CIS 9: Final Project

Data Analysis of San Jose Police Department Incident Reports (2018-2022): Part 1

Written by: Tiffany Overbo and Cherry Withers

Project Summary

This notebook is one of two notebooks for this project. There was a necessity to divide the notebooks into two because of our very large dataset (~4M rows). The dataset made for very slow progress. This notebook cleans the data and will be used for our other notebook on data analysis.

This project is to analyze incidence reports from police calls for services which are documented by the San Jose City Police Department. The goal is to analyze the trend in types of incidents, address area, time of day, frequency, and if there were any resolutions over 5 years from 2018 - 2022. This will include pre-pandemic, pandemic, and post-pandemic years to see if there is a trend and hopes to answer the following questions:

- What are the 10 top incidents being reported each year by frequency?
- How did the pandemic affect the incident counts of the following crimes/categories: Assault, Burglary, Disturbing Peace, Drugs/Alcohol, DUI, Fraud, Motor Vehicle Theft, Robbery, Sex Crime, Larceny, Vandalism, Vehicle Breakin/Theft, Others?
- Arrest rates for each category mentioned above (per year/per month)?
- Where are most of these incidents occurring (street names)?
- What part of the day are the incidents taking place, ie. am, pm, after midnight?
- Which months have higher incidents of crimes being reported year after year?
- Trend analysis of each major category from 2018-2022.

*Note: 2023 data has only 01-01-23 thru 03-18-23 and is an incomplete year. Therefore, we will not include that in our analysis.

Data Information

We will be loading 5 .csv files, each containing the total incidents for each year from 2018-2022. [Source:] (https://data.sanjoseca.gov/dataset/police-calls-for-service)

This header name and description are the following:

1. CDTS: Unique ID
2. EID Unique ID

3. START_DATE Start Date
4. CALL_NUMBER Unique Call ID (potentially different callers or different time calling back for the same instance)
5. PRIORITY Priority of the incident
6. REPORT_DATE Date of when the incident is reported
7. OFFENSE_DATE Date of when the offense happened (same date as Report Date)
8. OFFENSE_TIME Time of when the offense happened
9. CALLTYPE_CODE Numeric equivalence of Call Type
10. CALL_TYPE Type of incident (186 unique types queried, possibly more)
11. FINAL_DISPO_CODE Final Disposition Code
12. FINAL_DISPO Final Disposition
13. COMMON_PLACE_NAME Common Place Name
14. ADDRESS Address
15. CITY City
16. STATE State

Data Cleaning

1. Importing Modules and Initial Look of Pre-processed Data

In [1]:
```
# Import modules here:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

In [2]:
```
# Read in all our files and assemble them into one dataframe
start = time.time()

files = ["FinalProject/policecalls2018.csv",
         "FinalProject/policecalls2019.csv",
         "FinalProject/policecalls2020.csv",
         "FinalProject/policecalls2021.csv",
         "FinalProject/policecalls2022.csv"]

df_og = pd.DataFrame()

# extract the year names form of each file name and print row/column count
for file in files:
    df = pd.read_csv(file)
    print(f"{file[-8:-4]}: Row count: {df.shape[0]} and Column count: {df.shape[1]}
    df_og = pd.concat([df_og, df], ignore_index=True)

print("All Years: Row count:", df_og.shape[0], "and Column count:", df_og.shape[1])

end = time.time()
print("Time:", end-start)
# Time = ~14-20 seconds on my desktop. Time will vary on different machine and othe
```

