311 Service Requests from 2010 to present
Data Profiling and Cleaning

GitHub Repository: https://github.com/Zankar100/BigDataGroup23

Bansi Shah
(Net Id: bks7385)
bks7385@nyu.edu

Zankar Murudkar
(Net Id: zm2117)
zm2117@nyu.edu

Tirth Patel
(Net Id: tmp9936)
tmp9936@nyu.edu

ABSTRACT

In this paper, we describe the data profiling and cleaning techniques, data integrity on NYC open data 311 Service requests from 2010 to present.

Keywords

311 Service requests, NYC open data, data cleaning, data profiling, parking violations, .

1. INTRODUCTION

This is NYU CS-9223 Big data Management & Analysis final project report. We had assigned 311 Service requests from 2010-present to analyze. To analyze the dataset, firstly we find anomalies and integrity in dataset. After that we done the data cleaning and data profiling to get rid of invalid and dirty data. Then we are having summary of the how we done the cleaning of dataset.

2. PROBLEM FORMULATION

In data issue detection part, we investigate that the dataset had four major integrity issues: null values problem, unexpected type problem, mapping problem, range problem. From the mentioned anomalies and errors, we cleaned the dataset based on the detection of the integrity issues.

3. METHODS, ARCHITECTURE AND DESIGN

3.1 Data Issues

3.1.1 NULL Values:

We see some columns with extremely large amounts of null values. The column values seem to be specified for certain incidents and requests.

We also find numerous columns with repetitive or non-distinct data; such columns could be termed as class related columns. These columns have values which belong to different classes.

Column	Null Condition
Unique Key	Cannot be null and must be a number string
Created Date	Cannot be null
Closed Date	Can be null but only if the complaint is not closed
Agency	Cannot be null
Agency Name	Cannot be null
Complaint Type	Cannot be null
Descriptor	Can be null
Location Type	
Incident Zip	Cannot be null
Incident Address	Cannot be null
Street Name	Cannot be null
Cross Street 1	Cannot be null
Cross Street 2	Cannot be null
Address Type	Cannot be null
City	Cannot be null
Landmark	Can be null but not if incident location is specified as landmark
Facility Type	Can be null
Status	Cannot be null
Due Date	Cannot be null
Resolution Description	Cannot be null
Resolution Action Updated Date	Cannot be null

Community Board	Cannot be null
BBL	Cannot be null
Borough	Cannot be null
X Coordinate (State Plane)	Cannot be null
Y Coordinate (State Plane)	Cannot be null
Open Data Channel Type	
Park Facility Name	Can be null if borough of incident is not a Parks Dept facilty
Park Borough	Can be null if borough of incident is not a Parks Dept facilty
Vehicle Type	Can be null if the incident is not a taxi
Taxi Company Borough	Can be null if the incident is not a taxi
Taxi Pick Up Location	Can be null if the incident is not a taxi
Bridge Highway Name	Can be null if the incident is not identified as a Bridge/Highway
Bridge Highway Direction	Can be null if the incident is not identified as a Bridge/Highway
Road Ramp	Can be null if the incident is not identified as a Bridge/Highway
Bridge Highway Segment	Can be null if the incident is not identified as a Bridge/Highway
Latitude	Cannot be null
Longitude	Cannot be null
Location	Cannot be null

Table 1 Null values

Issues found when checking for null values written for sample data for now

- 201273 rows of data have missing values for Closed Date

3.1.2 Range Problem:

Some columns have a range of values that are considered valid for them. We check for this problem in the following columns:

Column	Range
Borough	“Manhattan”, “Brooklyn”, “Queens”, “Bronx”, “Staten Island”
Status	“Closed”, “Pending”, “Assigned”, “Open”, “In Progress”, “Started”, “Email Sent”, “Unassigned”, “Closed - Testing”, “Draft”, “Unspecified”

Table 2 Range values

3.1.3 Unexpected type problem:

We checked if some columns have a particular data type requirement. List of issues detected:

- Invalid datetime – We checked if each date column had a valid and real timestamp.

```
df_data.printSchema()
]

root
 |-- _c0: string (nullable = true)
 |-- Unique Key: string (nullable = true)
 |-- Created Date: string (nullable = true)
 |-- Closed Date: string (nullable = true)
```

Fig. 1 Unexpected type Problem

We converted the date columns from string to timestamp so comparisons and validation can be easily done.

