heights (CD2)' 'Sheepshead Bay (CD15)'
 'Stapleton - St. George' 'Morningside Heights and Hamilton Heights (CD9)'
 'East Harlem (CD11)' 'Bensonhurst (CD11)'
 'Parkchester and Soundview (CD9)' 'Morris Park and Bronxdale (CD11)'
 'Kew Gardens and Woodhaven (CD9)' 'Bayside and Little Neck (CD11)'
 'South Crown Heights and Lefferts Gardens (CD9)' 'Willowbrook'
 'Brownsville (CD16)' 'East New York and Starrett City (CD5)'
 'Ridgewood and Maspeth (CD5)' 'Sunset Park (CD7)'
 'Ridgewood - Forest Hills' 'Union Square-Lower Manhattan'
 'Bayside - Little Neck' 'New York City']
Unique Values in 'Geo_Place_Name':
 ['geo place name' 'bedford stuyvesant - crown heights' 'east new york'
 'fordham - bronx pk' 'pelham - throgs neck' 'chelsea-village'
 'borough park' 'high bridge - morrisania' 'bensonhurst - bay ridge'
 'coney island - sheepshead bay' 'rockaways'
 'mott haven and melrose (cd1)' 'financial district (cd1)'
 'greenwich village and soho (cd2)' 'woodside and sunnyside (cd2)'
 'greenpoint' 'kingsbridge - riverdale' 'northeast bronx' 'west queens'
 'washington heights' 'hunts point - mott haven'
 'east flatbush - flatbush' 'canarsie - flatlands' 'southwest queens'
 'morrisania and crotona (cd3)' 'lower east side and chinatown (cd3)'
 'central harlem - morningside heights' 'downtown - heights - slope'
 'bronx' 'williamsburg - bushwick' 'northern si' 'port richmond'
 'upper east side (cd8)' 'central harlem (cd10)'
 'washington heights and inwood (cd12)'
 'bay ridge and dyker heights (cd10)' 'borough park (cd12)'
```

```
 'flushing - clearview' 'lower manhattan' 'southeast queens' 'east harlem'
 'upper west side' 'upper east side-gramercy'
 'hunts point and longwood (cd2)' 'brooklyn' 'jamaica' 'sunset park'
 'upper west side (cd7)' 'bayside little neck-fresh meadows'
 'rockaway and broad channel (cd14)' 'staten island'
 'long island city - astoria' 'south bronx' 'coney island (cd13)'
 'queens village (cd13)' 'queens' 'throgs neck and co-op city (cd10)'
 'williamsbridge and baychester (cd12)' 'jamaica and hollis (cd12)'
 'southern si' 'crotona -tremont' 'clinton and chelsea (cd4)'
 'stuyvesant town and turtle bay (cd6)' 'belmont and east tremont (cd6)'
 'riverdale and fieldston (cd8)' 'upper east side' 'fresh meadows'
 'fordham and university heights (cd5)' 'midtown (cd5)'
 'chelsea - clinton' 'st. george and stapleton (cd1)' 'manhattan'
 'south beach and willowbrook (cd2)' 'greenpoint and williamsburg (cd1)'
 'highbridge and concourse (cd4)' 'gramercy park - murray hill'
 'greenwich village - soho' 'east flatbush (cd17)'
 'flatlands and canarsie (cd18)' 'elmhurst and corona (cd4)'
 'south ozone park and howard beach (cd10)'
 'rego park and forest hills (cd6)' 'hillcrest and fresh meadows (cd8)'
 'park slope and carroll gardens (cd6)'
 'crown heights and prospect heights (cd8)'
 'kingsbridge heights and bedford (cd7)' 'south beach - tottenville'
 'flatbush and midwood (cd14)' 'flushing and whitestone (cd7)'
 'bedford stuyvesant (cd3)' 'bushwick (cd4)'
 'tottenville and great kills (cd3)' 'jackson heights (cd3)'
 'union square - lower east side' 'long island city and astoria (cd1)'
 'fort greene and brooklyn heights (cd2)' 'sheepshead bay (cd15)'
 'stapleton - st. george' 'morningside heights and hamilton heights (cd9)'
 'east harlem (cd11)' 'bensonhurst (cd11)'
 'parkchester and soundview (cd9)' 'morris park and bronxdale (cd11)'
 'kew gardens and woodhaven (cd9)' 'bayside and little neck (cd11)'
 'south crown heights and lefferts gardens (cd9)' 'willowbrook'
 'brownsville (cd16)' 'east new york and starrett city (cd5)'
 'ridgewood and maspeth (cd5)' 'sunset park (cd7)'
 'ridgewood - forest hills' 'union square-lower manhattan'
 'bayside - little neck' 'new york city']
```

Let's dive into what we mean by 'functional dependency.' In a dataset, we expect certain data columns to be dependent on others. For instance, if we have a column for 'City' and another for 'State,' whenever we see 'New York' in the 'City' column, it should always correspond to 'NY' in the 'State' column. If this isn't the case, we've got an inconsistency that needs fixing.

The function you see here, naive_fd_repair, takes on this task. It works by ensuring that for each unique value in column A, only one corresponding value in column B is allowed, maintaining the principle of functional dependency A -> B.

Let's break down the code:

We begin by loading our air quality data into a pandas DataFrame. We define our function, naive_fd_repair, which takes a subset of the DataFrame and two column names as arguments.

Inside the function, we iterate over each row, checking for violations of our dependency rule. If a violation is detected, we correct it by updating column B to match the value corresponding to column A in the first occurrence. We've applied this function to several columns, ensuring that our dataset respects the established dependencies and thus preventing any misleading analysis due to data inconsistency. In summary, this function is a key component of our data preprocessing, ensuring that our dataset is not only clean but also logically coherent.

[8]:
```
#data = vizierdb.get_data_frame('air_quality')


