Big Data Final Project Report, Group 5

YIMING LI (YL7538), New York University, USA
PINSHUO YE (PY2033), New York University, USA
CHUANYU ZHOU (CZ2117), New York University, USA

.

ACM Reference Format:
Yiming Li (yl7538), Pinshuo Ye (py2033), and Chuanyu Zhou (cz2117). 2021. Big Data Final Project Report, Group 5. 1, 1 (December 2021), 9 pages.

1 INTRODUCTION

OpenClean is an open source python library for data cleaning that supports data transformation using existing values, data standardization using KNN clustering, and data enrichment using master data. [7] While OpenClean provides decent data cleaning methods, we found it difficult to reapply our existing cleaning pipelines to other similar datasets. Therefore, we propose a wrapping layer above OpenClean methods that can be easily reapplied and reproduced. Our wrapper class utilizes OpenClean methods to provide easy-to-use and easy-to-understand functions that can be reapplied to different datasets. Relevant code, data, and test notebooks are uploaded in GitHub repository: https://github.com/CharlesPoletowin/BigDataProject

2 PROBLEM FORMULATION

There are two main problems in this project:

- Build up an OpenClean pipeline and apply it to one dataset
- Warp a pipeline using our wrapping layer and apply it to multiple datasets

3 RELATED WORK

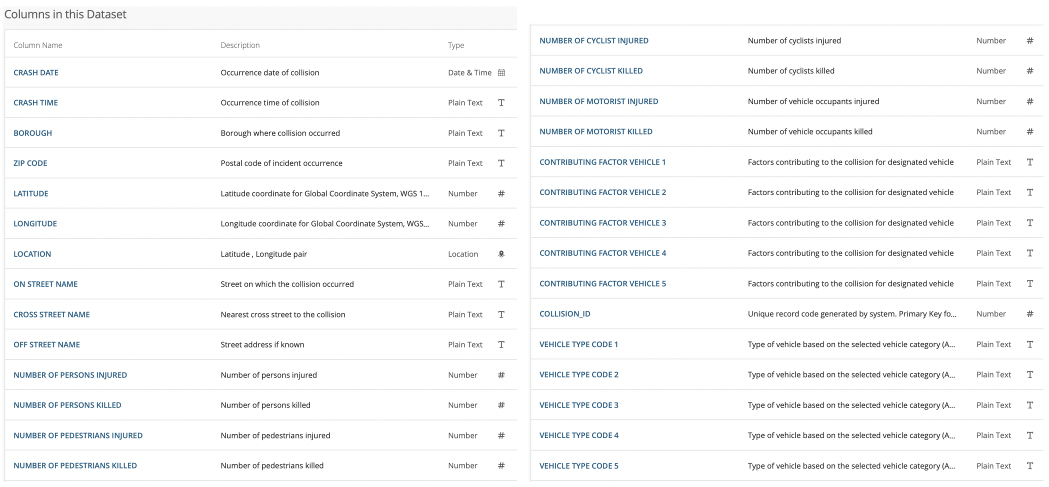
There are numerous data cleaning libraries or toolkit such as Dora[4], datacleaner[18], and PrettyPandas[5]. Dora is able to read missing or poorly scaled data and impute missing values. Datacleaner is able to drop missing values or replacing missing values with the mode or median on a column-by-column basis. PrettyPandas is able to create summaries, add styling, and format numbers, columns, and rows of a DataFrame. [2]

4 METHODS, ARCHITECTURE AND DESIGN

As summarized in chapter 2, in order to design and build our wrapping layer for OpenClean, we need to build data cleaning pipelines using OpenClean in the first place. After building and testing our OpenClean pipelines on one sample dataset, we will extract methods we used for different kind of columns and wrap them separately as new functions. Then, we should test and improve our new functions using other datasets with same or similar columns as the sample dataset. Lastly, we can put all new functions together in a class as our desired wrapping layer.

4.1 Sample Dataset

We will build pipelines to clean the "Motor Vehicle Collisions - Crashes"[8] dataset from NYC open data[15]. As in the figure 1, the dataset has 29 columns and 4 types of values.



