

part2_project_cleanup_EDA

July 30, 2021



Modeling and Forecasting Crime Rate in Colorado

Data Science Capstone Project, part 2; (pre-processing DataFrames and EDA) * Student name: Elena Kazakova * Student pace: Full-time * Cohort: DS02222021 * Scheduled project review date: 07/26/2021 * Instructor name: James Irving * Application url: TBD

TABLE OF CONTENTS

- **Section 1**
- **Section 2**
- **Section 3**

1 OBTAIN

If you are running this notebook without restarting the kernel replace ‘%load_ext autoreload’ in imports with ‘%reload_ext autoreload’

1.1 Imports

```
[1]: # Importing packages
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import statsmodels
import statsmodels.tsa.api as tsa
import plotly.express as px
import plotly.io as pio
import plotly
import math
from math import sqrt
import holidays
import pmdarima as pm

from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error

import pickle
#import shutil
import os
import json

# from pathlib import Path
# import subprocess
# import io

import warnings
warnings.filterwarnings(action='ignore', category=FutureWarning)

from functions_all import *

%reload_ext autoreload
%autoreload 2
%matplotlib inline

[2]: CO_zip_json=json.load(open('data/co_zip.min.json', 'r'))
CO_county_json=json.load(open('data/CO_counties_geo.json', 'r'))
```

1.2 Data

1.2.1 Data source and data description

Data is from FBI Crime Data Explorer [NIBRS data for Colorado from 2009-2019](#)

The [data dictionary](#) is and a [record description](#) are available.

The description of the main and reference tables is in data/README.md file. The FBI implemented some changes to the files structure in 2016 and removed the sqlite create and load scripts from the zip directories. Another fact worth mentioning is that files 'nibrs__property_desc.csv' from 2014 and 2015 have duplicated nibrs__property_desc_ids (unique identifier in the nibrs__property_desc table) which complicated the loading of the data.

1. The first part of the scrubbing process (working with sqlite3 database, production1) is in [part I notebook](#). It takes about 12 minutes to run the code in this notebook. The following notebook is using dataframes created in part I.
2. Also the full original data description is in the same [part I notebook](#).
3. The notebook with database creation is in the [part ZERO notebook](#). The referenced database is in *data/sqlite/db/production1*. It takes 2.5 minutes to run the database creation script.

2 SCRUB

2.1 Part I, pre-processing the data in SQL database

In part I the following dataframes have been created and saved in the pickle files:

1. df_incident: data/pickled_dataframes/incident.pickle; main incident DF with date/time of an
2. df_offense: data/pickled_dataframes/offense.pickle; main offense DF with offense names and c
3. df_offender: data/pickled_dataframes/offender.pickle; main offender DF with demographic inf
4. df_victim: data/pickled_dataframes/victim.pickle; main victim DF with demographic info
5. df_weapon: data/pickled_dataframes/weapon.pickle; main weapon DF with a weapon category use
6. df_bias: data/pickled_dataframes/bias.pickle; main bias DF with offense bias motivation
7. df_rel: data/pickled_dataframes/relationship.pickle; main victim-offender relationship DF w

2.2 Part II, scrubbing the data in DataFrames

2.2.1 Using pickle files to create dataframes

```
[3]: with open('data/pickled_dataframes/incident.pickle', 'rb') as f:
      df_incident=pickle.load(f)
      df_incident.head()
```

```
[3]:   agency_id  incident_id  incident_date  incident_hour  primary_county  \
0        1971    51264520  2009-01-05 00:00:00             22      Kit Carson
1        1971    51264521  2009-01-13 00:00:00             25      Kit Carson
2        1971    51264523  2009-01-17 00:00:00             19      Kit Carson
3        1971    51264524  2009-01-20 00:00:00             25      Kit Carson
4        1971    51264525  2009-01-21 00:00:00             25      Kit Carson
```

```

icpsr_zip
0      80807
1      80807
2      80807
3      80807
4      80807

```

```
[4]: len(df_incident)
```

```
[4]: 2819463
```

```
[5]: with open('data/pickled_dataframes/offense.pickle', 'rb') as f:
      df_offense=pickle.load(f)
      df_offense.head()
```

```
[5]:  offense_id  incident_id  location_name  offense_name \
0      53563151      51264520  Residence/Home  Aggravated Assault
1      53563402      51264521  Residence/Home  Theft From Motor Vehicle
2      53558278      51264523  School/College  Drug/Narcotic Violations
3      53558279      51264523  School/College  Drug Equipment Violations
4      53563403      51264524  Other/Unknown  Impersonation
```

```

      crime_against  offense_category_name
0      Person      Assault Offenses
1      Property  Larceny/Theft Offenses
2      Society  Drug/Narcotic Offenses
3      Society  Drug/Narcotic Offenses
4      Property      Fraud Offenses

```

```
[6]: len(df_offense)
```

```
[6]: 3201143
```

```
[7]: with open('data/pickled_dataframes/offender.pickle', 'rb') as f:
      df_offender=pickle.load(f)
      df_offender.head()
```

```
[7]:  offender_id  incident_id  age_num  sex_code  race  age_group  ethnicity
0      57702592      51264520      25    Male  White  Age in Years      None
1      57702593      51264521      20    Male  White  Age in Years      None
2      57702595      51264523      20    Male  White  Age in Years      None
3      57702596      51264524      55    Male  White  Age in Years      None
4      57702597      51264525      55    Male  White  Age in Years      None
```

```
[8]: len(df_offender)
```

```
[8]: 3197991
```

```
[9]: with open('data/pickled_dataframes/victim.pickle', 'rb') as f:
      df_victim=pickle.load(f)
      df_victim.head()
```

```
[9]:  victim_id  incident_id  age_num  sex_code  resident_status_code  race  \
0    55514644    51264520      23    Male          Resident  White
1    55514645    51264521      49  Female        Non-resident  White
2    55514647    51264523      28  Female          Resident  White
3    55514648    51264524      16    Male          Resident  White
4    55514649    51264525      28  Female          Resident  White

      age_group      ethnicity      victim_type
0  Age in Years  Not Hispanic or Latino  Law Enforcement Officer
1  Age in Years              Unknown      Individual
2           None              None      Society/Public
3  Age in Years              Unknown      Individual
4  Age in Years              Unknown      Individual
```

```
[10]: len(df_victim)
```

```
[10]: 3229640
```

```
[11]: with open('data/pickled_dataframes/weapon.pickle', 'rb') as f:
      df_weapon=pickle.load(f)
      df_weapon.head()
```

```
[11]:  offense_id      weapon
0    53563151  Non-automatic firearm
1    53558280  Non-automatic firearm
2    53563153  Non-automatic firearm
3    53579810  Non-automatic firearm
4    53572975  Non-automatic firearm
```

```
[12]: len(df_weapon)
```

```
[12]: 551049
```

```
[13]: with open('data/pickled_dataframes/bias.pickle', 'rb') as f:
      df_bias=pickle.load(f)
      df_bias.head()
```

```
[13]:  offense_id  bias_name
0    53563151      None
1    53563402      None
2    53558278      None
3    53558279      None
4    53563403      None
```

```
[14]: len(df_bias)
```

```
[14]: 3201158
```

```
[15]: with open('data/pickled_dataframes/relationship.pickle', 'rb') as f:
      df_rel=pickle.load(f)
      df_rel.head()
```

```
[15]:
```

	victim_id	offender_id	relationship_name
0	55514644	57702592	Victim was Otherwise Known
1	55514649	57702597	Victim Was Stepchild
2	55514652	57702601	Victim Was Spouse
3	55514653	57702602	Victim Was Boyfriend/Girlfriend
4	55514655	57702604	Victim Was Child

```
[16]: len(df_rel)
```

```
[16]: 794157
```

The next step is scrubbing the dataframes

2.2.2 Checking for duplicates, missing values and other abnormalities, incident table

```
[17]: df_incident.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2819463 entries, 0 to 2819462
Data columns (total 6 columns):
#   Column          Dtype
---  -
0   agency_id       int64
1   incident_id     int64
2   incident_date   object
3   incident_hour   int64
4   primary_county  object
5   icpsr_zip       object
dtypes: int64(3), object(3)
memory usage: 129.1+ MB
```

Converting incident_date column to a datetime type

```
[18]: df_incident.head()
```

```
[18]:
```

	agency_id	incident_id	incident_date	incident_hour	primary_county	\
0	1971	51264520	2009-01-05 00:00:00	22	Kit Carson	
1	1971	51264521	2009-01-13 00:00:00	25	Kit Carson	
2	1971	51264523	2009-01-17 00:00:00	19	Kit Carson	
3	1971	51264524	2009-01-20 00:00:00	25	Kit Carson	
4	1971	51264525	2009-01-21 00:00:00	25	Kit Carson	

```

    icpsr_zip
0      80807
1      80807
2      80807
3      80807
4      80807