def naive_fd_repair(df, A, B):
    """
    Naïve FD Repair for a single functional dependency A -> B on the first 10␣
 ↪rows.
    :param df: Pandas DataFrame.
    :param A: The column name of attribute A.
    :param B: The column name of attribute B.
    :return: Repaired DataFrame.
    """
    # Working on a copy of the first 10 rows of the DataFrame
    df_subset = df.head(200).copy()

    # Iterate over each row
    for i in range(len(df_subset)):
        for j in range(i+1, len(df_subset)):
            # Check if there's a violation of the functional dependency A -> B
            #print('in fooooooooooooooooooooooooooooor')
            if df_subset.iloc[i][A] == df_subset.iloc[j][A] and df_subset.
 ↪iloc[i][B] != df_subset.iloc[j][B]:
                # Resolve the violation by updating B in the second row
                print( df_subset.iloc[j, df_subset.columns.get_loc(B)],␣
 ↪df_subset.iloc[i][B])
                df_subset.iloc[j, df_subset.columns.get_loc(B)] = df_subset.
 ↪iloc[i][B]

    return df_subset
# Assuming 'data' is your DataFrame
print(data.columns)
naive_fd_repair(data,'Time_Period','Start_Date')
naive_fd_repair(data,'Geo_Join_ID', 'Geo_Place_Name')
naive_fd_repair(data,'Geo_Type_Name', 'Geo_Join_ID')
```

```
Index(['Unique_ID', 'Indicator_ID', 'Name', 'Measure', 'Measure_Info',
       'Geo_Type_Name', 'Geo_Join_ID', 'Geo_Place_Name', 'Time_Period',
       'Start_Date', 'Data_Value'],
      dtype='object')
morrisania and crotona (cd3) bedford stuyvesant - crown heights
morrisania and crotona (cd3) bedford stuyvesant - crown heights
```