Columns in this Dataset

Column Name	Description	Type	
CRASH DATE	Occurrence date of collision	Date & Time	📅
CRASH TIME	Occurrence time of collision	Plain Text	T
BOROUGH	Borough where collision occurred	Plain Text	T
ZIP CODE	Postal code of incident occurrence	Plain Text	T
LATITUDE	Latitude coordinate for Global Coordinate System, WGS 1...	Number	#
LONGITUDE	Longitude coordinate for Global Coordinate System, WGS...	Number	#
LOCATION	Latitude , Longitude pair	Location	📍
ON STREET NAME	Street on which the collision occurred	Plain Text	T
CROSS STREET NAME	Nearest cross street to the collision	Plain Text	T
OFF STREET NAME	Street address if known	Plain Text	T
NUMBER OF PERSONS INJURED	Number of persons injured	Number	#
NUMBER OF PERSONS KILLED	Number of persons killed	Number	#
NUMBER OF PEDESTRIANS INJURED	Number of pedestrians injured	Number	#
NUMBER OF PEDESTRIANS KILLED	Number of pedestrians killed	Number	#
NUMBER OF CYCLIST INJURED	Number of cyclists injured	Number	#
NUMBER OF CYCLIST KILLED	Number of cyclists killed	Number	#
NUMBER OF MOTORIST INJURED	Number of vehicle occupants injured	Number	#
NUMBER OF MOTORIST KILLED	Number of vehicle occupants killed	Number	#
CONTRIBUTING FACTOR VEHICLE 1	Factors contributing to the collision for designated vehicle	Plain Text	T
CONTRIBUTING FACTOR VEHICLE 2	Factors contributing to the collision for designated vehicle	Plain Text	T
CONTRIBUTING FACTOR VEHICLE 3	Factors contributing to the collision for designated vehicle	Plain Text	T
CONTRIBUTING FACTOR VEHICLE 4	Factors contributing to the collision for designated vehicle	Plain Text	T
CONTRIBUTING FACTOR VEHICLE 5	Factors contributing to the collision for designated vehicle	Plain Text	T
COLLISION_ID	Unique record code generated by system. Primary Key fo...	Number	#
VEHICLE TYPE CODE 1	Type of vehicle based on the selected vehicle category (A...	Plain Text	T
VEHICLE TYPE CODE 2	Type of vehicle based on the selected vehicle category (A...	Plain Text	T
VEHICLE TYPE CODE 3	Type of vehicle based on the selected vehicle category (A...	Plain Text	T
VEHICLE TYPE CODE 4	Type of vehicle based on the selected vehicle category (A...	Plain Text	T
VEHICLE TYPE CODE 5	Type of vehicle based on the selected vehicle category (A...	Plain Text	T

Fig. 1. Profile of Motor Vehicle Collisions dataset

We can divide the dataset's 29 columns into 5 catalogues:

(1) Should/Do not need to be cleaned
(2) Can be cleaned using data transformation
(3) Can be cleaned using data standardization
(4) Can be cleaned using data enrichment
(5) Can be cleaned by simply filling empties

One example of catalogue # 1 is the column "COLLISION_ID". It is the primary key of the dataset so it should not be cleaned.

One example of catalogue #2 is the column "NUMBER OF PERSONS INJURED". It has missing values that can be filled using existing values from columns "NUMBER OF PEDESTRIANS INJURED", "NUMBER OF CYCLIST INJURED", and "NUMBER OF MOTORIST INJURED".

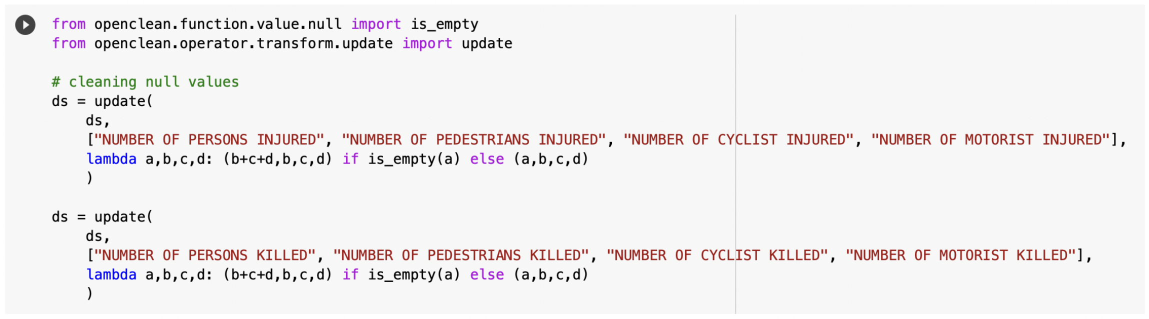
One example of catalogue #3 is the column "VEHICLE TYPE CODE 1". It has values with typo that can be standardized using KNN clustering over all values.

One example of catalogue #4 is the column "ZIP CODE". It is related to the column "LOCATION" so that we can find a master dataset that maps location to zip code and use it as reference to fill the column "ZIP CODE".