```

```
[19]: df_incident['timestamp']=pd.to_datetime(df_incident.incident_date)
df_incident.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2819463 entries, 0 to 2819462
Data columns (total 7 columns):
#   Column          Dtype
---  -
0   agency_id       int64
1   incident_id     int64
2   incident_date   object
3   incident_hour   int64
4   primary_county  object
5   icpsr_zip       object
6   timestamp       datetime64[ns]
dtypes: datetime64[ns](1), int64(3), object(3)
memory usage: 150.6+ MB

```

```
[20]: df_incident.sort_values('timestamp', ascending=True)
```

```
[20]:
```

	agency_id	incident_id	incident_date	incident_hour	\
40230	1984	51269326	2009-01-01 00:00:00	12	
184495	2119	47560373	2009-01-01 00:00:00	1	
184494	2119	47560372	2009-01-01 00:00:00	22	
109516	1831	49921447	2009-01-01 00:00:00	10	
109514	1831	49942735	2009-01-01 00:00:00	25	
...	
2685274	2119	122863693	31-Dec-19	16	
2807672	2010	119343129	31-Dec-19	21	
2719556	1920	120335390	31-Dec-19	21	
2583020	2051	120330349	31-Dec-19	23	
2686115	2119	122863605	31-Dec-19	15	
	primary_county	icpsr_zip	timestamp		
40230	Larimer	80525	2009-01-01		
184495	Denver	80204	2009-01-01		
184494	Denver	80204	2009-01-01		
109516	Adams	80031	2009-01-01		
109514	Adams	80031	2009-01-01		
...		

2685274	Denver	80204	2019-12-31
2807672	Moffat	81625	2019-12-31
2719556	El Paso	80901	2019-12-31
2583020	Pueblo	81003	2019-12-31
2686115	Denver	80204	2019-12-31

[2819463 rows x 7 columns]

Checking for duplicates and dropping them

```
[21]: df=df_incident[df_incident.duplicated(subset=['incident_id'],keep=False)].
      ↪sort_values(by=['incident_id','timestamp'])
df
```

```
[21]:      agency_id  incident_id  incident_date  incident_hour  \
1456847      1908    85757101  2015-08-10 00:00:00           17
1733099      1908    85757101           20-Jan-16           22
1456848      1908    85757105  2015-08-10 00:00:00           17
1733102      1908    85757105           19-Jan-16           10
1456849      1908    85757108  2015-08-10 00:00:00           17
...          ...          ...          ...          ...
1889452      1920    88326562           1-Nov-16           14
1448247      1893    88338695  2015-05-06 00:00:00           15
1830888      1827    88338695           31-Jul-16            7
1448388      1893    88339624  2015-12-09 00:00:00           14
1830718      1827    88339624           6-Oct-16           13
```

	primary_county	icpsr_zip	timestamp
1456847	Douglas	80124	2015-08-10
1733099	Douglas	80124	2016-01-20
1456848	Douglas	80124	2015-08-10
1733102	Douglas	80124	2016-01-19
1456849	Douglas	80124	2015-08-10
...
1889452	El Paso	80901	2016-11-01
1448247	Delta	81416	2015-05-06
1830888	Arapahoe	80012	2016-07-31
1448388	Delta	81416	2015-12-09
1830718	Arapahoe	80012	2016-10-06

[548 rows x 7 columns]

There are 548 duplicate incident_id, they seem to be from different dates, counties, zipcodes. Only the first duplicate will be left in the set. The presence of duplicate incident_ids is most probably a human error when the system got switched to another format in 2016.


```
[22]: # Dropping rows with duplicate ids and 2016 timestamp (because their indices are
      ↪ higher). Removing 'incident_date' column.
```

```
df_incident=df_incident.drop_duplicates(subset=['incident_id'],keep='last')
```

```
[23]: df_incident=df_incident.drop(columns=['incident_date'])
      df_incident.head()
```

```
[23]:
```

	agency_id	incident_id	incident_hour	primary_county	icpsr_zip	timestamp
0	1971	51264520	22	Kit Carson	80807	2009-01-05
1	1971	51264521	25	Kit Carson	80807	2009-01-13
2	1971	51264523	19	Kit Carson	80807	2009-01-17
3	1971	51264524	25	Kit Carson	80807	2009-01-20
4	1971	51264525	25	Kit Carson	80807	2009-01-21

Checking for empty strings/null values and updating the rows with new values

```
[24]: # Cheching for empty strings and null values
      empty_string_count(df_incident)
```

```
Column agency_id empty string count: 0
```

```
Column agency_id null values count: 0
```

```
*****
```

```
Column incident_id empty string count: 0
```

```
Column incident_id null values count: 0
```

```
*****
```

```
Column incident_hour empty string count: 0
```

```
Column incident_hour null values count: 0
```

```
*****
```

```
Column primary_county empty string count: 13771
```

```
Column primary_county null values count: 0
```

```
*****
```

```
Column icpsr_zip empty string count: 2277
```

```
Column icpsr_zip null values count: 0
```

```
*****
```

```
Column timestamp empty string count: 0
```

```
Column timestamp null values count: 0
```

```
*****
```

```
Total number of records in the dataframe: 2819189
```

There are no NaN values but ''(empty string) values are present in primary_county and icpsr_zipcode fields

```
[25]: df=df_incident.loc[df_incident['primary_county']=='']
      df.icpsr_zip.unique()
```

```
[25]: array(['80215'], dtype=object)
```

Due to the fact that all primary_county missing values are associated with 80215 zip code, which belongs to Jefferson county. I am filling in these records primary county with 'Jefferson' string.

```
[26]: df_incident.loc[df_incident.primary_county == '', 'primary_county'] =  
      ↪ 'Jefferson'
```

```
[27]: df=df_incident.loc[df_incident['icpsr_zip']=='']  
      df.agency_id.unique()
```

```
[27]: array([ 1982, 23131, 25314], dtype=int64)
```

The missing zip codes belong to the following agencies: 1. agency_id=1982: Fort Lewis College, located in 81301 zip code 2. agency_id=23131: South Metro Drug Task Force, located in 80160 zip code 3. agency_id=25314: Gypsum Police Department, located in 81637 zip code

The values above will be used to fill in icpsr_zip column values in place of '' values

```
[28]: df_incident.loc[((df_incident.icpsr_zip == '')&(df_incident.agency_id==1982)),  
      ↪ 'icpsr_zip'] = '81301'  
  
      df_incident.loc[((df_incident.icpsr_zip == '')&(df_incident.agency_id==23131)),  
      ↪ 'icpsr_zip'] = '80160'  
  
      df_incident.loc[((df_incident.icpsr_zip == '')&(df_incident.agency_id==25314)),  
      ↪ 'icpsr_zip'] = '81637'
```

```
[29]: empty_string_count(df_incident)
```

```
Column agency_id empty string count: 0  
Column agency_id null values count: 0  
*****  
Column incident_id empty string count: 0  
Column incident_id null values count: 0  
*****  
Column incident_hour empty string count: 0  
Column incident_hour null values count: 0  
*****  
Column primary_county empty string count: 0  
Column primary_county null values count: 0  
*****  
Column icpsr_zip empty string count: 0  
Column icpsr_zip null values count: 0  
*****  
Column timestamp empty string count: 0  
Column timestamp null values count: 0  
*****  
Total number of records in the dataframe: 2819189
```

2.2.3 Checking for duplicates, missing values and other abnormalities, offense table

```
[30]: df_offense.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3201143 entries, 0 to 3201142
Data columns (total 6 columns):
#   Column                Dtype
---  -
0   offense_id            int64
1   incident_id           int64
2   location_name         object
3   offense_name          object
4   crime_against         object
5   offense_category_name object
dtypes: int64(2), object(4)
memory usage: 146.5+ MB
```

```
[31]: df_offense.head()
```

```
[31]:   offense_id  incident_id  location_name  offense_name \
0    53563151    51264520  Residence/Home    Aggravated Assault
1    53563402    51264521  Residence/Home  Theft From Motor Vehicle
2    53558278    51264523  School/College  Drug/Narcotic Violations
3    53558279    51264523  School/College  Drug Equipment Violations
4    53563403    51264524  Other/Unknown    Impersonation

   crime_against  offense_category_name
0         Person      Assault Offenses
1        Property  Larceny/Theft Offenses
2         Society  Drug/Narcotic Offenses
3         Society  Drug/Narcotic Offenses
4        Property      Fraud Offenses
```