The script we are used for that as follows:

```
df_data=df_data.withColumn('Created_Date',to_timestamp(df_data.Created_Date,"MM/dd/yyyy hh:mm:ss a"))
df_data=df_data.withColumn('Closed_Date',to_timestamp(df_data.Closed_Date,"MM/dd/yyyy hh:mm:ss a"))
df_data=df_data.withColumn('Due_Date',to_timestamp(df_data.Closed_Date,"MM/dd/yyyy hh:mm:ss a"))
```

3.1.4 Mapping Problem:

There may be dependencies between two columns. We found the following dependencies:

- The start time of the incident must be earlier than the end time of the incident i.e., Open Date must be earlier than Closed Date

3.2 Data cleaning

After analyzing and finding possible errors, we need to solve and remove these invalid data rows.

3.2.1 Improvise conflicting data:

3.2.1(a) Null Values identification:

We renamed the columns for easier access and if more null values are in column, then we dropped that row.

We used the following script for drop row for null values.

```
#Converting Null value data to a dictionary
null_dict_list = [row.asDict() for row in null_col_data]
null_dict = null_dict_list[0]
col_null_75p=list({i for i in null_dict if null_dict[i] > 0.75})
print(col_null_75p)
df_data = df_data.drop(*col_null_75p)
```

After that we also inspect the columns and row that must not be null or if column contain more Null values, then we dropped rows from the dataset.

3.2.1(b) Investigate duplicate values:

There are also duplicate data values in the dataset. We removed it by checking if the Unique key column has any duplicate data or not. If the column has instances of Duplicate data, We also removed duplicate data or repeated entries from the dataset using open clean and pyspark. For that we analyze and decided that there must be only one unique key in the dataset. we can remove the duplicate data rows usng the below script:

```
df1=df_data.groupBy("Unique_Key").count().filter("count > 1")
df1.drop('count').show()

df_data = df_data.dropDuplicates(['Unique_Key'])
```

3.2.1(c) Conflicts regarding missing values:

One important component in data cleansing is how we determine whether a few data ought to be taken into consideration invalid or now no longer while it has conflicts with different data. i.e., we found conflicts in Borough Name. If we found there are conflicts in borough name i.e., missing borough name, incident address, street name, cross street, intersection street, invalid borough name. Here we decide to use NYC zip code data and from this we can remove the conflicts of borough names, street name, cross street, intersection street.

For this process we used following script.

```
for i in range(len(specified_zip_data)):

  if (specified_zip_data.Borough[i] ==
zip_data.loc[zip_data['ZipCode'] ==
specified_zip_data.Incident_Zip[i], 'Borough'].iloc[0]):

    continue

  else:

    specified_zip_data.Borough[i] =
zip_data.loc[zip_data['ZipCode'] ==
specified_zip_data.Incident_Zip[i], 'Borough'].iloc[0]
```

	Unnamed: 0	Unique Key	Created_Date	Closed_Date	Agency	Agency_Name	Complaint_Type	Descriptor	Location_Type	Incident_Zip	Incident_Address
0	0	51560911	08/17/2021 07:34:45 AM	08/17/2021 09:34:29 AM	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11421	92-13 92 AVENUE
1	1	51560923	08/17/2021 06:04:48 PM	08/17/2021 07:25:36 PM	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	10468	CLAFLIN AVENUE
2	2	50469886	05/04/2021 11:40:00 AM	07/21/2021 01:55:00 PM	DOT	Department of Transportation	Street Light Condition	Street Light Out	NaN	11214	NaN
3	3	50470191	05/04/2021 04:31:32 PM	NaN	DPR	Department of Parks and Recreation	New Tree Request	For One Address	Street	11229	2041 EAST 18 STREET
4	4	50470218	05/05/2021 01:47:05 AM	10/01/2021 10:53:17 AM	DOHMH	Department of Health and Mental Hygiene	Rodent	Signs of Rodents	1-2 Family Dwelling	11420	115-44 135 STREET

Fig. 2 Missing Values

We applied date conversion technique using pyspark to our dataset, in this technique we are checking that if closed date is greater than the created date or not?