One example of catalogue #5 is the column "ON STREET NAME". It has values with typo as catalogue #3 columns, however, it has values that will be clustered incorrectly.

4.2 Building OpenClean Pipeline

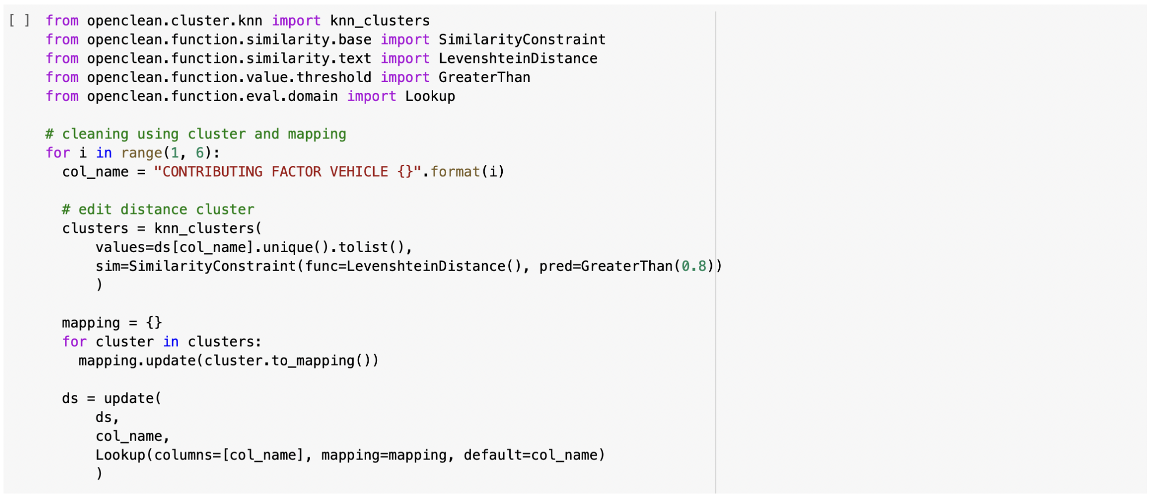
To clean catalogue #2 columns in section 4.1, we will utilize the "update" method from OpenClean. As in figure 2, we simply adding values from required columns and fill in the target column.



```
from openclean.function.value.null import is_empty
from openclean.operator.transform.update import update

# cleaning null values
ds = update(
    ds,
    ["NUMBER OF PERSONS INJURED", "NUMBER OF PEDESTRIANS INJURED", "NUMBER OF CYCLIST INJURED", "NUMBER OF MOTORIST INJURED"],
    lambda a,b,c,d: (b+c+d,b,c,d) if is_empty(a) else (a,b,c,d)
    )

ds = update(
    ds,
    ["NUMBER OF PERSONS KILLED", "NUMBER OF PEDESTRIANS KILLED", "NUMBER OF CYCLIST KILLED", "NUMBER OF MOTORIST KILLED"],
    lambda a,b,c,d: (b+c+d,b,c,d) if is_empty(a) else (a,b,c,d)
    )
```

Fig. 2. Method update

To clean catalogue #3 columns in section 4.1, we will utilize the "knn_clusters" method from OpenClean. As in figure 3, we first calculate clustering in the target column and then use the clusters as a mapping from typos to correct values.



```
[ ] from openclean.cluster.knn import knn_clusters
    from openclean.function.similarity.base import SimilarityConstraint
    from openclean.function.similarity.text import LevenshteinDistance
    from openclean.function.value.threshold import GreaterThan
    from openclean.function.eval.domain import Lookup

    # cleaning using cluster and mapping
    for i in range(1, 6):
      col_name = "CONTRIBUTING FACTOR VEHICLE {}".format(i)

      # edit distance cluster
      clusters = knn_clusters(
          values=ds[col_name].unique().tolist(),
          sim=SimilarityConstraint(func=LevenshteinDistance(), pred=GreaterThan(0.8))
          )

      mapping = {}
      for cluster in clusters:
        mapping.update(cluster.to_mapping())

      ds = update(
          ds,
          col_name,
          Lookup(columns=[col_name], mapping=mapping, default=col_name)
          )
```