```
2018: Row count: 322371 and Column count: 16
2019: Row count: 322628 and Column count: 16
2020: Row count: 297463 and Column count: 16
2021: Row count: 1375897 and Column count: 16
2022: Row count: 1945529 and Column count: 16
All Years: Row count: 4263888 and Column count: 16
Time: 14.938721418380737
```

In [3]:
```
# Looking at header column names and datatype:
df_og.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4263888 entries, 0 to 4263887
Data columns (total 16 columns):
 #   Column             Dtype 
---  ------             ----- 
 0   CDTS               object
 1   EID                int64 
 2   START_DATE         object
 3   CALL_NUMBER        object
 4   PRIORITY           int64 
 5   REPORT_DATE        object
 6   OFFENSE_DATE       object
 7   OFFENSE_TIME       object
 8   CALLTYPE_CODE      object
 9   CALL_TYPE          object
 10  FINAL_DISPO_CODE   object
 11  FINAL_DISPO        object
 12  COMMON_PLACE_NAME  object
 13  ADDRESS            object
 14  CITY               object
 15  STATE              object
dtypes: int64(2), object(14)
memory usage: 520.5+ MB
```

In [4]:
```
# Sample of the data:
df_og.head(5)
```

Out[4]:

	CDTS	EID	START_DATE	CALL_NUMBER	PRIORITY	REPORT_DATE	OF
0	20180101000426PS	7000038	14-MAY-21	P180010001	2	2018-01-01	
1	20180101000120PS	7000040	14-MAY-21	P180010003	3	2018-01-01	
2	20180101003329PS	7000041	14-MAY-21	P180010004	4	2018-01-01	
3	20180101000358PS	7000043	14-MAY-21	P180010005	4	2018-01-01	
4	20180101000535PS	7000044	14-MAY-21	P180010006	4	2018-01-01	

2. Finding Duplicates and NaNs and dropping them

Review each column to see if there are duplicates, have any NaNs and understand if they add value.

2.1. Finding which columns have duplicates or add values

Use unique and counts to understand the dataset.

In [5]:
```
# Find the frequency of each column to assess if there are duplicates:
header_name = df_og.columns
for header in header_name:
    unique_count = len(df_og[header].value_counts())
    print(header, "unique count is", unique_count)
```

```
CDTS unique count is 1529044
EID unique count is 1505300
START_DATE unique count is 560
CALL_NUMBER unique count is 1507152
PRIORITY unique count is 6
REPORT_DATE unique count is 1820
OFFENSE_DATE unique count is 1820
OFFENSE_TIME unique count is 86381
CALLTYPE_CODE unique count is 223
CALL_TYPE unique count is 237
FINAL_DISPO_CODE unique count is 22
FINAL_DISPO unique count is 25
COMMON_PLACE_NAME unique count is 6239
ADDRESS unique count is 67752
CITY unique count is 1
STATE unique count is 1
```

Since CDTS, EID, and CALL_NUMBER have high unique count and low duplicates, more investigation is needed. We emailed OpenData at opendata@sanjoseca.gov to help understand what these columns are. Arti Tangri, the Equity Data Lead for the City of San Jose wrote back explaining that the CDTS, EID and Call_Number are all

system IDs that are generated from their internal systems. After reviewing these columns, we decide to use EID as the unique identifiers for a particular instance or call.

2.2. Dropping Duplicate Rows

The EID column is comprised of the unique incident ID. However, multiple rows of the same EID occurs for several reasons. One being that multiple people have called about the same incident, or updates about the incident are logged with the same EID. Before we drop EID, we'd like to grab the last instance of (based on the last reported date, which is the START_DATE) so we know if there was a resolution or not with the incident.

In [6]:
```
# Dropping duplicates of EID but grabbing the latest STATE_DATE:
# Sorting for START_DATE descending order:
df_og = df_og.sort_values(by=["EID", "START_DATE"], ascending=[True, False])
# Only keeping the first row of the descending sort:
df = df_og.drop_duplicates(subset="EID", keep="first")

# Checking df size post removing duplicates:
print("Total data length: ",len(df))
```

```
Total data length:  1505300
```

In [7]:
```
# Check for column counts after removing duplicates from EID + START_DATE:
header_name = df.columns
for header in header_name:
    print(header, "unique count is", str(len(df[header].value_counts()))+".")
```

