Project: Analyzing NYPD Complaint Data (Historic)

Team Member	NetId
Niranjhana Nayar	nn1024
Priyanka Bhadoriya	pb1826
Suchetha Siddagangappa	ss11436

Contents

Abstract

The goal of this project is to analyze the data in a dataset, clean it and generate a descriptive summary of the data quality issues. Also we will process the cleaned data (from the first part) and generate a descriptive summary for the features and contents. The dataset used (NYPD Complaint Data Historic) includes all valid felony, misdemeanor, and violation crimes reported to the New York City Police Department (NYPD) from 2006 to the end of last year (2016).

Introduction

In the data cleaning part, we have looked for data quality issues as well as anomalies such as the ones mentioned below.

- Are there non-empty values that represent missing data (e.g., NULL, N/A, UNSPECIFIED, TBA, (999)999-9999) If so, how many in each column?
- Are there different kinds of values in the same column (e.g., integers and strings)?
- Are there suspicious or invalid values in columns? (e.g., a negative value in a price field; for spatial data, coordinates outside the city perimeter; an invalid zipcode)

Part I : Data Cleaning

Column Details

Color Codes:- Red(ignored), Green(completely valid) and Yellow(validated)

#	Column Name	NULLS	INVALIDS	VALIDS	TYPE	CHECK FOR INVALID
0	CMPLNT_NUM	0	0	5580035	String(num)	check_col1()
1	CMPLNT_FR_DT	655	18972	5560408	Date	check_date()
2	CMPLNT_FR_TM	48	903	5579084	Time	check_time()
3	CMPLNT_TO_DT	1472786	4938	4102311	Date	check_date()
4	CMPLNT_TO_TM	1468882	1376	4109777	Time	check_time()
5	RPT_DT	0	0	5580035	Date	check_date()
6	KY_CD	0	0	5580035	String(num)	check_id()
7	OFNS_DESC	18892	0	5561143	String	check_id()
8	PD_CD	4909	0	5575126	String	check_desc()
9	PD_DESC	4909	0	5575126	String	check_desc()
10	CRM_ATPT_CPTD_CD	7	0	5580028	String	check_action()
11	LAW_CAT_CD	0	0	5580035	String	check_lawcd()
12	JURIS_DESC	0	0	5580035	String	check_desc()
13	BORO_NM	463	0	5579572	String	check_boro()
	QUEENS	0	0	1105621		
	BRONX	0	0	1666903		
	BROOKLYN	0	0	1331760		
	MANHATTAN	463	0	1331760		
	STATEN ISLAND	0	0	265641		
14	ADDR_PCT_CD	390	0	5579645	String(num)	check_addrpctcd()
15	LOC_OF_OCCUR_DESC	1223392	213	4356430	String	check_loc()
16	PREM_TYP_DESC	35198	0	5544837	String	check_desc()
17	PARKS_NM	5567497	0	12538	String	check_desc()
18	HADEVELOPT	5302218	0	277817	String	check_desc()
19	X_COORD_CD	0	195868	5384167	Coordinates	check_xloc()
20	Y_COORD_CD	0	195868	5384167	Coordinates	check_yloc()

21	Latitude	0	195868	5384167	Coordinates	check_lat()
22	Longitude	0	195868	5384167	Coordinates	check_lon()
23	LatLon	0	195868	5384167	Coordinates	lat_lon()

check_date()	Check if the year is between 2006-2016
check_time()	Check whether the time is in 24 hour format
check_id()	Check whether id is 3 character long
check_desc()	check if the field is a string
check_action()	Valid values are COMPLETED, ATTEMPTED
check_boro()	check if the field is a valid New York borough. Valid boroughs are : MANHATTAN, BROOKLYN, BRONX, QUEENS, STATEN ISLAND
check_addrpctcd()	The precinct in which the incident occurred 1 to 123
check_xloc()	valid x coord range : from 909900 to 1067600
check_yloc()	valid y coord range : from 117500 to 275000
check_lat()	valid latitude range : between -73 and -75
check_lon()	valid longitude range : between 40 and 41

Descriptive summary of the data quality issues :

- Total no. of columns - 24
- 5 columns contain no invalid or null values.
- 14 columns contain NULL values. The highest NULL values are present in column PARKS_NM and the least is present in CRM_ATPT_CPTD_CD.
- 10 columns contain invalid values. Highest no. of invalid values are found in columns x_coord_cd, y_coord_cd, latitude, longitude, lat_lon

Strategies for cleaning

- From the above data it can be seen that CMPLNT_NUM,RPT_DT,KY_CD,LAW_CAT_CD and JURIS_DESC is always valid and can be considered as the most important features.