We used following script for date conversion.

```
df_new = df_data.filter(df_data.Closed_Date >
df_data.Created_Date)
df_new.count()

df_new2 = df_data.filter(df_data.Closed_Date.isNull())
df_new2.count()
```

3.2.1(d) Unspecified values:

We also applied one more technique to reduce the anomalies, we took community board dataset which has same zip codes column as our on NYC open data 311 Service requests from 2010 to present as. We cross verified the zip codes column with other community board data and if zip codes don't match range, then we gave "unspecified" tag to it.

```
def check_zip(z):

  if z in zip_data.ZipCode.unique():
      return int(z)

  else:
      return 'Unspecified'

pd_data.Incident_Zip = pd_data.Incident_Zip.map(lambda x:
check_zip(x))
```

3.2.1(e) Unique entities:

In addition, we also used zip to borough technique with fire incident in that we are converting zip codes in borough and also by checking the dataset we found that we same zip also contain unique boroughs.

We used following script for identify unique borough

```
for i in range(len(specified_zip_data)):
  if (specified_zip_data.BOROUGH[i] == zip_data.loc[zip_data['Z
ipCode'] == specified_zip_data.Postcode[i], 'Borough'].iloc[0]):
    continue
  else:
    specified_zip_data.BOROUGH[i] = zip_data.loc[zip_data['Zip
Code'] == specified_zip_data.Postcode[i], 'Borough'].iloc[0]
```

3.2.2 Data Standardization:

We try creating clusters of values which may have the same meaning and context but have different representations. For our data we use Key Collision Clustering on Street Name, Incident Address and Agency Name. Since address inputs may have inconsistencies due to manual input or varied representation of data received from different sources, our focus will be on address related attributes.

We created following script for the clustering.

```
def print_k_clusters(clusters, k=5):
    clusters = sorted(clusters, key=lambda x: len(x), reverse=True)
    val_count = sum([len(c) for c in clusters])
    print('Total number of clusters is {} with {} values'.format(len(clusters), val_count))
    for i in range(min(k, len(clusters))):
        print('\nCluster {}'.format(i + 1))
        for key, cnt in clusters[i].items():
            if key == '':
                key = """
            print(f'  {key} (x {cnt})')
```

```
Total number of clusters is 1629 with 3300 values

Cluster 1
  '' (x 1395710)
  * (x 7)
  . (x 4)
  / (x 1)

Cluster 2
  ST JOHNS AVENUE (x 143)
  ST JOHN'S AVENUE (x 10)
  AVENUE ST JOHNS (x 38)
  AVENUE ST JOHN'S (x 1)

Cluster 3
  CENTRAL PARK W (x 19)
  W CENTRAL PARK (x 4)
  W CENTRAL PARK PARK (x 4)
  CENTRAL PARK PARK W (x 1)

Cluster 4
  CENTRAL PARK WEST PARK W (x 1)
  CENTRAL WEST PARK W (x 4)
  CENTRAL PARK WEST. PARK W (x 1)
  W CENTRAL PARK WEST PARK W (x 5)

Cluster 5
  N/A (x 516)
  NA (x 139)
  N/A/ (x 1)
  NA/ (x 1)
```

Fig. 3 Clustering

After the clustering we got the data which belongs to same clusters.

4. RESULTS

From the data cleaning section we done analysis of borough name and missing borough name, street name, incident address, inter sections as a result of the script we get the missing borough name as well other entities.

	Borough	Neighborhood	ZipCode
0	Bronx	Central Bronx	10453
1	Bronx	Central Bronx	10457
2	Bronx	Central Bronx	10460
3	Bronx	Bronx Park and Fordham	10458
4	Bronx	Bronx Park and Fordham	10467