```
CDTS unique count is 1497431.
EID unique count is 1505300.
START_DATE unique count is 424.
CALL_NUMBER unique count is 1505300.
PRIORITY unique count is 6.
REPORT_DATE unique count is 1820.
OFFENSE_DATE unique count is 1820.
OFFENSE_TIME unique count is 86381.
CALLTYPE_CODE unique count is 223.
CALL_TYPE unique count is 237.
FINAL_DISPO_CODE unique count is 22.
FINAL_DISPO unique count is 25.
COMMON_PLACE_NAME unique count is 6239.
ADDRESS unique count is 67752.
CITY unique count is 1.
STATE unique count is 1.
```

In [8]:
```
# Check for NaNs:
df.isna().sum()
```

Out[8]:
```
CDTS                       0
EID                        0
START_DATE                 0
CALL_NUMBER                0
PRIORITY                   0
REPORT_DATE                0
OFFENSE_DATE               0
OFFENSE_TIME               0
CALLTYPE_CODE              0
CALL_TYPE                 29
FINAL_DISPO_CODE           0
FINAL_DISPO                0
COMMON_PLACE_NAME    1197782
ADDRESS                44695
CITY                       0
STATE                      0
dtype: int64
```

2.3. Dropping Other Columns

We will be dropping these columns for the following reasons:

- CALLTYPE_CODE because it is a duplicate of CALL_TYPE.
- CALL_NUMBER because it is a phone call id number and it's no value to the analysis
- CDTS because it's an internal system id and it has no value to the analysis
- CITY because there is one value San Jose
- COMMON_PLACE_NAME because there are 1.2M NaNs which is 79% of the dataset. We will use the ADDRESS column for location of incident
- EID because it is a unique id and has no value to the analysis
- FINAL_DISPO_CODE because it is a variation of FINAL_DISPO. We will use FINAL_DISPO.
- REPORT_DATE because it is the same as OFFENSE_DATE
- STATE because there is one value CA
- START_DATE because it is the date of a call and instances of the calls. We will be using OFFENSE_DATE as the incident date.

In [9]:
```
# Dropping unnecessary columns:
drop_list = ['CALLTYPE_CODE','CALL_NUMBER','CDTS','CITY','COMMON_PLACE_NAME','EID',
             'FINAL_DISPO_CODE','REPORT_DATE','STATE','START_DATE']
for i in drop_list:
    df = df.drop(columns=[i])
```

In [10]:
```
# Chekcing the remaining column names:
print(df.columns.values)
df.shape
```

```
['PRIORITY' 'OFFENSE_DATE' 'OFFENSE_TIME' 'CALL_TYPE' 'FINAL_DISPO'
 'ADDRESS']
```

Out[10]:
```
(1505300, 6)
```

3. Deletion of NAs and Changing Column names to lowercase

In [11]:
```
# Delete rows that are NaNs:
```