Checking for duplicates

```
[32]: df=df_offense[df_offense.duplicated(subset=['offense_id'],keep=False)].
      ↪sort_values(by='offense_id')
df
```

```
[32]: Empty DataFrame
Columns: [offense_id, incident_id, location_name, offense_name, crime_against,
offense_category_name]
Index: []
```

There are no duplicate offense_ids

Checking for empty strings/null values and updating the rows with new values

```
[33]: empty_string_count(df_offense)
```

```

Column offense_id empty string count: 0
Column offense_id null values count: 0
*****
Column incident_id empty string count: 0
Column incident_id null values count: 0
*****
Column location_name empty string count: 0
Column location_name null values count: 0
*****
Column offense_name empty string count: 0
Column offense_name null values count: 0
*****
Column crime_against empty string count: 0
Column crime_against null values count: 0
*****
Column offense_category_name empty string count: 0
Column offense_category_name null values count: 0
*****
Total number of records in the dataframe: 3201143

```

There are no rows with empty strings or NaN values

2.2.4 Checking for duplicates, missing values and other abnormalities, victim table

```
[34]: df_victim.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3229640 entries, 0 to 3229639
Data columns (total 9 columns):
 #   Column                Dtype
---  -
 0   victim_id             int64
 1   incident_id           int64
 2   age_num               object
 3   sex_code              object
 4   resident_status_code object
 5   race                 object
 6   age_group            object
 7   ethnicity            object
 8   victim_type          object
dtypes: int64(2), object(7)
memory usage: 221.8+ MB

```

```
[35]: df_victim.head()
```

```

[35]:   victim_id  incident_id  age_num  sex_code  resident_status_code  race \
0   55514644    51264520     23    Male                Resident  White
1   55514645    51264521     49  Female                Non-resident  White

```

2	55514647	51264523				None
3	55514648	51264524	28	Female	Resident	White
4	55514649	51264525	16	Male	Resident	White

	age_group	ethnicity	victim_type
0	Age in Years	Not Hispanic or Latino	Law Enforcement Officer
1	Age in Years	Unknown	Individual
2	None	None	Society/Public
3	Age in Years	Unknown	Individual
4	Age in Years	Unknown	Individual

Checking for duplicates The same person can be a victim in several incidents therefore we are only checking for duplicates with victim_ids AND incident_ids

```
[36]: df=df_victim[df_victim.
      ↳duplicated(subset=['victim_id','incident_id'],keep=False)].
      ↳sort_values(by='victim_id')
      df
```

```
[36]: Empty DataFrame
      Columns: [victim_id, incident_id, age_num, sex_code, resident_status_code, race,
      age_group, ethnicity, victim_type]
      Index: []
```

No duplicates found

Checking for empty strings/null values

```
[37]: empty_string_count(df_victim)

Column victim_id empty string count: 0
Column victim_id null values count: 0
*****
Column incident_id empty string count: 0
Column incident_id null values count: 0
*****
Column age_num empty string count: 1032415
Column age_num null values count: 0
*****
Column sex_code empty string count: 991009
Column sex_code null values count: 0
*****
Column resident_status_code empty string count: 1068153
Column resident_status_code null values count: 0
*****
Column race empty string count: 0
Column race null values count: 991009
*****
Column age_group empty string count: 0
```

```

Column age_group null values count: 991009
*****
Column ethnicity empty string count: 0
Column ethnicity null values count: 1013219
*****
Column victim_type empty string count: 0
Column victim_type null values count: 0
*****
Total number of records in the dataframe: 3229640

```

Abnormal values, victim table

race, NaN values

```
[38]: df=df_victim[df_victim.race.isnull()]
      df.victim_type.unique()
```

```
[38]: array(['Society/Public', 'Business', 'Government', 'Other', 'Unknown',
            'Financial Institution', 'Religious Organization'], dtype=object)
```

The NAN values in the race column of victims with of types ‘Society/Public’, ‘Business’, ‘Government’, ‘Other’, ‘Unknown’, ‘Financial Institution’, and ‘Religious Organization’ will be replaced with ‘NA’ value. Due to the fact that these victim types are the only types of NULL race records, all race NULL values will be replaced with ‘NA’.

```
[39]: df_victim.loc[df_victim.race.isnull(), 'race'] = 'NA'
```

ethnicity, NaN values

```
[40]: df=df_victim[df_victim.ethnicity.isnull()]
      df.victim_type.unique()
```

```
[40]: array(['Society/Public', 'Individual', 'Business', 'Government', 'Other',
            'Unknown', 'Financial Institution', 'Religious Organization',
            'Law Enforcement Officer'], dtype=object)
```

```
[41]: df=df_victim[((df_victim.ethnicity.isnull()) & (df.victim_type.isin(['Law_
      ↳Enforcement Officer', 'Individual'])))
print('Number of records with empty string in resident_status_code and_
      ↳Individual or \
Law Enforcement victim type: {}'.format(len(df)))
df.head()
```

Number of records with empty string in resident_status_code and Individual or Law Enforcement victim type: 22210

```
[41]:
```

	victim_id	incident_id	age_num	sex_code	resident_status_code	\
7	55514663	51264539	65	Male	Resident	
37	55514681	51264550	24	Male	Resident	
55	55514698	51264566	29	Female	Resident	

116	54355540	50210712	39	Female	Resident
13355	54431861	50279345	43	Male	Resident

	race	age_group	ethnicity	victim_type
7	White	Age in Years	None	Individual
37	White	Age in Years	None	Individual
55	White	Age in Years	None	Individual
116	White	Age in Years	None	Individual
13355	Black or African American	Age in Years	None	Individual

1. The NaN values in the ethnicity column of victims with of types ‘Society/Public’, ‘Business’, ‘Government’, ‘Other’, ‘Unknown’, ‘Financial Institution’, and ‘Religious Organization’ will be replaced with ‘NA’ value 2. The NaN values in the ethnicity column of victims with of types ‘Law Enforcement Officer’, ‘Individual’ will be replaced with ‘Unknown’ value

```
[42]: df_victim.loc[(df_victim.ethnicity.isnull()
                  &df_victim.victim_type.isin(['Society/Public','Business',
                  →'Government','Other','Unknown',
                  →'Financial Institution','Religious
                  →Organization']))], 'ethnicity'] = 'NA'

df_victim.loc[(df_victim.ethnicity.isnull()
                  &df_victim.victim_type.isin(['Law Enforcement Officer',
                  →'Individual']))], 'ethnicity'] = 'Unknown'
```

age_group, NaN values

```
[43]: df=df_victim[df_victim.age_group.isnull()]
df.victim_type.unique()
```

```
[43]: array(['Society/Public', 'Business', 'Government', 'Other', 'Unknown',
          'Financial Institution', 'Religious Organization'], dtype=object)
```

The NAN values in the age_group column of victims with of types ‘Society/Public’, ‘Business’, ‘Government’, ‘Other’, ‘Unknown’, ‘Financial Institution’, and ‘Religious Organization’ will be replaced with ‘NA’ value. Due to the fact that these victim types are the only types of NULL age_group records, all age_group NULL will replaced with ‘NA’.

```
[44]: df_victim.loc[df_victim.age_group.isnull(), 'age_group'] = 'NA'
```

age_num, empty string values

```
[45]: df=df_victim[df_victim.age_num=='']
print('Number of records with empty string in age_num: {}'.format(len(df)))
df.victim_type.unique()
```

Number of records with empty string in age_num: 1032415

```
[45]: array(['Society/Public', 'Business', 'Government', 'Other',
          'Law Enforcement Officer', 'Individual', 'Unknown',
```

```
'Financial Institution', 'Religious Organization'], dtype=object)
```

```
[46]: df=df_victim[((df_victim.age_num=='') & (df_victim_type.isin(['Law Enforcement Officer', 'Individual'])))  
print('Number of records with empty string in age_num and Individual or Law Enforcement victim type: {}'.format(len(df)))
```

Number of records with empty string in age_num and Individual or Law Enforcement victim type: 41406