```
morrisania and crotona (cd3) bedford stuyvesant - crown heights
morrisania and crotona (cd3) bedford stuyvesant - crown heights
lower east side and chinatown (cd3) fordham - bronx pk
lower east side and chinatown (cd3) fordham - bronx pk
lower east side and chinatown (cd3) fordham - bronx pk
lower east side and chinatown (cd3) fordham - bronx pk
greenpoint mott haven and melrose (cd1)
greenpoint mott haven and melrose (cd1)
greenpoint mott haven and melrose (cd1)
greenpoint mott haven and melrose (cd1)
kingsbridge - riverdale financial district (cd1)
kingsbridge - riverdale financial district (cd1)
kingsbridge - riverdale financial district (cd1)
kingsbridge - riverdale financial district (cd1)
kingsbridge - riverdale financial district (cd1)
kingsbridge - riverdale financial district (cd1)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
northeast bronx greenwich village and soho (cd2)
west queens woodside and sunnyside (cd2)
west queens woodside and sunnyside (cd2)
west queens woodside and sunnyside (cd2)
west queens woodside and sunnyside (cd2)
west queens woodside and sunnyside (cd2)
upper west side (cd7) hunts point - mott haven
upper west side (cd7) hunts point - mott haven
upper west side (cd7) hunts point - mott haven
upper west side (cd7) hunts point - mott haven
upper west side (cd7) hunts point - mott haven
hunts point and longwood (cd2) downtown - heights - slope
lower manhattan bay ridge and dyker heights (cd10)
lower manhattan bay ridge and dyker heights (cd10)
lower manhattan bay ridge and dyker heights (cd10)
204 203
103 203
104 203
104 203
306308 203
306308 203
204 203
204 203
103 203
104 203
```

```
201 203
101 203
101 203
102 203
102 203
402 203
402 203
301 203
301 203
302 203
302 203
302 203
302 203
202 203
202 203
101 203
101 203
102 203
102 203
102 203
209 203
209 203
209 203
211 203
211 203
211 203
211 203
501502 203
501502 203
501502 203
209 203
209 203
209 203
211 203
211 203
211 203
104 203
403 203
403 203
303 203
304 203
204 203
204 203
204 203
103 203
103 203
104 203
104 203
```

```
403 203
305307 203
305307 203
305307 203
306308 203
302 203
302 203
302 203
201 203
202 203
101 203
102 203
102 203
102 203
402 203
402 203
402 203
301 203
205 203
409 203
205 203
205 203
404406 203
404406 203
205 203
205 203
205 203
409 203
106 206
106 206
106 206
209 206
209 206
210 206
210 206
210 206
410 206
107 206
107 206
107 206
207 206
207 206
208 206
407 206
107 206
107 206
201 206
201 206
```

101 206
501 206
501 206
501 206
310 206
310 206
310 206
209 206
209 206
210 206
210 206
410 206
409 206
207 206
208 206
208 206
407 206
407 206
408 206
408 206
107 206
107 206
204 206
204 206
104 206
104 206
211 206
211 206
204 206
104 206
211 206
304 206
304 206
304 206
304 206
304 206
101 201
101 201
102 201
102 201
102 201
402 201
402 201
203 201
203 201
103 201
108 201
108 201

```
110 201
112 201
310 201
310 201
310 201
310 201
310 201
312 201
312 201
202 201
203 201
203 201
103 201
103 201
103 201
107 201
107 201
107 201
107 201
414 201
414 201
414 201
107 201
2 1
2 1
2 1
5 1
```

[8]:

```
     Unique_ID  Indicator_ID                    Name  Measure  Measure_Info  \
0    Unique ID  Indicator ID                    Name  Measure  Measure Info
1       172653           375  Nitrogen dioxide (NO2)     Mean           ppb
2       172585           375  Nitrogen dioxide (NO2)     Mean           ppb
3       336637           375  Nitrogen dioxide (NO2)     Mean           ppb
4       336622           375  Nitrogen dioxide (NO2)     Mean           ppb
..         …             …                       …        …             …
195     212437           375  Nitrogen dioxide (NO2)     Mean           ppb
196     643383           375  Nitrogen dioxide (NO2)     Mean           ppb
197     667183           375  Nitrogen dioxide (NO2)     Mean           ppb
198     179432           375  Nitrogen dioxide (NO2)     Mean           ppb
199     165922           375  Nitrogen dioxide (NO2)     Mean           ppb