```
df = df.dropna()
# Check if there are any more NaNs left:
df.isna().sum()
```

Out[11]:
```
PRIORITY        0
OFFENSE_DATE    0
OFFENSE_TIME    0
CALL_TYPE       0
FINAL_DISPO     0
ADDRESS         0
dtype: int64
```

In [12]:
```
# Changing each headers to lowercase:
df.columns = df.columns.str.lower()
# First look of processed-cleaner dataframe
df.columns
```

Out[12]:
```
Index(['priority', 'offense_date', 'offense_time', 'call_type', 'final_dispo',
       'address'],
      dtype='object')
```

Addition of Other Columns for Data Analysis

1. Separating offense_date column to columns month, day, year

We decide to separate offense_date into 3 periods: year, month, and month_year. This is to help with year over year, month over month, or consecutive month year analysis.

In [13]:
```
# Separate offense_date to month, day, year
# Setting up a year column for yearly analysis:
df['year'] = pd.DatetimeIndex(df['offense_date']).year
# Setting up a monthly column for month analysis:
df['month'] = pd.DatetimeIndex(df['offense_date']).month
# Setting up month_year column for month and year analysis:
df['month_year'] = pd.to_datetime(df['offense_date']).dt.to_period('M')
```

2. Adding a time_group column

We were going to map hours of the day that incident occurs into 4 different categories based on when it happened: early morning (midnight to 6 AM), morning (6 AM to 12 PM), afternoon (12 PM to 6 PM), and evening (6 PM to midnight). We would like to know which part of the day most of these incidents occur. However, after reviewing the line graphs with the 4 different categories, the line graphs were very rigid and spiked up and down steeply for every 6 hour increments. However, using the offense_time, the line graph showed a smoother line with smaller periods incremented being plotted. Therefore, we have not included our codes of the 4 different categories. We will only analyze time of day with the offense_time which better reflects the actual time of the incident.

3. Grouping of call_type column into a new type column

There are 237 call_types and some of them are duplicates with spaces, partial spelling, or similar types of incidents that can be grouped under a broader category. Therefore we created a new column named type to group alike call_types. We decide to keep call_type

becuase we may do more analysis of the specific call_type. For instance, we may want to analyze what specific types are under "suspicious activities". There are suspicious circumstances, suspicious female, suspicious package, suspicious person, prowler, and vagrant. These details may help explain future analysis so therefore we decide to keep this in our dataset.

In [14]:
```
# List the call_type and it's count:
df.groupby("call_type").final_dispo.count()
```

Out[14]:
```
call_type
1091AB          - VICIOUS ANIM                          13
1091AB          - VICIOUS ANIMAL (COMBINED EVENT)      248
ABANDONED VEHICLE                                     6975
ALARM                                                 3030
ALARM, AUDIBLE                                      103445
                                                       ...
W&I UNCONTROLLABLE JUVENILE                           32
W&I-UNDER JURIS OF JUV COURT                          41
WELFARE CHECK                                       103984
WELFARE CHECK (COMBINED EVENT)                       13372
X-RAY MACHINE ALARM/AIRPORT                              1
Name: final_dispo, Length: 237, dtype: int64
```

In [15]:
```
# Adding a new column named type column that will group alike call_type:
df['type'] = df['call_type']
display(df.shape)
df.head(3)
```

(1460576, 10)

Out[15]:

	priority	offense_date	offense_time	call_type	final_dispo	address	year
0	2	2018-01-01	00:00:02	FIRE DEPARTMENT REQUEST FOR PD	No report required; dispatch record only	[2900]-[3000] ALUM ROCK AV	2018
1	3	2018-01-01	00:00:15	ALARM, AUDIBLE	No Response	[4700]-[4800] CALENDULA CT	2018
2	4	2018-01-01	00:00:32	DISTURBANCE, MUSIC	Canceled	[3100]-[3200] WILLIAMSBURG DR	2018

In [17]:
```
start = time.time()
# Using regex to replace the old call_type name in the new column type:
df.type = df.type.str.replace(r'.*WEAPON.*|FIREARM.*|SHOOTING.*|.*SHOT.*|.*BOMB.*',
df.type = df.type.str.replace(r'.*DUI.*', 'DUI', regex=True)
df.type = df.type.str.replace(r'CARSTOP.*|UNLICENSED DRIVER|.*DRIVING.*|EXPIRED REG
df.type = df.type.str.replace(r'RECOVERED STOLEN VEHICLE', 'Recovered Stolen Auto',
df.type = df.type.str.replace(r'.*VEHICLE.*|CARJACKING.*|ALTERED VIN NUMBER', 'Vehi
df.type = df.type.str.replace(r'.*ALARM.*', 'Alarm', regex=True)
df.type = df.type.str.replace(r'.*ANIM.*|.*HORSE.*', 'Animal', regex=True)
df.type = df.type.str.replace(r'.*ASSAULT.*|.*HARASS.*', 'Assault', regex=True)
df.type = df.type.str.replace(r'.*DISTURBANCE.*', 'Disturbance', regex=True)
```