- CMPLNT_FR_DT should never be null. However CMPLNT_TO_DT can be null in case CMPLNT_FR_DT is present for that record.
- Similar logic applies for CMPLNT_FR_TM and CMPLNT_TO_TM.
- Similarly LOC_OF_OCCUR_DESC, PREM_TYP_DESC, PARKS_NM and HADEVELOPT need not be present for all the fields so cannot be used to eliminate records.
- X_COORD_CD,Y_COORD_CD,Latitude and Longitude conveys the same data as LatLon. Hence, LatLon is used for cleaning the data.
- **CMPLNT_TO_DT, CMPLNT_TO_TM, OFNS_DESC, PD_CD, PD_DESC, CRM_ATPT_CPTD_CD, BORO_NM, ADDR_PCT_CD, LatLon** are used for filtering VALID data.
- There maybe null values in the CMPLNT_TO_DT, CMPLNT_TO_TM, LOC_OF_OCCUR_DESC, PREM_TYP_DESC, PARKS_NM and HADEVELOPT, X_COORD_CD,Y_COORD_CD,Latitude and Longitude columns.
- Appropriate replacement of these null values are done.(Code)
    - For dates **01/01/1900** was given
    - For time **00:00:00** was given
    - For strings **NA** was given
    - The coordinate fields were never null so it was not replaced with any value
- The final csv is then generated. (Code)

GitHub path : https://github.com/pb1826/Big-Data-Project

Code used to recognise NULL values :

```
def check_null(x):
        try:
                if x=='':
                        return True;
        except ValueError(x):
                return False;
```

Code that reports statistical information : (No. of invalid values)

check_col1()

```
def check_col1(x):
    try:
        if(check_null(x)):
            return "NULL";
        elif re.match('[1-9][0-9]+',x):
            return "VALID", check_type(x);
        else:
            return "INVALID";
    except ValueError:
        return "INVALID";
```

check_type()

```
def check_type(x):
        try:
                switcher={
                        int: "INT",
                        float: "FLOAT",
                        long: "LONG",
                        str: "STRING",
                }
                return switcher.get(type(x),"NULL")
        except ValueError:
                return "INVALID";
```

check_date()

```
def check_date(x):
        # check if the field is null or if it is between the years 2006-2016
        try:
                if(check_null(x)):
                        return "NULL";

                elif(re.match('(0?[1-9]|1[0-2])/(3[01]|[12][0-9]|0?[1-9])/(20)(0[6-9]|1[0-6])$',x)):
                        return "VALID","DATE";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_time()

```
def check_time(x):
        #check if the field is null or if it is in 24 hr format
        try:
                if(check_null(x)):
                        return "NULL";
                elif(re.match('(?:(?:([01]?\d|2[0-3]):)?([0-5]?\d):)?([0-5]?\d)$',x)):
                        return "VALID","TIME";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_desc()

```
def check_desc(x):
        #check if the field is null or is a string
        try:
                if(check_null(x)):
                        return "NULL";
                elif(type(x)==str):
                        return "VALID","STRING";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_boro()

```
def check_boro(x):
        #check if the field is null or in the list of new york boroughs
        try:
                if(check_null(x)):
                        return "NULL";
                elif(x in ['MANHATTAN', 'BRONX', 'BROOKLYN', 'QUEENS', 'STATEN ISLAND']):
                        return "VALID","STRING";
                else:
                        return "INVALID";

        except ValueError:
                return "INVALID";
```

check_action()

```
def check_action(x):
        try:
                if(check_null(x)):
                        return "NULL";
                elif(x in ['COMPLETED','ATTEMPTED']):
                        return "VALID","STRING";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_xloc()

```
def check_xloc(x):
        #check if item is null or if it is a coordinate
        #X-coordinate (East-West): minimum: 909900; maximum: 1067600
        try:
                x=int(x);
                if(check_null(x)):
                        return "NULL";
                elif((909900<x) & (x<1067600)):
                        return "VALID","COORDINATES";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_yloc()

```
def check_yloc(x):
        #check if the item is null or if it is a coordinate
        #for nyc Y-coordinate (North-South): minimum: 117500; maximum: 275000
        try:
                x=int(x);
                if(check_null(x)):
                        return "NULL";
                elif((117500<x) & (x<275000)):
                        return "VALID","COORDINATES";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_lon()

```
def check_lon(x):
        #checking if the field is null or if it is a lon between -73 and -75
        try:
                x=float(x);
                if(check_null(x)):
                        return "NULL";
                elif((-75<x) & (x<-73)):
                        return "VALID","COORDINATE";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_lat()