Fig. 4 Zip codes

Location_Type	Incident_Zip	Incident_Address	Street_Name	Cross_Street_1	Cross_Street_2	Intersection_Street_1	Intersection_Street_2	Address_Type	
Street/Sidewalk	11421	92-13 92 AVENUE	92 AVENUE	92 STREET	WOODHAVEN BOULEVARD	92 STREET	WOODHAVEN BOULEVARD	NaN	W
Street/Sidewalk	10468	CLAFLIN AVENUE	CLAFLIN AVENUE	CLAFLIN AVENUE	WEST 195 STREET	CLAFLIN AVENUE	WEST 195 STREET	NaN	
NaN	11214	NaN	NaN	NaN	NaN	NEW UTRECHT AVENUE	84 STREET	INTERSECTION	
Street	11229	2041 EAST 18 STREET	EAST 18 STREET	AVENUE T	AVENUE U	AVENUE T	AVENUE U	NaN	
1-2 Family Dwelling	11420	115-44 135 STREET	135 STREET	115 AVENUE	116 AVENUE	115 AVENUE	116 AVENUE	ADDRESS	S

Fig. 4.1 Zip Codes

Then we worked out with Null values also and by getting percentage of null values in each column we decided that if columns contains more than 75% null values then we remove it.

Drop Columns containing more than 75% null values.

```
ll_col_data = df_data.select([(count(when(col(c).isNull(), c))/df_data.count()).alias(c) for c in df_data.columns]).collect()
ll_col_data
```

```
[Row(_c0=0.0, Unique_Key=0.0, Created_Date=0.0, Closed_Date=0.04949, Agency=0.0, Agency_Name=0.0, Complaint_Type=0.0, Descri
ptor=0.0032593333333333333, Location_Type=0.305324, Incident_Zip=0.3192426666666667, Incident_Address=0.3448526666666664, S
treet_Name=0.3449113333333335, Cross_Street_1=0.5335066666666666, Cross_Street_2=0.54303, Intersection_Street_1=0.620698666
6666666, Intersection_Street_2=0.6230033333333334, Address_Type=0.52, City=0.33925133333333333, Landmark=0.762316, Facility_T
ype=0.44323, Status=0.0, Due_Date=0.682708, Resolution_Description=0.024336, Resolution_Action_Updated_Date=0.050738666666666
667, Community_Board=0.012646, BBL=0.5459406666666666, Borough=0.012646, X Coordinate (State Plane)=0.4785166666666665, Y C
oordinate (State Plane)=0.47848066666666667, Open Data Channel Type=1.2666666666666667e-05, Park Facility Name=1.26666666666
66667e-05, Park_Borough=0.012646, Vehicle Type=0.9994933333333333, Taxi Company Borough=0.9989966666666666, Taxi Pick Up Loc
ation=0.9869713333333333, Bridge Highway Name=0.9947166666666667, Bridge Highway Direction=0.9948233333333333, Road Ramp=0.9
948666666666667, Bridge Highway Segment=0.9946873333333334, Latitude=0.4785246666666665, Longitude=0.4785246666666665, Loc
ation=0.4785126666666664)]
```

The above data shows the percentage of Null values present in each column of our dataset. Percentage is displayed within range 0-1.

Fig. 5 Null values

Dropped column after finding the results.

Finding Columns containing more than 75% null values.

```
col_null_75p=list([i for i in null_dict if null_dict[i] > 0.75])
print(col_null_75p)
```

```
['Bridge Highway Segment', 'Road Ramp', 'Landmark', 'Vehicle Type', 'Bridge Highway Name', 'Taxi Pick Up Location', 'Taxi Co
mpany Borough', 'Bridge Highway Direction']
```

```
df_data = df_data.drop(*col_null_75p)
```

Fig. 6 eliminate null values

In duplication values in dataset, we used unique key concept and if dataset had one more unique then we made it only one unique key through the dataset.

Removing Duplicate value rows for Unique Key Column

We first check if Unique Key column has any Duplicate Data

```
df1=df_data.groupBy("Unique_Key").count().filter("count > 1")
df1.drop('count').show()
```

```
+----------+
|Unique_Key|
+----------+
+----------+
```

Fig. 7 Unique Key

After that we also used community board dataset to verify the zip codes and if they don't match we tagged it "unspecified".

```
pd_data.Community_Board.head(10)
```

```
1]: 0         9 QUEENS
    1          8 BRONX
    2     11 BROOKLYN
    3     15 BROOKLYN
    4       10 QUEENS
    5       10 QUEENS
    6        11 BRONX
    7         4 BRONX
    8      6 BROOKLYN
    9     Unspecified
    Name: Community_Board, dtype: object
```