1. The empty string values in the age_num column of victims with types ‘Society/Public’, ‘Business’, ‘Government’, ‘Other’, ‘Unknown’, ‘Financial Institution’, and ‘Religious Organization’ will be replaced with 999. 2. The empty string values in the age_num column of victims with types ‘Law Enforcement Officer’, ‘Individual’ AND age_group equal ‘Unknown’ will be replaced with 999. 3. The empty string values in the age_num column of victims with of types ‘Law Enforcement Officer’, ‘Individual’ AND age_group in (‘7-364 Days Old’, ‘Under 24 Hours’, ‘1-6 Days Old’) will be replaced with 0. 4. The empty string values in the age_num column of victims with of types ‘Law Enforcement Officer’, ‘Individual’ AND age_group ‘Over 98 Years Old’ will be replaced with 99.

```
[47]: df_victim.loc[((df_victim.age_num=='')  
                  &df_victim.victim_type.isin(['Society/Public','Business',  
                  'Government','Other','Unknown',  
                  'Financial Institution','Religious Organization'])), 'age_num'] = '999'  
df_victim.loc[((df_victim.age_num=='')  
                  &(df_victim.victim_type.isin(['Law Enforcement Officer',  
                  'Individual'])))  
                  &(df_victim.age_group.isin(['7-364 Days Old','Under 24 Hours',  
                  '1-6 Days Old'])))], 'age_num'] = '0'  
  
df_victim.loc[((df_victim.age_num=='')  
                  &(df_victim.victim_type.isin(['Law Enforcement Officer',  
                  'Individual'])))  
                  &(df_victim.age_group=='Over 98 Years Old')), 'age_num'] = '99'  
  
df_victim.loc[((df_victim.age_num=='')  
                  &(df_victim.victim_type.isin(['Law Enforcement Officer',  
                  'Individual'])))  
                  &(df_victim.age_group=='Unknown')), 'age_num'] = '999'
```

sex_code, empty string values

```
[48]: df=df_victim[df_victim.sex_code=='']  
print('Number of records with empty string in sex_code: {}'.format(len(df)))  
df.victim_type.unique()
```

Number of records with empty string in sex_code: 991009


```
[48]: array(['Society/Public', 'Business', 'Government', 'Other', 'Unknown',
          'Financial Institution', 'Religious Organization'], dtype=object)
```

The empty string values in the `sex_code` column of victims with of types ‘**Society/Public**’, ‘**Business**’, ‘**Government**’, ‘**Other**’, ‘**Unknown**’, ‘**Financial Institution**’, and ‘**Religious Organization**’ will be replaced with ‘**NA**’ value. Due to the fact that these victim types are the only types of `sex_code` empty string records, all `sex_code` empty string values will be replaced with ‘**NA**’.

```
[49]: df_victim.loc[df_victim.sex_code=='', 'sex_code'] = 'NA'
```

resident_status_code, empty string values

```
[50]: df=df_victim[df_victim.resident_status_code=='']
print('Number of records with empty string in resident_status_code: {}'.
      format(len(df)))
df.victim_type.unique()
```

Number of records with empty string in `resident_status_code`: 1068153

```
[50]: array(['Society/Public', 'Business', 'Government', 'Other', 'Unknown',
          'Financial Institution', 'Religious Organization', 'Individual',
          'Law Enforcement Officer'], dtype=object)
```

```
[51]: df=df_victim[((df_victim.resident_status_code=='') & (df.victim_type.isin(['Law
      Enforcement Officer', 'Individual'])))
print('Number of records with empty string in resident_status_code and
      Individual or \
      Law Enforcement victim type: {}'.format(len(df)))
```

Number of records with empty string in `resident_status_code` and Individual or Law Enforcement victim type: 77144

1. The empty string values in the `resident_status_code` column of victims with of types ‘**Society/Public**’, ‘**Business**’, ‘**Government**’, ‘**Other**’, ‘**Unknown**’, ‘**Financial Institution**’, and ‘**Religious Organization**’ will be replaced with ‘**NA**’ value 2. The empty string values in the `resident_status_code` column of victims with of types ‘**Law Enforcement Officer**’, ‘**Individual**’ will be replaced with ‘**Unknown**’ value

```
[52]: df_victim.loc[((df_victim.resident_status_code=='')
                    &df_victim.victim_type.isin(['Society/Public','Business',
      →'Government','Other',
                                                    'Unknown','Financial Institution',
                                                    'Religious Organization']))],
      →'resident_status_code'] = 'NA'

df_victim.loc[((df_victim.resident_status_code=='')
              &(df_victim.victim_type.isin(['Law Enforcement Officer',
      →'Individual'])))],
      →'resident_status_code'] = 'Unknown'
```

Renaming the columns

```
[53]: df_victim=df_victim.rename(columns={'age_num': 'victim_age', 'sex_code':  
    ↳'victim_sex',  
    'resident_status_code':  
    ↳'victim_resident_status','race': 'victim_race',  
    'age_group':'victim_age_group','ethnicity':  
    ↳'victim_ethnicity'})
```

```
[54]: empty_string_count(df_victim)
```

```
Column victim_id empty string count: 0  
Column victim_id null values count: 0  
*****  
Column incident_id empty string count: 0  
Column incident_id null values count: 0  
*****  
Column victim_age empty string count: 0  
Column victim_age null values count: 0  
*****  
Column victim_sex empty string count: 0  
Column victim_sex null values count: 0  
*****  
Column victim_resident_status empty string count: 0  
Column victim_resident_status null values count: 0  
*****  
Column victim_race empty string count: 0  
Column victim_race null values count: 0  
*****  
Column victim_age_group empty string count: 0  
Column victim_age_group null values count: 0  
*****  
Column victim_ethnicity empty string count: 0  
Column victim_ethnicity null values count: 0  
*****  
Column victim_type empty string count: 0  
Column victim_type null values count: 0  
*****  
Total number of records in the dataframe: 3229640
```

2.2.5 Checking for duplicates, missing values and other abnormalities, offender table

```
[55]: df_offender.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3197991 entries, 0 to 3197990  
Data columns (total 7 columns):  
#   Column      Dtype  
---  ---  
-----
```

```

0  offender_id  int64
1  incident_id  int64
2  age_num      object
3  sex_code     object
4  race         object
5  age_group    object
6  ethnicity    object
dtypes: int64(2), object(5)
memory usage: 170.8+ MB

```

```
[56]: df_offender.head()
```

```

[56]:   offender_id  incident_id  age_num  sex_code  race  age_group  ethnicity
0      57702592      51264520      25     Male  White  Age in Years      None
1      57702593      51264521      20     Male  White  Age in Years      None
2      57702595      51264523      20     Male  White  Age in Years      None
3      57702596      51264524      20     Male  White  Age in Years      None
4      57702597      51264525      55     Male  White  Age in Years      None

```

Checking for duplicates The same person can be an offender in several incidents therefore we are only checking for duplicates with offender_ids AND incident_ids

```

[57]: df=df_offender[df_offender.duplicated(subset=['offender_id',
→ 'incident_id'],keep=False)].sort_values(by='offender_id')
df

```

```

[57]: Empty DataFrame
Columns: [offender_id, incident_id, age_num, sex_code, race, age_group,
ethnicity]
Index: []

```

No duplicates found

Checking for empty strings/null values

```
[58]: empty_string_count(df_offender)
```

```

Column offender_id empty string count: 0
Column offender_id null values count: 0
*****
Column incident_id empty string count: 0
Column incident_id null values count: 0
*****
Column age_num empty string count: 1509300
Column age_num null values count: 0
*****
Column sex_code empty string count: 912428
Column sex_code null values count: 0
*****

```

```

Column race empty string count: 0
Column race null values count: 912428
*****
Column age_group empty string count: 0
Column age_group null values count: 912428
*****
Column ethnicity empty string count: 0
Column ethnicity null values count: 1972733
*****
Total number of records in the dataframe: 3197991

```

Abnormal values, offender table

ethnicity, NaN values

```

[59]: print('Number of records with NaN values in ethnicity: {}'.
        ↳format(df_offender['ethnicity'].isnull().sum()))
df_offender['ethnicity'].value_counts()

```

Number of records with NaN values in ethnicity: 1972733

```

[59]: Not Hispanic or Latino      577692
      Unknown                    434094
      Hispanic or Latino          213472
      Name: ethnicity, dtype: int64

```

The NaN value in the **ethnicity** column of offender table will be replaced with ‘**Unknown**’ value

```

[60]: df_offender.loc[df_offender.ethnicity.isnull(), 'ethnicity'] = 'Unknown'

```

race, NaN values

```

[61]: print('Number of records with NaN values in race: {}'.
        ↳format(df_offender['race'].isnull().sum()))
df_offender['race'].value_counts()

```

Number of records with NaN values in race: 912428

```

[61]: White                        1438051
      Unknown                    549611
      Black or African American    270218
      Asian                      11110
      American Indian or Alaska Native 10566
      Asian, Native Hawaiian, or Other Pacific Islander 5175
      Native Hawaiian or Other Pacific Islander 832
      Name: race, dtype: int64

```

The NaN value in the **race** column of offender table will be replaced with **Unknown** value

```

[62]: df_offender.loc[df_offender.race.isnull(), 'race'] = 'Unknown'