```
df.type = df.type.str.replace(r'.*ROBBERY.*', 'Robbery', regex=True)
df.type = df.type.str.replace(r'.*THEFT.*', 'Theft', regex=True)
df.type = df.type.str.replace(r'.*WELFARE.*', 'Welfare', regex=True)
df.type = df.type.str.replace(r'.*NARCOTICS.*|.*CONTROLLED SUB.*|.*MARIJUANA.*|.*AL
df.type = df.type.str.replace(r'.*SUSPICIOUS.*|.*PROWLER*|.*VAGRANT*', 'Suspicious
df.type = df.type.str.replace(r'.*COMMUNI.*POLICING.*', 'Community Policing', regex
df.type = df.type.str.replace(r'.*MISSING.*', 'Missing', regex=True)
df.type = df.type.str.replace(r'.*PEDESTRIAN.*', 'Pedestrian', regex=True)
df.type = df.type.str.replace(r'.*BATTERY.*', 'Battery', regex=True)
df.type = df.type.str.replace(r'.*BAD CHECKS.*|.*BAR CHECK.*|.*DEFRAUD.*|.*EMBEZZLE
df.type = df.type.str.replace(r'.*FIRE .*|.*EXPLOSION.*|.*ARSON.*', 'Fire', regex=T
df.type = df.type.str.replace(r'.*MINOR.*|.*JUV.*|.*TRUANT.*|.*CHILD.*', 'Minor', r
df.type = df.type.str.replace(r'.*MALICIOUS MISCHIEF.*', 'Malicious Mischief', rege
df.type = df.type.str.replace(r'.*PARKING.*', 'Parking', regex=True)
df.type = df.type.str.replace(r'.*INJURE.*OBSTRUCT.*|.*INJURED PERSON.*|.*PERSON DO
df.type = df.type.str.replace(r'.*BURGLARY.*', 'Burglary', regex=True)
df.type = df.type.str.replace(r'.*MENTALLY DISTURBED.*|.*SICK PERSON.*', 'Mentally
df.type = df.type.str.replace(r'MURDER.*|CORONERS CASE', 'Murder', regex=True)
df.type = df.type.str.replace(r'.*DOMESTIC.*', 'Domestic Violence', regex=True)
df.type = df.type.str.replace(r'.*EXTORTION.*', 'Extortion', regex=True)
df.type = df.type.str.replace(r'.*CALL.*', 'Unknown', regex=True)
df.type = df.type.str.replace(r'.*SOLICITING.*|.*MOLEST.*', 'Sex Crime', regex=True
df['type'] = df['type'].str.lower()
end = time.time()
print("Time:", end-start)
```

```
Time: 68.08530569076538
```

In [18]:
```
# Converting the types into lower case
df['type'] = df['type'].str.lower()
```

In [19]:
```
# Checking the clean type:
type_count=df.groupby("type").final_dispo.count()
display(type_count)
```

```
type
alarm                            117688
animal                             3631
assault                            1071
battery                           12358
breach of aoa                        20
                                   ...
unknown                           39907
vehicle                           59898
violation of protective order      5466
weapon                            31054
welfare                          117356
Name: final_dispo, Length: 62, dtype: int64
```

In [20]:
```
# Converting the call_types into lower case
df['call_type'] = df['call_type'].str.lower()
call_type_count=df.groupby("call_type").final_dispo.count()
display(type_count)
```

```
type
alarm                              117688
animal                               3631
assault                              1071
battery                             12358
breach of aoa                          20
                                      ...
unknown                             39907
vehicle                             59898
violation of protective order        5466
weapon                              31054
welfare                            117356
Name: final_dispo, Length: 62, dtype: int64
```

In [21]:
```
display(df.shape)
df.head(3)
```

```
(1460576, 10)
```

Out[21]:

	priority	offense_date	offense_time	call_type	final_dispo	address	year	mo
0	2	2018-01-01	00:00:02	fire department request for pd	No report required; dispatch record only	[2900]-[3000] ALUM ROCK AV	2018	
1	3	2018-01-01	00:00:15	alarm, audible	No Response	[4700]-[4800] CALENDULA CT	2018	
2	4	2018-01-01	00:00:32	disturbance, music	Canceled	[3100]-[3200] WILLIAMSBURG DR	2018	

In [22]:
```
df.columns
```

Out[22]:
```
Index(['priority', 'offense_date', 'offense_time', 'call_type', 'final_dispo',
       'address', 'year', 'month', 'month_year', 'type'],
      dtype='object')
```

We decided to break up the notebook into 2 notebooks so that we can separate the resources necessary to do the preliminary data cleaning process vs the data analytics. Since the files are so large, reloading the dataset and rerunning the cells will be very taxing on some less powerful laptops and time consuming to rerun. In order for this to be achieved, we will export the clean dataset into 1 file for Notebook 2.

In [23]:
```
# Exporting the clean dataset into 1 file for Notebook 2.
# df.to_csv(r'C:\Users\Overbo\Desktop\CIS9 Data Science\Labs\Lab Project\PoliceCall
```