Fig. 8 Unspecified data

Then after all we used clustering to make clusters of the data and divided into the same belongings' groups.

```
Total number of clusters is 2 with 4 values

Cluster 1
  SCHOOL - ARCHIMEDES ACADEMY FOR MATH, SCIENCE AND TECHNOLOGY APPLICATIONS (x 1)
  SCHOOL - ARCHIMEDES ACADEMY FOR MATH, SCIENCE, AND TECHNOLOGY APPLICATIONS (x 3)

Cluster 2
  SCHOOL - SCHOLARS ACADEMY (x 2)
  SCHOOL - SCHOLARS' ACADEMY (x 1)
```

Fig. 9 Clustering

```
Total number of clusters is 43204 with 90546 values

Cluster 1
  N/A N/A (x 486)
  NA NA (x 133)
  N/A (x 17)
  N/A NA (x 4)
  NA (x 3)
  N/A NA/ (x 1)
  NA N/A (x 1)

Cluster 2
  12 WEST 12 STREET (x 9)
  WEST   12 STREET (x 38)
  12 WEST STREET (x 8)
  WEST 12 STREET (x 74)
  12 WEST   12 STREET (x 9)
  12 WEST WEST 12 STREET (x 1)
```

Fig. 9.1 Clustering

```
Cluster 3
  140 WEST STREET (x 30)
  140 WEST  140 STREET (x 10)
  WEST 140 STREET (x 57)
  WEST  140 STREET (x 7)
  140 WEST 140 STREET (x 30)
  140 WEST WEST 140 STREET (x 2)

Cluster 4
  147 WEST  230 STREET (x 13)
  230 WEST  147 STREET (x 31)
  230 WEST 147 STREET (x 47)
  147 WEST 230 STREET (x 18)
  230 WEST WEST 147 STREET (x 1)
  230 WEST WEST  147 STREET (x 1)

Cluster 5
  WEST   16 STREET (x 15)
  WEST 16 STREET (x 76)
  16 WEST   16 STREET (x 9)
  16 WEST 16 STREET (x 18)
  16 WEST STREET (x 1)
  16 WEST WEST   16 STREET (x 1)
```

Fig. 9.2 Clustering

In date conversion we get following results after executing the script.

```
df_new = df_data.filter(df_data.Closed_Date > df_data.Created_Date)

df_new.count()

]: 1363073

df_new2 = df_data.filter(df_data.Closed_Date.isNull())

df_new2.count()

]: 74235
```

Fig 9.3 Date Conversation

In unique borough we get following result.

```
zip_data.BOROUGH.unique()

array(['BRONX', 'BROOKLYN', 'MANHATTAN', 'QUEENS', 'STATEN ISLAND'],
      dtype=object)

for i in range(len(specified_zip_data)):
  if (specified_zip_data.BOROUGH[i] == zip_data.loc[zip_data['ZipCode'] == specified_zip_data.Postcode[i], 'Borough'].iloc[0]):
    continue
  else:
    specified_zip_data.BOROUGH[i] = zip_data.loc[zip_data['ZipCode'] == specified_zip_data.Postcode[i], 'Borough'].iloc[0]

specified_zip_data.BOROUGH.unique()

array(['BRONX', 'QUEENS', 'STATEN ISLAND'], dtype=object)
```

Fig 9.4 Unique Borough

5. ACKNOWLEDGMENTS

we would like to express my gratitude to our Professor Juliana Freire for the useful remarks and engagement through the learning process of this final project. Also, I like to thank the contributors in NYC open data 311 Service requests from 2010- present dataset. we would like to thank my loved ones, who have supported me throughout entire process, both by keeping us harmonious and helping me putting pieces together. we will be grateful forever for your love.

6. REFERENCES

[1] Ref: https://communityprofiles.planning.nyc.gov/

[2] Red : https://www.splitgraph.com/cityofnewyork-us/fire-incident-dispatch-data-8m42-w767

[3] Ref: https://www.splitgraph.com/cityofnewyork-us/incidents-responded-to-by-fire-companies-tm6d-hbzd/latest/-/tables

[4] Ref: https://data.cityofnewyork.us/Public-Safety/NYPD-Shooting-Incident-Data-Year-To-Date-/5ucz-vwe8

[5] Ref: https://data.cityofnewyork.us/widgets/yhuu-4pt3