```

age_group, NaN values

```
[63]: print('Number of records with NaN values in age_group: {}'.format(df_offender['age_group'].isnull().sum()))
df_offender['age_group'].value_counts()
```

Number of records with NaN values in age_group: 912428

```
[63]: Age in Years      1688691
      Unknown          596172
      Over 98 Years Old    700
      Name: age_group, dtype: int64
```

```
[64]: df_offender.loc[df_offender['age_group'].isnull()]
```

```
[64]:
```

	offender_id	incident_id	age_num	sex_code	race	age_group	\
1	57702593	51264521			Unknown	None	
3	57702596	51264524			Unknown	None	
7	57702612	51264539			Unknown	None	
11	57702603	51264530			Unknown	None	
13	57702605	51264532			Unknown	None	
...		
3197957	133652222	117657878			Unknown	None	
3197970	133657157	117657929			Unknown	None	
3197974	133652341	117648019			Unknown	None	
3197980	133652400	117658019			Unknown	None	
3197981	133652472	117653056			Unknown	None	

	ethnicity
1	Unknown
3	Unknown
7	Unknown
11	Unknown
13	Unknown
...	...
3197957	Unknown
3197970	Unknown
3197974	Unknown
3197980	Unknown
3197981	Unknown

```
[912428 rows x 7 columns]
```

The NaN value in the **age_group** column of offender table will be replaced with **Unknown** value. Spot checking the records did not generate any insights. All those offenders are simply not known, never got identified.

```
[65]: df_offender.loc[df_offender.age_group.isnull(), 'age_group'] = 'Unknown'
```

age_num, empty string values

```
[66]: df=df_offender[df_offender.age_num=='']
print('Number of records with empty string in age_num: {}'.format(len(df)))
print('Number of records with NaN values in age_group: {}'.
      ↪format(df['age_group'].isnull().sum()))
df['age_group'].value_counts()
```

Number of records with empty string in age_num: 1509300
 Number of records with NaN values in age_group: 0

```
[66]: Unknown          1508600
      Over 98 Years Old    700
      Name: age_group, dtype: int64
```

1. The empty string in the **age_num** of offender table with age_group values equal **‘Over 98 Years Old’** will be replaced with **99** value 2. The empty string in the **age_num** of offender table with age_group values equal **‘Unknown’** will be replaced with **999** value

```
[67]: df_offender.loc[((df_offender.age_num=='')&(df_offender.age_group=='Over 98_
      ↪Years Old')), 'age_num'] = '99'

df_offender.loc[((df_offender.age_num=='')
      ↪&(df_offender.age_group=='Unknown')), 'age_num'] = '999'
```

sex_code, empty string values

```
[68]: df_offender['sex_code'].value_counts()
```

```
[68]: Male          1325988
      912428
      Female        501641
      Unknown       457934
      Name: sex_code, dtype: int64
```

The empty string value in the **sex_code** column of offender table will be replaced with **‘Unknown’** value

```
[69]: df_offender.loc[df_offender.sex_code='', 'sex_code'] = 'Unknown'
```

Renaming the columns

```
[70]: df_offender=df_offender.rename(columns={'age_num': 'offender_age', 'sex_code':_
      ↪'offender_sex',
      ↪'race': 'offender_race', 'age_group':
      ↪'offender_age_group',
      ↪'ethnicity': 'offender_ethnicity'})
```

```
[71]: empty_string_count(df_offender)
```

Column offender_id empty string count: 0
 Column offender_id null values count: 0

```

Column incident_id empty string count: 0
Column incident_id null values count: 0
*****
Column offender_age empty string count: 0
Column offender_age null values count: 0
*****
Column offender_sex empty string count: 0
Column offender_sex null values count: 0
*****
Column offender_race empty string count: 0
Column offender_race null values count: 0
*****
Column offender_age_group empty string count: 0
Column offender_age_group null values count: 0
*****
Column offender_ethnicity empty string count: 0
Column offender_ethnicity null values count: 0
*****
Total number of records in the dataframe: 3197991

```

2.2.6 Checking for duplicates, missing values and other abnormalities, weapon table

[72]: `df_weapon.info()`

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551049 entries, 0 to 551048
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   offense_id  551049 non-null  int64
 1   weapon      551049 non-null  object
dtypes: int64(1), object(1)
memory usage: 8.4+ MB

```

[73]: `empty_string_count(df_weapon)`

```

Column offense_id empty string count: 0
Column offense_id null values count: 0
*****
Column weapon empty string count: 0
Column weapon null values count: 0
*****
Total number of records in the dataframe: 551049

```

[74]: `# Checking for duplicates in offense_id column`
`df=df_weapon[df_weapon.duplicated(subset=['offense_id'],keep=False)].`
`↪sort_values(by='offense_id')`
`df`

```
[74]:
```

	offense_id	weapon
14148	51643793	Non-automatic firearm
14149	51643793	Other weapon
14002	51646792	Other weapon
14003	51646792	Other weapon
13978	51646830	Non-automatic firearm
...
528537	148659366	Non-automatic firearm
528538	148659366	Other weapon
528539	148659366	Other weapon
528614	148660117	Other weapon
528613	148660117	Non-automatic firearm

[19709 rows x 2 columns]

There can be several types of weapons used in one offense. For the sake of simplicity I will drop duplicates from the table.

```
[75]: df_weapon=df_weapon.drop_duplicates(subset=['offense_id'],keep='last')
```

2.2.7 Checking for duplicates, missing values and other abnormalities, bias table

```
[76]: df_bias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3201158 entries, 0 to 3201157
Data columns (total 2 columns):
#   Column      Dtype
---  -
0   offense_id  int64
1   bias_name   object
dtypes: int64(1), object(1)
memory usage: 48.8+ MB
```

```
[77]: empty_string_count(df_bias)
```

```
Column offense_id empty string count: 0
Column offense_id null values count: 0
*****
Column bias_name empty string count: 0
Column bias_name null values count: 0
*****
Total number of records in the dataframe: 3201158
```

```
[78]: # Checking for duplicates in offense_id column
df=df_bias[df_bias.duplicated(subset=['offense_id'],keep=False)].
    ↪sort_values(by='offense_id')
df
```



```
[78]:
```

	offense_id	bias_name
2060439	111048055	Anti-White
2060440	111048055	Anti-Jewish
1926231	111048057	Anti-White
1926232	111048057	Anti-Jewish
1916086	111048061	Anti-White
1916087	111048061	Anti-Jewish
2060448	111048070	Anti-Hispanic or Latino
2060447	111048070	Anti-Multi-Racial Group
2060446	111048070	Anti-Black or African American
2060445	111048070	Anti-White
2060443	111048071	Anti-Multi-Racial Group
2060444	111048071	Anti-Hispanic or Latino
2060441	111048071	Anti-White
2060442	111048071	Anti-Black or African American
2029958	111048073	Anti-Female Homosexual (Lesbian)
2029957	111048073	Anti-Jewish
2029956	111048073	Anti-White
2755767	123052012	Anti-Black or African American
2755768	123052012	Anti-Islamic (Muslem)
3114725	132470461	Anti-Black or African American
3114726	132470461	Anti-Jewish
2827001	133862508	Anti-Multi-Racial Group
2827002	133862508	Anti-Female Homosexual (Lesbian)
3070181	146759794	Anti-Black or African American
3070182	146759794	Anti-Male Homosexual (Gay)

```
[79]: df_bias.duplicated(subset='offense_id').sum()
```

```
[79]: 15
```

There can be several types of biases in one offense. The number of duplicates is low. For the sake of simplicity I will drop duplicates from the table.