Fig. 3. Method knn_clusters

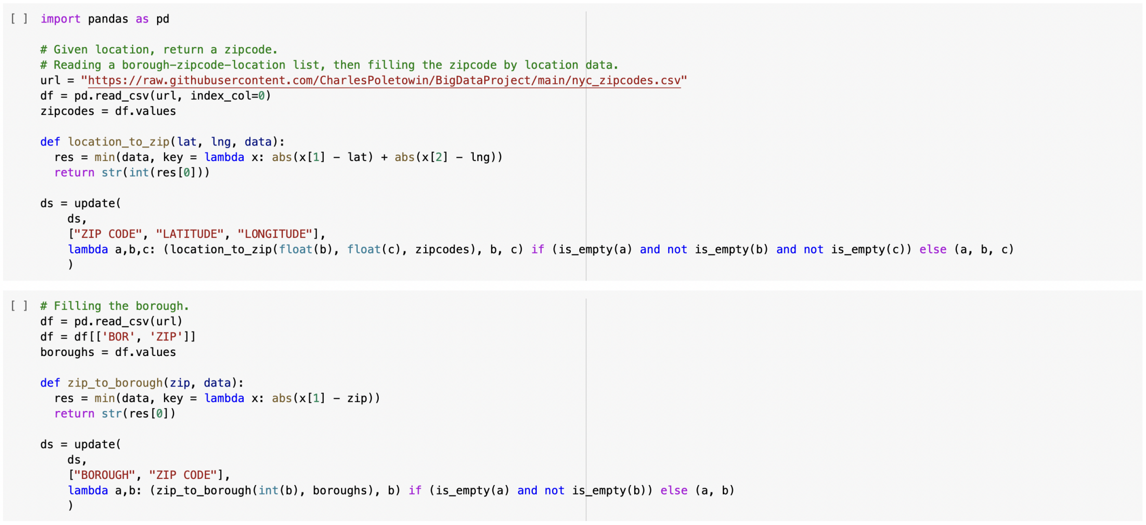
To clean catalogue #4 columns in section 4.1, we build a master dataset using data from two websites[3][6] and use the master dataset as reference to fill empty values, as shown in figure 4.

To clean catalogue #5 columns in section 4.1, we simply fill in empty values using string "Unknown", shown in figure 5.

After applied our OpenClean pipelines to the sample dataset[8], we managed to fill up all values in all columns using values from the dataset, values from master dataset, or simply "Unknown". We achieved relatively good precision of the 75% and recall of 100%.

4.3 Wrapping Methods

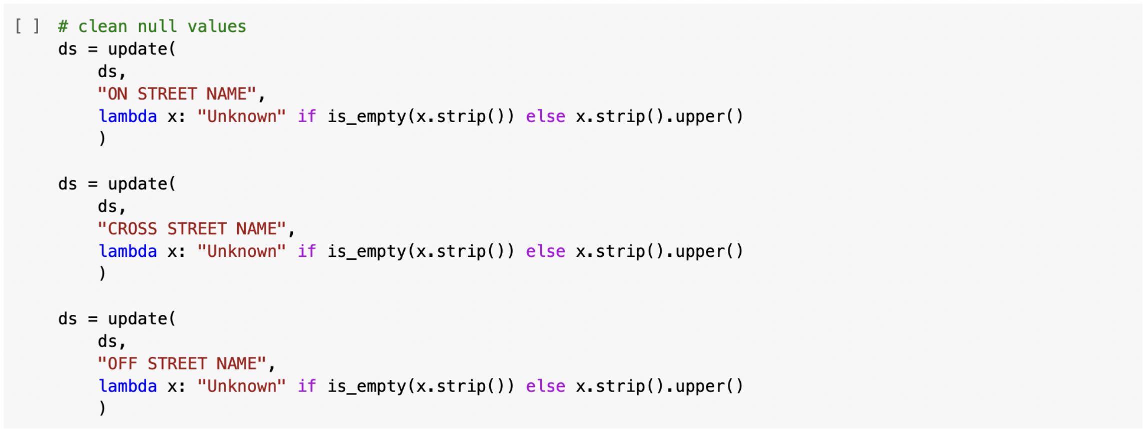
While our OpenClean pipelines are capable to achieve good cleaning results in our sample dataset, we need to expand its scalability and apply it to other datasets with similar columns. Therefore, we extract methods we used for different



```
[ ] import pandas as pd

    # Given location, return a zipcode.
    # Reading a borough-zipcode-location list, then filling the zipcode by location data.
    url = "https://raw.githubusercontent.com/CharlesPoletowin/BigDataProject/main/nyc_zipcodes.csv"
    df = pd.read_csv(url, index_col=0)
    zipcodes = df.values

    def location_to_zip(lat, lng, data):
      res = min(data, key = lambda x: abs(x[1] - lat) + abs(x[2] - lng))
      return str(int(res[0]))

    ds = update(
        ds,
        ["ZIP CODE", "LATITUDE", "LONGITUDE"],
        lambda a,b,c: (location_to_zip(float(b), float(c), zipcodes), b, c) if (is_empty(a) and not is_empty(b) and not is_empty(c)) else (a, b, c)
        )
```

```
[ ] # Filling the borough.
    df = pd.read_csv(url)
    df = df[['BOR', 'ZIP']]
    boroughs = df.values

    def zip_to_borough(zip, data):
      res = min(data, key = lambda x: abs(x[1] - zip))
      return str(res[0])

    ds = update(
        ds,
        ["BOROUGH", "ZIP CODE"],
        lambda a,b: (zip_to_borough(int(b), boroughs), b) if (is_empty(a) and not is_empty(b)) else (a, b)
        )
```