```
def check_lat(x):
        #checking if the field is null or if it is a lat in between 40 and 41
        try:
                x=float(x);
                if(check_null(x)):
                        return "NULL";
                elif((40<x) & (x<41)):
                        return "VALID","COORDINATE";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_loc()

```
def check_loc(x):
        try:
                occur = ['FRONT OF', 'INSIDE', 'REAR OF', 'OUTSIDE', 'OPPOSITE OF']
                if(check_null(x)):
                        return "NULL";
                elif x in occur:
                        return "VALID", "STRING";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_lawcd()

```
def check_lawcd(x):
        try:
                law = ['MISDEMEANOR', 'VIOLATION', 'FELONY'];
                if(check_null(x)):
                        return "NULL";
                elif x in law:
                        return "VALID","STRING";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_addrpctcd()

```
def check_addrpctcd(x):
        try:
                if (check_null(x)):
                        return "NULL";
                elif (int(x)>=1 and int(x)<=123):
                        return "VALID","INT";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

lat_lon()

```
def lat_lon(x):
        try:
                x=x.replace('(','').replace(')','');
                lat,lon=x.split(',');
                a=check_lat(lat);
                b=check_lon(lon);
                if(a[0]=="VALID" and b[0]=="VALID"):
                        return "VALID","STRING";
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

check_id()

```
def check_id(x):
        #check if the field is null or 3 digits long
        try:
                if(check_null(x)):
                        return "NULL";

                elif(re.match('[1-9][0-9][0-9]$',x)):
                        return "VALID",check_type(x);
                else:
                        return "INVALID";
        except ValueError:
                return "INVALID";
```

Data Cleaning

```
cleaned=output.filter(lambda x: x[2][0]=='VALID' and \
x[4][0]=='VALID' and \
x[10][0]=='VALID' and \
x[12][0]=='VALID' and \
x[14][0]=='VALID' and \
x[16][0]=='VALID' and \
x[20][0]=='VALID' and \
x[22][0]=='VALID' and \
x[32][0]=='VALID').map(lambda x: (x[0],\
x[1],x[3],x[5],x[6],x[7],x[8],x[9],x[11],x[13],x[15],x[17],x[18],x[19],x[21],x[23],\
x[24],x[25],x[26],x[27],x[28],x[29],x[30],x[31]));

file=cleaned.map(toCSV);
file.saveAsTextFile("cleaned.csv");
```

Nulls

```
# the coordinate features don't have any null values so it is not been considered
#cmplt_to_dt
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],\
(x[5].replace('',datetime.strptime('01/01/1900', "%m/%d/%Y").strftime('%m/%d/%Y'))),\
x[6],x[7],x[8],x[9],x[10],x[11],x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],\
x[21],x[22],x[23],x[24],x[25],x[26],x[27],x[28],x[29],x[30],x[31],x[32]) if check_null(x[5])==True else (x));
#cmplt_to_tm
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],x[5],\
(x[6].replace('',datetime.strptime('00:00:00', "%H:%M:%S").strftime('%H:%M:%S'))),\
x[7],x[8],x[9],x[10],x[11],x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],\
x[21],x[22],x[23],x[24],x[25],x[26],x[27],x[28],x[29],x[30],x[31],x[32]) if check_null(x[6])==True else (x));
#LOC_OF_OCCUR_DESC
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x[11],\
x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],x[21],x[22],\
(x[23].replace('','NA')),\
x[24],x[25],x[26],x[27],x[28],x[29],x[30],x[31],x[32]) if check_null(x[23])==True else (x));
#PREM_TYP_DESC
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x[11],\
x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],x[21],x[22],x[23],\
(x[24].replace('','NA')),\
x[25],x[26],x[27],x[28],x[29],x[30],x[31],x[32]) if check_null(x[24])==True else (x));
#PARKS_NM
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x[11],\
x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],x[21],x[22],x[23],x[24],\
(x[25].replace('','NA')),\
x[26],x[27],x[28],x[29],x[30],x[31],x[32])if check_null(x[25])==True else (x));
#HADEVELOPT
cleaned=cleaned.map(lambda x : (x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x[11],\
x[12],x[13],x[14],x[15],x[16],x[17],x[18],x[19],x[20],x[21],x[22],x[23],x[24],x[25],\
(x[26].replace('','NA')),\
x[27],x[28],x[29],x[30],x[31],x[32]) if check_null(x[26])==True else (x));
```