```
[80]: df_bias=df_bias.drop_duplicates(subset=['offense_id'],keep='last')
```

2.2.8 Checking for duplicates, missing values and other abnormalities, relationship table

```
[81]: df_rel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794157 entries, 0 to 794156
Data columns (total 3 columns):
#   Column              Non-Null Count  Dtype
---  -
0   victim_id           794157 non-null  int64
1   offender_id         794157 non-null  int64
2   relationship_name    791868 non-null  object
```

```
dtypes: int64(2), object(1)
memory usage: 18.2+ MB
```

```
[82]: empty_string_count(df_rel)
```

```
Column victim_id empty string count: 0
Column victim_id null values count: 0
*****
Column offender_id empty string count: 0
Column offender_id null values count: 0
*****
Column relationship_name empty string count: 0
Column relationship_name null values count: 2289
*****
Total number of records in the dataframe: 794157
```

```
[83]: df_rel['relationship_name'].value_counts()
```

```
[83]: Victim Was Stranger          168712
Relationship Unknown             133504
Victim Was Boyfriend/Girlfriend  101800
Victim Was Acquaintance          92034
Victim was Otherwise Known       77439
Victim Was Spouse                48593
Victim Was Offender              29886
Victim Was Child                 24853
Victim Was Friend                 19718
Victim Was Parent                16199
Victim Was Other Family Member   13803
Victim Was Sibling               13440
Victim Was Neighbor              9883
Victim was Ex-Spouse             8359
Victim Was Common-Law Spouse     8189
Homosexual Relationship           4639
Victim Was Stepchild             4326
Victim Was Child of Boyfriend or Girlfriend  3281
Victim Was In-law                2984
Victim Was Stepparent            2150
Victim Was Grandchild            2030
Victim was Employee              1562
Victim Was Grandparent           1471
Victim Was Stepsibling           1151
Victim was Employer              1065
Victim Was Babysittee            797
Name: relationship_name, dtype: int64
```

```
[84]: # Replacing NULL values in relationship_name to 'Relationship Unknown'
```

```
df_rel.loc[df_rel.relationship_name.isnull(), 'relationship_name'] =  
    ↳ 'Relationship Unknown'
```

```
[85]: # Checking for duplicates in offense_id column  
df=df_rel[df_rel.duplicated(subset=['victim_id','offender_id'],keep=False)].  
    ↳ sort_values(by='victim_id')  
df
```

```
[85]: Empty DataFrame  
Columns: [victim_id, offender_id, relationship_name]  
Index: []
```

2.3 Part III, combining the DataFrames

2.3.1 DFs Info

```
[86]: df_incident.info()  
  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 2819189 entries, 0 to 2819462  
Data columns (total 6 columns):  
#   Column          Dtype  
---  ---  
0   agency_id       int64  
1   incident_id      int64  
2   incident_hour    int64  
3   primary_county   object  
4   icpsr_zip        object  
5   timestamp        datetime64[ns]  
dtypes: datetime64[ns](1), int64(3), object(2)  
memory usage: 150.6+ MB
```

```
[87]: with open('data/pickled_dataframes/incident_clean.pickle', 'wb') as f:  
    pickle.dump(df_incident, f)
```

```
[88]: df_offense.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3201143 entries, 0 to 3201142  
Data columns (total 6 columns):  
#   Column          Dtype  
---  ---  
0   offense_id       int64  
1   incident_id      int64  
2   location_name     object  
3   offense_name      object  
4   crime_against     object  
5   offense_category_name object
```

```
dtypes: int64(2), object(4)
memory usage: 146.5+ MB
```

```
[89]: with open('data/pickled_dataframes/offense_clean.pickle', 'wb') as f:
      pickle.dump(df_offense, f)
```

```
[90]: df_offender.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3197991 entries, 0 to 3197990
Data columns (total 7 columns):
#   Column              Dtype
---  -
0   offender_id         int64
1   incident_id         int64
2   offender_age        object
3   offender_sex        object
4   offender_race       object
5   offender_age_group  object
6   offender_ethnicity  object
dtypes: int64(2), object(5)
memory usage: 170.8+ MB
```

```
[91]: with open('data/pickled_dataframes/offender_clean.pickle', 'wb') as f:
      pickle.dump(df_offender, f)
```

```
[92]: df_victim.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3229640 entries, 0 to 3229639
Data columns (total 9 columns):
#   Column              Dtype
---  -
0   victim_id          int64
1   incident_id         int64
2   victim_age         object
3   victim_sex         object
4   victim_resident_status object
5   victim_race        object
6   victim_age_group   object
7   victim_ethnicity   object
8   victim_type        object
dtypes: int64(2), object(7)
memory usage: 221.8+ MB
```

```
[93]: with open('data/pickled_dataframes/victim_clean.pickle', 'wb') as f:
      pickle.dump(df_victim, f)
```

```
[94]: df_weapon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 540940 entries, 0 to 551048
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   offense_id  540940 non-null  int64
1   weapon      540940 non-null  object
dtypes: int64(1), object(1)
memory usage: 12.4+ MB
```

```
[95]: with open('data/pickled_dataframes/weapon_clean.pickle', 'wb') as f:
      pickle.dump(df_weapon, f)
```

```
[96]: df_weapon.weapon.value_counts()
```

```
[96]: Non-automatic firearm    420917
      Other weapon            104428
      Unknown                  10189
      Unarmed                  2803
      Automatic firearm        2603
      Name: weapon, dtype: int64
```

```
[97]: df_bias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3201143 entries, 0 to 3201157
Data columns (total 2 columns):
#   Column      Dtype
---  ---
0   offense_id  int64
1   bias_name   object
dtypes: int64(1), object(1)
memory usage: 73.3+ MB
```

```
[98]: with open('data/pickled_dataframes/bias_clean.pickle', 'wb') as f:
      pickle.dump(df_bias, f)
```

```
[99]: df_rel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 794157 entries, 0 to 794156
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   victim_id    794157 non-null  int64
1   offender_id  794157 non-null  int64
2   relationship_name  794157 non-null  object
dtypes: int64(2), object(1)
memory usage: 18.2+ MB
```

```
[100]: with open('data/pickled_dataframes/rel_clean.pickle', 'wb') as f:
        pickle.dump(df_rel, f)
```

1. Offense, incident, bias and weapon DataFrames will be combined into one for the Times-series analysis
2. Offender, victim and relationship DataFrames will be set aside for the dashboard.

2.3.2 Combining Incident, Offense, Bias and Weapon DataFrames

```
[101]: df_full=df_offense.merge(df_incident, how='left', on='incident_id')
        df_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3201143 entries, 0 to 3201142
Data columns (total 11 columns):
#   Column                      Dtype
---  -
0   offense_id                  int64
1   incident_id                 int64
2   location_name               object
3   offense_name                object
4   crime_against               object
5   offense_category_name       object
6   agency_id                   int64
7   incident_hour               int64
8   primary_county              object
9   icpsr_zip                   object
10  timestamp                    datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(6)
memory usage: 293.1+ MB
```

```
[102]: df_full=df_full.merge(df_bias, how='left', on='offense_id')
        df_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3201143 entries, 0 to 3201142
Data columns (total 12 columns):
#   Column                      Dtype
---  -
0   offense_id                  int64
1   incident_id                 int64
2   location_name               object
3   offense_name                object
4   crime_against               object
5   offense_category_name       object
6   agency_id                   int64
7   incident_hour               int64
8   primary_county              object
9   icpsr_zip                   object
10  timestamp                    datetime64[ns]
```

```

11  bias_name          object
dtypes: datetime64[ns](1), int64(4), object(7)
memory usage: 317.5+ MB

```

```
[103]: df_full=df_full.merge(df_weapon, how='left', on='offense_id')
df_full.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 3201143 entries, 0 to 3201142
Data columns (total 13 columns):
#   Column              Dtype
---  -
0   offense_id          int64
1   incident_id          int64
2   location_name        object
3   offense_name         object
4   crime_against        object
5   offense_category_name object
6   agency_id            int64
7   incident_hour         int64
8   primary_county        object
9   icpsr_zip            object
10  timestamp             datetime64[ns]
11  bias_name            object
12  weapon                object
dtypes: datetime64[ns](1), int64(4), object(8)
memory usage: 341.9+ MB

```

```
[104]: empty_string_count(df_full)
```

```

Column offense_id empty string count: 0
Column offense_id null values count: 0
*****
Column incident_id empty string count: 0
Column incident_id null values count: 0
*****
Column location_name empty string count: 0
Column location_name null values count: 0
*****
Column offense_name empty string count: 0
Column offense_name null values count: 0
*****
Column crime_against empty string count: 0
Column crime_against null values count: 0
*****
Column offense_category_name empty string count: 0
Column offense_category_name null values count: 0
*****
Column agency_id empty string count: 0

```

```

Column agency_id null values count: 0
*****
Column incident_hour empty string count: 0
Column incident_hour null values count: 0
*****
Column primary_county empty string count: 0
Column primary_county null values count: 0
*****
Column icpsr_zip empty string count: 0
Column icpsr_zip null values count: 0
*****
Column timestamp empty string count: 0
Column timestamp null values count: 0
*****
Column bias_name empty string count: 0
Column bias_name null values count: 0
*****
Column weapon empty string count: 0
Column weapon null values count: 2660203
*****
Total number of records in the dataframe: 3201143

```

```
[105]: df_full.weapon.unique()
```

```
[105]: array(['Non-automatic firearm', nan, 'Other weapon', 'Unknown', 'Unarmed',
            'Automatic firearm'], dtype=object)
```

```
[106]: df=df_full[df_full.weapon.isnull()]
df.offense_category_name.unique()
```