Fig. 4. Method master data

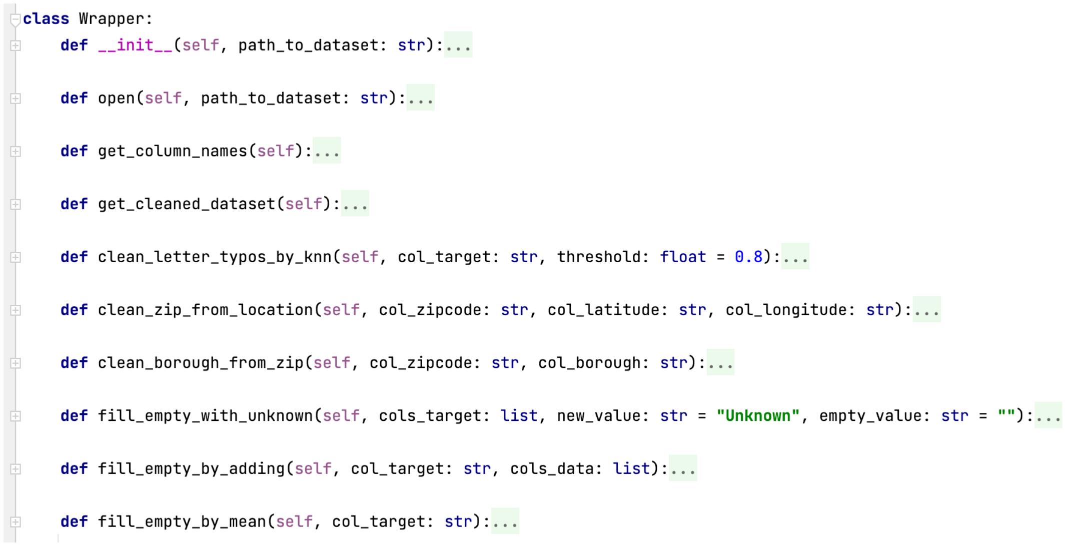


```
[ ] # clean null values
    ds = update(
        ds,
        "ON STREET NAME",
        lambda x: "Unknown" if is_empty(x.strip()) else x.strip().upper()
        )

    ds = update(
        ds,
        "CROSS STREET NAME",
        lambda x: "Unknown" if is_empty(x.strip()) else x.strip().upper()
        )

    ds = update(
        ds,
        "OFF STREET NAME",
        lambda x: "Unknown" if is_empty(x.strip()) else x.strip().upper()
        )
```

Fig. 5. Filling empty values

kind of columns, wrap them separately as new functions, and put all functions in a wrapper class. As shown in figure 6, our wrapper class reads an uncleaned dataset from a string "path", which can either be a location path or internet URL, and provides high level functions to solve different cleaning problems.

- The "open" function receives path to a new dataset and reads it in memory.
- The "get_column_names" function returns all columns' name for user.
- The "get_cleaned_dataset" function returns the current dataset.
- The "clean_letter_typos_by_knn" function utilizes the OpenClean pipeline in figure 3 to clean the internal dataset.



```python
class Wrapper:
    def __init__(self, path_to_dataset: str):...

    def open(self, path_to_dataset: str):...

    def get_column_names(self):...

    def get_cleaned_dataset(self):...

    def clean_letter_typos_by_knn(self, col_target: str, threshold: float = 0.8):...

    def clean_zip_from_location(self, col_zipcode: str, col_latitude: str, col_longitude: str):...

    def clean_borough_from_zip(self, col_zipcode: str, col_borough: str):...

    def fill_empty_with_unknown(self, cols_target: list, new_value: str = "Unknown", empty_value: str = ""):...

    def fill_empty_by_adding(self, col_target: str, cols_data: list):...

    def fill_empty_by_mean(self, col_target: str):...
```

Fig. 6. Class Wrapper

- The "clean_zip_from_location" function and the "clean_borough_from_zip" function utilize the OpenClean pipeline in figure 4 to clean the internal dataset.
- The "fill_empty_with_unknown" function, the "fill_empty_by_adding" function, and the "fill_empty_by_mean" function utilize the OpenClean pipeline in figure 2, figure 5, and functions from the library pandas[19].