     Geo_Type_Name  Geo_Join_ID                       Geo_Place_Name  \
0    Geo Type Name  Geo Join ID                       geo place name
1            UHF34          203  bedford stuyvesant - crown heights
2            UHF34          203  bedford stuyvesant - crown heights
3            UHF34          203                        east new york
4            UHF34          203                   fordham - bronx pk
```

```
..            …          …                              …
195          UHF42          206                   upper west side
196          UHF42          206                   upper west side
197          UHF42          206                   upper west side
198        Borough            1                     staten island
199             CD          201             upper west side (cd7)

            Time_Period  Start_Date  Data_Value
0           Time Period  Start_Date  Data Value
1   Annual Average 2011  12/01/2010        25.3
2   Annual Average 2009  12/01/2008       26.93
3   Annual Average 2015  01/01/2015       19.09
4   Annual Average 2015  01/01/2015       19.76
..                  …         …          …
195         Summer 2014  06/01/2014       20.87
196      Winter 2018-19  12/01/2018       25.25
197 Annual Average 2020  01/01/2020       19.62
198 Annual Average 2011  12/01/2010       16.17
199      Winter 2011-12  12/01/2011       28.89

[200 rows x 11 columns]
```

[9]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def optimized_entity_resolution(df, column_name, threshold=0.8):
    """
    An optimized entity resolution function using TF-IDF and cosine similarity.

    :param df: DataFrame containing the data.
    :param column_name: The name of the column to perform entity resolution on.
    :param threshold: Similarity threshold for considering matches.
    :return: DataFrame with potential matches.
    """
    # Convert the column to string and drop duplicates
    unique_strings = df[column_name].astype(str).drop_duplicates()

    # Create a TF-IDF Vectorizer
    vectorizer = TfidfVectorizer().fit(unique_strings)
    tfidf_matrix = vectorizer.transform(unique_strings)

    # Calculate cosine similarity
    similarity_matrix = cosine_similarity(tfidf_matrix)

    # Extract potential matches above the threshold
    potential_matches = []
    for i in range(similarity_matrix.shape[0]):
```

```
        for j in range(i+1, similarity_matrix.shape[1]):
            if similarity_matrix[i, j] >= threshold:
                potential_matches.append((unique_strings.iloc[i],␣
↪unique_strings.iloc[j]))

    return pd.DataFrame(potential_matches, columns=[column_name + '_1',␣
↪column_name + '_2'])

# Example usage
# resolved_entities = optimized_entity_resolution(air_quality_data, 'Name')
# print(resolved_entities.head())


resolved_entities = optimized_entity_resolution(data, 'Geo_Place_Name')
print(resolved_entities.head())
```

```
                  Geo_Place_Name_1          Geo_Place_Name_2
0                     borough park       borough park (cd12)
1  greenwich village and soho (cd2)  greenwich village - soho
2              upper east side (cd8)           upper east side
3                    upper west side      upper west side (cd7)
4          upper east side-gramercy           upper east side
```

In the realm of data analysis, especially when dealing with large and diverse datasets, one common task is to identify and merge records that refer to the same entity, a process known as entity resolution. The code shown here performs entity resolution using two powerful tools from the Python ecosystem: TF-IDF and cosine similarity. Let's break down how this works.

First, we convert the values in our column of interest to strings and remove any duplicates. This is crucial because we want to compare unique entities only, avoiding redundant computations.

Next, we create a TF-IDF Vectorizer. This tool transforms the text into a numerical representation that reflects not just the frequency of words but also their importance in the context of the dataset.

We then compute the cosine similarity for each pair of entities. Cosine similarity measures how similar two documents are, regardless of their size. In our case, it helps us to determine how similar two place names are. The function iterates through the similarity matrix, extracting pairs of entities that are above a certain threshold—in this case, 0.8. This threshold can be adjusted based on how strict or lenient we want to be with our matching criteria.

Finally, we return a DataFrame with potential matches. This is our output on the right side of the slide. As you can see, it pairs up entities like 'Greenwich Village and Soho (CD2)' with 'Greenwich Village - Soho,' indicating they are likely referring to the same place despite the slight variation in naming.

By applying this method, we can vastly reduce the complexity and improve the accuracy of our dataset, ensuring that each unique place is represented consistently

[ ]:

```

```