```
[106]: array(['Larceny/Theft Offenses', 'Drug/Narcotic Offenses',
            'Fraud Offenses', 'Destruction/Damage/Vandalism of Property',
            'Burglary/Breaking & Entering', 'Assault Offenses', 'Sex Offenses',
            'Arson', 'Motor Vehicle Theft', 'Pornography/Obscene Material',
            'Counterfeiting/Forgery', 'Bribery', 'Stolen Property Offenses',
            'Prostitution Offenses', 'Embezzlement', 'Gambling Offenses',
            'Animal Cruelty'], dtype=object)
```

```
[107]: # Replacing NaN values in weapon column by 'NA'. Offenses associated with
↪ weapon NaN values seem
# to be offenses with no weapon necessary

df_full.loc[df_full.weapon.isnull(), 'weapon'] = 'NA'
```

```
[108]: df_full.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 3201143 entries, 0 to 3201142

```



```
Data columns (total 13 columns):
#   Column                               Dtype
---  -
0   offense_id                           int64
1   incident_id                           int64
2   location_name                         object
3   offense_name                          object
4   crime_against                         object
5   offense_category_name                 object
6   agency_id                             int64
7   incident_hour                         int64
8   primary_county                       object
9   icpsr_zip                             object
10  timestamp                             datetime64[ns]
11  bias_name                             object
12  weapon                                object
dtypes: datetime64[ns](1), int64(4), object(8)
memory usage: 341.9+ MB
```

```
[109]: with open('data/pickled_dataframes/df_full_clean.pickle', 'wb') as f:
        pickle.dump(df_full, f)
```

3 EXPLORE

3.1 EDA

3.1.1 General information about the data

```
[110]: print('There are {} records of offenses in Colorado between 2009 and 2019'.
        ↪format(len(df_full)))
```

There are 3201143 records of offenses in Colorado between 2009 and 2019

```
[111]: df_full.nunique()
```

```
[111]: offense_id           3201143
       incident_id        2819189
       location_name         46
       offense_name         51
       crime_against         4
       offense_category_name  23
       agency_id           249
       incident_hour        25
       primary_county       64
       icpsr_zip           195
       timestamp           4017
       bias_name            30
       weapon              6
```

```
dtype: int64
```

Plotting crime rate in different offense categories

```
[112]: freq='W'

df_x = df_full.groupby(['offense_category_name', pd.Grouper(key='timestamp',
    ↵
    ↪freq=freq)]['offense_category_name'].agg(['count']).reset_index()
df_x = df_x.sort_values(by=['timestamp', 'count'])
df_x
```

```
[112]:
```

	offense_category_name	timestamp	count
5845	Homicide Offenses	2009-01-04	1
8255	Pornography/Obscene Material	2009-01-04	1
1317	Bribery	2009-01-04	2
4690	Extortion/Blackmail	2009-01-04	2
8823	Prostitution Offenses	2009-01-04	2
...
4114	Drug/Narcotic Offenses	2020-01-05	120
8254	Motor Vehicle Theft	2020-01-05	126
1316	Assault Offenses	2020-01-05	163
3539	Destruction/Damage/Vandalism of Property	2020-01-05	242
7679	Larceny/Theft Offenses	2020-01-05	481

```
[11691 rows x 3 columns]
```

```
[113]: colors_dark24=px.colors.qualitative.Dark24
colors_dark24=colors_dark24[:-1]
crime_categories=['Assault Offenses', 'Larceny/Theft Offenses',
    'Drug/Narcotic Offenses', 'Fraud Offenses',
    'Destruction/Damage/Vandalism of Property',
    'Burglary/Breaking & Entering', 'Sex Offenses',
    'Arson', 'Motor Vehicle Theft', 'Kidnapping/Abduction',
    'Weapon Law Violations', 'Robbery',
    'Pornography/Obscene Material', 'Counterfeiting/Forgery',
    'Bribery', 'Stolen Property Offenses', 'Prostitution Offenses',
    'Homicide Offenses', 'Extortion/Blackmail',
    'Embezzlement', 'Gambling Offenses',
    'Human Trafficking', 'Animal Cruelty']

color_discrete_map_=dict(zip(crime_categories,colors_dark24))
```

```
[114]: fig1 = px.line(df_x, x='timestamp', y='count', color='offense_category_name',
    color_discrete_map=color_discrete_map_,
    labels={ "timestamp": "Date", "count": "Number of Offenses", ↵
    ↪"offense_category_name": "Offense Category"},
    title='Number of Offenses in Different Crime Categories',
```

```

template="plotly_dark"
    )

fig1.update_layout(width=1000,
                   height=800)

fig1.update_layout(
    xaxis=dict(
        #         rangeselector=dict(
        #             buttons=list([
        #                 dict(count=1,
        #                     step="month",
        #                     stepmode='backward'),
        #             ])),
        rangeslider=dict(
            visible=True
        ),
    )
)
fig1.show()

```

```

[115]: with open('images/pickled_figs/crime_cat.pickle', 'wb') as f:
        pickle.dump(fig1, f)

```

Number of Offenses in Weapon Categories

```

[116]: df_weapon = df_full.groupby(['weapon']).count().sort_values(['offense_id'],
    ↪ascending=False).reset_index()
df_weapon = df_weapon[df_weapon ['weapon'] != 'NA']

fig1 = px.bar(df_weapon, x='weapon', y='offense_id', color='weapon',
              color_discrete_sequence=px.colors.qualitative.Set1,
              labels={"weapon": "Weapon", "offense_id": "Number of Offenses"},
              title='Weapons Used in Offenses',
              template="plotly_dark"
            )

fig1.update_layout(width=1000,
                   height=700,
                   bargap=0.05)

fig1.show()

```

```

[117]: with open('images/pickled_figs/weapons.pickle', 'wb') as f:
        pickle.dump(fig1, f)

```

```

[118]: df_crime_against = df_full.groupby(['crime_against']).count().
    ↪sort_values(['offense_id'], ascending=False).reset_index()

```

```

df_crime_against = df_crime_against[df_crime_against['crime_against'] != 'Not a Crime']

fig1 = px.bar(df_crime_against, x='crime_against', y='offense_id',
              color='crime_against',
              color_discrete_sequence=px.colors.qualitative.Set1,
              labels={"crime_against": "Crime Against", "offense_id": "Number of Offenses"},
              title='Crime Against Offenses',
              template="plotly_dark"
              )

fig1.update_layout(width=1000,
                   height=700,
                   bargap=0.05)

fig1.show()

```

Crime rate per zip codes

```

[119]: df_zip = df_full.groupby(['icpsr_zip']).count().sort_values(['offense_id'],
                          ascending=False).reset_index()

fig1 = px.bar(df_zip[:15], x='icpsr_zip', y='offense_id', color='icpsr_zip',
              color_discrete_sequence=
              ["#800000", "#990000", "#A52A2A", "#B22222", "#DC143C", "#CD5C5C", "#FF0000", "#FF6347",
              "#FF7F50", "#FA8072", "#F08080", "#FFA074", "#FFCCE5", "#FFCCCC", "#FFE5CC"],
              labels={"icpsr_zip": "Zip Codes", "offense_id": "Number of Offenses"},
              title='Zip Codes Reporting the Highest Offense Numbers',
              template="plotly_dark"
              )

fig1.update_layout(width=1000,
                   height=700,
                   bargap=0.05)

fig1.show()

```

```

[120]: with open('images/pickled_figs/zips.pickle', 'wb') as f:
        pickle.dump(fig1, f)

```

Crime rate per county

```

[121]: df_county = df_full.groupby(['primary_county']).count().
        sort_values(['offense_id'], ascending=False).reset_index()

```

```

fig1 = px.bar(df_county[:15], y='primary_county', x='offense_id',
    color='primary_county', orientation='h',
    color_discrete_sequence=
    ["#800000", "#990000", "#A52A2A", "#B22222", "#DC143C", "#CD5C5C", "#FF0000", "#FF6347",
    "#FF7F50", "#FA8072", "#F08080", "#FFA074", "#FFCCE5", "#FFCCCC", "#FFE5CC"],
    labels={"primary_county": "County", "offense_id": "Number of
    Offenses"},
    title='Counties with the Highest Offense Numbers',
    template="plotly_dark"
)

fig1.update_layout(width=1000,
    height=700,
    bargap=0.05)

fig1.show()

```

```

[122]: with open('images/pickled_figs/counties.pickle', 'wb') as f:
    pickle.dump(fig1, f)

```

Crime rate over day hours

```

[123]: df_hour = df_full.groupby(['incident_hour']).count().
    sort_values(['offense_id'], ascending=False).reset_index()
df_hour = df_hour[df_hour['incident_hour'] != 25]

```