4.4 Evaluation on Other Datasets

We evaluated functions of our wrapper class on tasks including converting location value to zip code value and borough value, merging letter typos to one value with KNN, and filling empty slots in the dataset with "Unknown".

- **Datasets Involved**

  We obtained all following datasets from NYC open data website[15]:

  1. erm2-nwe9: 311 Service Requests from 2010 to present[1]
  2. bty7-2jhb: Historical DOB Permit Issuance Housing & Development[11]
  3. 43nn-pn8j: DOHMH New York City Restaurant Inspection Results[12]
  4. 59kj-x8nc: Housing Litigations[13]
  5. dpec-ucu7: TLC New Driver Application Status[20]
  6. hg8x-zxpr: Housing New York Units by Building[14]
  7. 9a87-6m4x: SBS ICAP Contract Opportunities[17]
  8. uip8-fykc: NYPD Arrest Data (Year to Date)[10]
  9. bm4k-52h4: Motor Vehicle Collisions - Vehicles[9]
  10. uvpi-gqnh: 2015 Street Tree Census - Tree Data[16]

After observing all datasets by column data types and a few row of their original data, we found appropriate columns for our wrapper class to clean. For the task of converting location data, we chose columns from dataset erm2-nwe9, bty7-2jhb, 43nn-pn8j, 59kj-x8nc, hg8x-zxpr, and 9a87-6m4x. For the task of merging typos, we chose columns from dataset erm2-nwe9, uip8-fykc, bm4k-52h4, and uvpi-gqnh.

- **Measuring Effectiveness**

For location to zipcode method, our goal is to fill all rows that zipcode are empty but location data are filled, so that we can using conversion from wrapping methods to calculate the empty slots in new datasets. Note that in order to compare the results and calculate accuracy, we use data that are already filled zipcode.

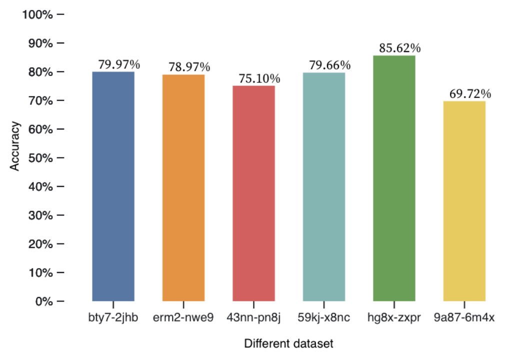
As for merging similar items to one categories with KNN, the goal is to correct some mistakes of the data, and ideally categorize similar data. We apply the same method KNN on new datasets, and the distinct items are reduced.

Filling empty slot are trivial. For those columns that have missing information but cannot be cleaned by other methods, we simply filling it with "unknown".

a. Precision and Recall of Converting Location Data

For converting location data to zip, we select data with complete information (including zip column) from other datasets, so that we can compare the calculated result with the actual information. We examined 6 different datasets that contains specific "zipcode" (or postcode or relavant items) column, and calculate the accuracy for each dataset in figure 7.

As for the recall of datasets, we want to calculate the relevant items retrieved. That is the ratio of the available location data to all the data, because we can only calculate zipcode when location data are available. In most of the cases our recall percentage is high. Here the recall shows in figure 7.



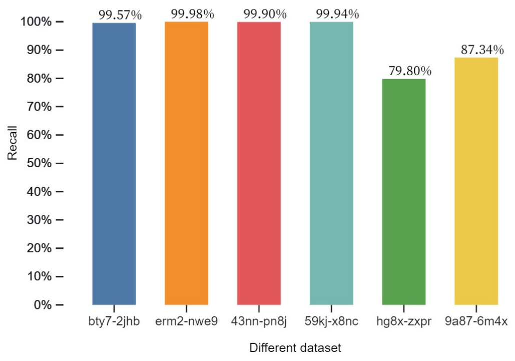


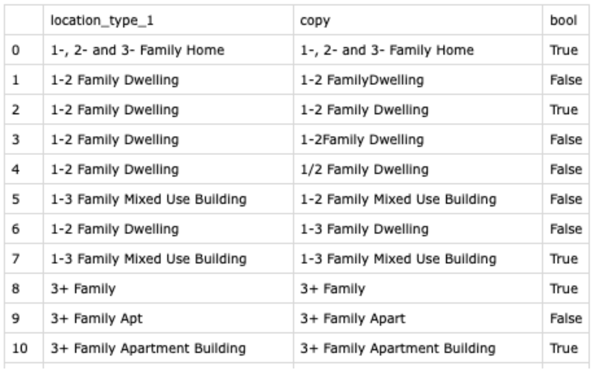
Fig. 7. Accuracy and recall of converting location data

b. Precision and Recall of Merging Typos

For converting similar strings using KNN, one way is to manually inspect the data to check precision. We have to examine the converted results, check if the string is converted to our desired result. Or we can also check the stats for all converted data by KNN method, manually choose and calculate the overall count of the corrected results.

Here are some example original data and merged results for "erm2-nwe9: 311 Service Requests from 2010 to present[1]" (boolean values check if corresponding row is modified):

As for the recall of the dataset, we have to manually inpect the results and the original data, count the amount of correctly conveted results and results that "needed to be converted" in original data. We have to decide which data fits the definition of "needed to be converted". Then we can calculate the ratio between the two.



	location_type_1	copy	bool
0	1-, 2- and 3- Family Home	1-, 2- and 3- Family Home	True
1	1-2 Family Dwelling	1-2 FamilyDwelling	False
2	1-2 Family Dwelling	1-2 Family Dwelling	True
3	1-2 Family Dwelling	1-2Family Dwelling	False
4	1-2 Family Dwelling	1/2 Family Dwelling	False
5	1-3 Family Mixed Use Building	1-2 Family Mixed Use Building	False
6	1-2 Family Dwelling	1-3 Family Dwelling	False
7	1-3 Family Mixed Use Building	1-3 Family Mixed Use Building	True
8	3+ Family	3+ Family	True
9	3+ Family Apt	3+ Family Apart	False
10	3+ Family Apartment Building	3+ Family Apartment Building	True

Fig. 8. Merged Result (left) and Original Data (right) for erm2-nwe9[1]

Based on the previous analysis, we implement the method and get "confusion matrices" (Table 1 - Table 4) to calculate the precision and recall for the following datasets that we pick from NYC Open Data specifically:

Using the formula and the confusion matrices to calculate the precision and recall. We also analyzed if KNN method is appropriate to be used on each dataset by oberving the precision and recall:

Total=	Predicted Positive	Predicted Negative
Actual Positive	8	6
Actual Negative	5	174

Table 1. Confusion Matrix for erm2-nwe9[1]

For dataset erm2-nwe9[1]: Precision = 61.54%, Recall = 57.14%. Both precision and recall perform good, because most of the typos rows have references or similar data, so our KNN method can correct them appropriately according to those similar rows; And there are typos in this column of data, so that KNN can detect them easily, that why recall of the dataset is good.

Total=	Predicted Positive	Predicted Negative
Actual Positive	4	3
Actual Negative	0	55

Table 2. Confusion Matrix for uip8-fykc[10]

For dataset uip8-fykc[10]: Precision = 100.00%, Recall = 57.14%. Both precision and recall perform good, and the reasons are similar to previous dataset erm2-nwe9[1].

Total=	Predicted Positive	Predicted Negative
Actual Positive	4	186
Actual Negative	8	302

Table 3. Confusion Matrix for bm4k-52h4[9]

For dataset bm4k-52h4[9]: Precision = 66.67%, Recall = 2.11%. The precision of the method is good, because most of the typos only have minor mistakes; but the recall is low because there are large amount of messy typos in this dataset, and most of these typos are not in the similar format, so it is difficult for the KNN method to find and correct them.

Total=	Predicted Positive	Predicted Negative
Actual Positive	1	0
Actual Negative	2	128

Table 4. Confusion Matrix for uvpi-gqnh[16]

For dataset uvpi-gqnh[16]: Precision = 33.33%, Recall = 100.00%. The precision is low in this dataset because almost all data in this colum are correct, it is likely to make mistakes when using the KNN method to correct the data that are already correct; And also because the dataset almost has no typos, it is very unlikely for KNN method to miss some typos in the dataset.

For a easier overlook of our KNN method on different datasets, here are the precision and recall barplots in figure 9:

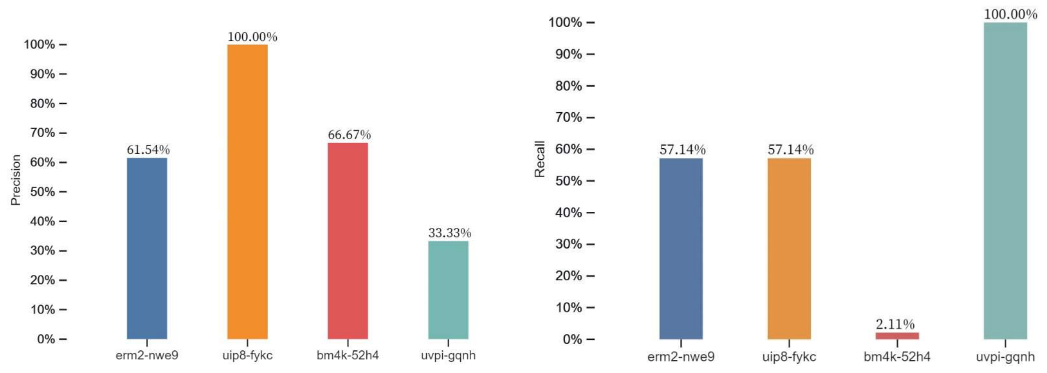


Fig. 9. Precision and Recall of Converting Using KNN

5 RESULTS

We presented a class Wrapper, which is an abstraction layer above several OpenClean methods. Our class utilized OpenClean methods to provide easy-to-use and easy-to-understand functions that can be reapplied to different datasets. As shown in section 4.4, we evaluated Wrapper on multiple datasets specified in section 4.3 and our class was proven to be fairly usable and effective. While we only wrapped three kinds of cleaning pipelines in our current Wrapper, we think that out class can be further improved by adding more pipelines and modifying current pipelines to fit more types of values.

REFERENCES

[1] DoITT 311. 2021. *311 Service Requests from 2010 to present.* https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

[2] Melissa Bierly. 2016. *8 Handy Python Libraries for Formatting and Cleaning Data.* https://mode.com/blog/python-data-cleaning-libraries/

[3] NYC by Natives. 2021. *New York City Zip Codes.* https://www.nycbynatives.com/nyc_info/new_york_city_zip_codes.php

[4] Nathan Epstein. 2016. *Dora.* https://github.com/NathanEpstein/Dora

[5] Henry Hammond. 2016. *PrettyPandas.* https://github.com/HHammond/PrettyPandas

[6] Eric Hurst. 2013. *US Zip Codes from 2013 Government Data.* https://gist.github.com/erichurst/7882666

[7] Heiko Müller, Sonia Castelo, Munaf Qazi, and Juliana Freire. 2021. From Papers to Practice: The Openclean Open-Source Data Cleaning Library. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2763–2766. https://doi.org/10.14778/3476311.3476339

[8] Police Department (NYPD). 2021. *Motor Vehicle Collisions - Crashes.* https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

[9] Police Department (NYPD). 2021. *Motor Vehicle Collisions - Vehicles.* https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Vehicles/bm4k-52h4

[10] Police Department (NYPD). 2021. *NYPD Arrest Data (Year to Date).* https://data.cityofnewyork.us/Public-Safety/NYPD-Arrest-Data-Year-to-Date-/uip8-fykc

[11] Department of Buildings (DOB). 2021. *Historical DOB Permit Issuance Housing Development.* https://data.cityofnewyork.us/Housing-Development/Historical-DOB-Permit-Issuance/bty7-2jhb

[12] Department of Health and Mental Hygiene (DOHMH). 2021. *DOHMH New York City Restaurant Inspection Results.* https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j

[13] Department of Housing Preservation and Development (HPD). 2021. *Housing Litigations.* https://data.cityofnewyork.us/Housing-Development/Housing-Litigations/59kj-x8nc

[14] Department of Housing Preservation and Development (HPD). 2021. *Housing New York Units by Building.* https://data.cityofnewyork.us/Housing-Development/Housing-New-York-Units-by-Building/hg8x-zxpr

[15] City of New York. 2021. *Open Data for All New Yorkers.* https://opendata.cityofnewyork.us

[16] Department of Parks and Recreation (DPR). 2015. *2015 Street Tree Census - Tree Data.* https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh

[17] Department of Small Business Services (SBS). 2021. *SBS ICAP Contract Opportunities.* https://data.cityofnewyork.us/Business/SBS-ICAP-Contract-Opportunities/9a87-6m4x

[18] Randy Olson and Will McGinnis. 2016. *datacleaner: Basic functionality.* https://doi.org/10.5281/zenodo.47063

[19] The pandas development team. 2020. *pandas-dev/pandas: Pandas.* https://doi.org/10.5281/zenodo.3509134

[20] Taxi and Limousine Commission (TLC). 2021. *TLC New Driver Application Status.* https://data.cityofnewyork.us/Transportation/TLC-New-Driver-Application-Status/dpec-ucu7