```

[124]: df_hour['hr_str'] = df_hour['incident_hour'].astype(str)

```

```

[125]: df_hour['hr_str'] = df_hour['hr_str'].map({
    '0': '12 AM', '1': '1 AM', '2': '2 AM', '3':
    '3 AM',
    '4': '4 AM', '5': '5 AM', '6': '6 AM', '7': '7 AM',
    '8': '8 AM', '9': '9 AM', '10': '10 AM', '11': '11
    AM',
    '12': '12 PM', '13': '1 PM', '14': '2 PM', '15': '3
    PM',
    '16': '4 PM', '17': '5 PM', '18': '6 PM', '19': '7
    PM',
    '20': '8 PM', '21': '9 PM', '22': '10 PM', '23': '11
    PM'})
df_hour.head()

```

```

[125]:
  incident_hour  offense_id  incident_id  location_name  offense_name  \
0              0      200701      200701      200701      200701
1             12     185895     185895     185895     185895
2             17     182505     182505     182505     182505
3             16     173320     173320     173320     173320

```

4	15	172326	172326	172326	172326
---	----	--------	--------	--------	--------

	crime_against	offense_category_name	agency_id	primary_county	icpsr_zip \
0	200701	200701	200701	200701	200701
1	185895	185895	185895	185895	185895
2	182505	182505	182505	182505	182505
3	173320	173320	173320	173320	173320
4	172326	172326	172326	172326	172326

	timestamp	bias_name	weapon	hr_str
0	200701	200701	200701	12 AM
1	185895	185895	185895	12 PM
2	182505	182505	182505	5 PM
3	173320	173320	173320	4 PM
4	172326	172326	172326	3 PM

```
[126]: fig1 = px.bar(df_hour, x='hr_str', y='offense_id', color='hr_str',
                    color_discrete_map={'12 AM': 'red', '1 AM': "#808080", '2 AM':
                    ↪ "#808080", '3 AM': "#808080",
                    '4 AM': "#808080", '5 AM': "#808080", '6 AM':
                    ↪ "#808080", '7 AM': "#808080",
                    '8 AM': 'red', '9 AM': "#808080", '10 AM':
                    ↪ "#808080", '11 AM': "#808080",
                    '12 PM': 'red', '1 PM': "#808080", '2 PM':
                    ↪ "#808080", '3 PM': "#808080",
                    '4 PM': "#808080", '5 PM': 'red', '6 PM':
                    ↪ "#808080", '7 PM': "#808080",
                    '8 PM': "#808080", '9 PM': "#808080", '10 PM':
                    ↪ "#808080", '11 PM': "#808080"},
                    category_orders={'hr_str': ['12 AM', '1 AM', '2 AM', '3 AM', '4 AM',
                    ↪ '5 AM', '6 AM', '7 AM', '8 AM', '9 AM',
                    '10 AM', '11 AM', '12 PM', '1 PM', '2 PM',
                    ↪ '3 PM', '4 PM', '5 PM', '6 PM', '7 PM',
                    '8 PM', '9 PM', '10 PM', '11 PM']},
                    labels={'hr_str': "Hour", "offense_id": "Number of Offenses"},
                    title='Most Dangerous Hours',
                    template="plotly_dark"
                    )

fig1.update_layout(width=1000,
                    height=700,
                    bargap=0.05)

fig1.show()
```

```
[127]: with open('images/pickled_figs/hours.pickle', 'wb') as f:
        pickle.dump(fig1, f)
```

Geography of crime

```
[128]: # fig2=map_choropleth_location(df_county, 'primary_county', 'County',  
    ↪ 'offense_id', 'Number of Offenses',  
    #                                CO_county_json, 'properties.name', 'Number of  
    ↪ Offenses per County')
```

```
[129]: # with open('images/pickled_figs/county_map.pickle', 'wb') as f:  
    #     pickle.dump(fig2, f)
```

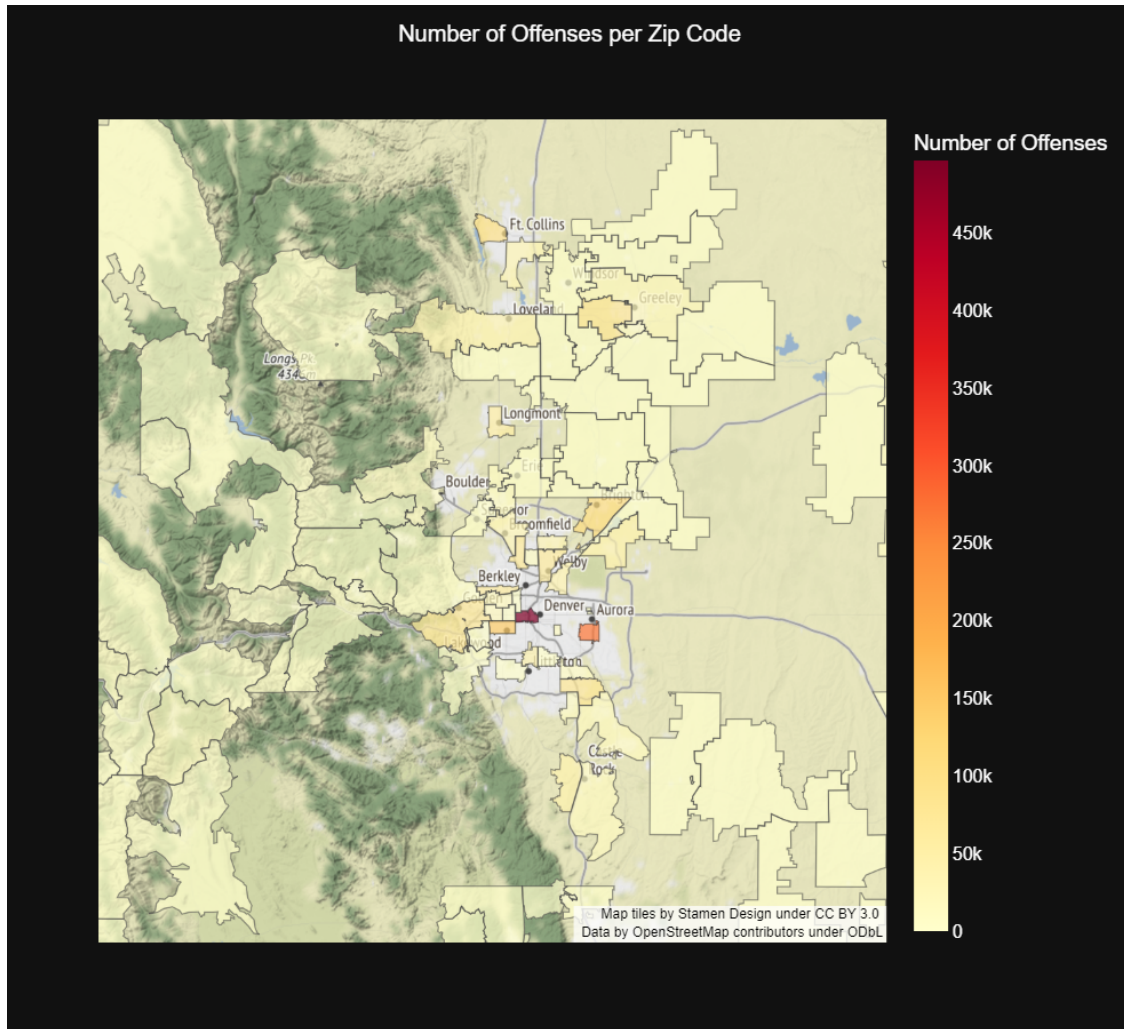
```
[130]: with open('images/pickled_figs/county_map.pickle', 'rb') as f:  
    fig2=pickle.load(f)  
    fig2.show()
```

```
[131]: # fig2=map_choropleth_location(df_zip, 'icpsr_zip', 'Zip code', 'offense_id',  
    ↪ 'Number of Offenses',  
    #                                CO_zip_json, 'properties.ZCTA5CE10', 'Number of  
    ↪ Offenses per Zip Code')
```

```
[132]: # with open('images/pickled_figs/zip_map.pickle', 'wb') as f:  
    #     pickle.dump(fig2, f)
```

```
[133]: # with open('images/pickled_figs/zip_map.pickle', 'rb') as f:  
    #     fig2=pickle.load(f)  
    #     fig2.show()
```

To reduce the size of the notebook the following map is a png image of the live plotly express choropleth plot. In order to generate the live plot uncomment the cell above and run it.



It takes ~1.5 minutes to run this notebook

General crime rate modeling is in [part III notebook](#). I decided to split them for better manageability.

[]: