capstone_prj_scrub_part2-px

July 18, 2021

1 Data Science Project

- Name: Author Name
- Email:

1.1 TABLE OF CONTENTS

2 INTRODUCTION

> Explain the point of your project and what question you are trying to answer with your modeling.

3 OBTAIN

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

3.1 Imports

[208]:

```
# 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
```

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

3.2 Data

3.2.1 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 descriptiopn are available.

The description of the main and reference tables is in data/README.md file. The agency 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.

The rest of the original data description is in description is in the notebook with the first part of data pre-processing.

3.2.2 Using an already created sqlite database

The notebook with database creation is here. The referenced database is in ***data/sqlite/db/production1 db***. It takes 2.5 minutes to run the database creation script.

4 SCRUB

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

The first part of the scrubbing process (working with sqlite3 database, production1) is in this notebook. It takes about 12 minutes to run the code in part1 notebook. The following code is using dataframes created in part I.

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`
3. `df_offender: data/pickled_dataframes/offender.pickle; main offender DF with demographic info`
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 used`
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 wi`

4.2 Part II, scrubbing the data in DataFrames

4.2.1 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]:
```python
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]:
```python
len(df_offense)
```

[6]: 3201143

[7]:
```python
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                    None          None      None
2     57702595     51264523      20     Male  White  Age in Years      None
3     57702596     51264524                    None          None      None
4     57702597     51264525      55     Male  White  Age in Years      None
```

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

[8]: 3197991

[9]:
```python
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                                          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
```

[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

4.2.2 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 (becase 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]:
```python
# 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]:
```python
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]:
```python
df_incident.loc[df_incident.primary_county == '', 'primary_county'] =␣
 ↪'Jefferson'
```

[27]:
```python
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
```

4.2.3 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

4.2.4 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 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 Inforcement victim type: {}'.format(len(df)))
df.head()
```

```
Number of records with empty string in resident_status_code and Individual or
Law Inforcement 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␣
 ↪Inforcement victim type: {}'.format(len(df)))
```

```
Number of records with empty string in age_num and Individual or Law Inforcement
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 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 Inforcement victim type: {}'.format(len(df)))
```

```
Number of records with empty string in resident_status_code and Individual or
Law Inforcement 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
```

4.2.5 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			None	None	None
2	57702595	51264523	20	Male	White	Age in Years	None

```
3     57702596     51264524                        None          None        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
```

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

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

[72]:
```
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
```

[73]:
```
# Checking for duplicates in offense_id column
df=df_weapon[df_weapon.duplicated(subset=['offense_id'],keep=False)].
 ↪sort_values(by='offense_id')
df
```

[73]:
```
        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.

[74]:
```
df_weapon=df_weapon.drop_duplicates(subset=['offense_id'],keep='last')
```

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

[75]:
```
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
```

[76]:
```
# Checking for duplicates in offense_id column
df=df_bias[df_bias.duplicated(subset=['offense_id'],keep=False)].
 ↪sort_values(by='offense_id')
df
```

[76]:

	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)
```

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.

[78]:
```
df_bias=df_bias.drop_duplicates(subset=['offense_id'],keep='last')
```

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

[79]:
```
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
```

[80]:
```
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
```

[81]:
```
df_rel['relationship_name'].value_counts()
```

[81]:
```
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
```

[82]:
```python
# Replacing NULL values in relationship_name to 'Relationship Unknown'
df_rel.loc[df_rel.relationship_name.isnull(), 'relationship_name'] =␣
↪'Relationship Unknown'
```

[83]:
```python
# 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
```

[83]:
```
Empty DataFrame
Columns: [victim_id, offender_id, relationship_name]
Index: []
```

4.3 Part III, combining the DataFrames

4.3.1 DFs Info

[84]:
```python
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
```

[85]:
```python
with open('data/pickled_dataframes/incident_clean.pickle', 'wb') as f:
    pickle.dump(df_incident, f)
```

[86]:
```python
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
```

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

[88]:
```python
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
```

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

[90]:
```
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
```

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

[92]:
```
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
```

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

[94]:
```
df_weapon.weapon.value_counts()
```

[94]:
```
Non-automatic firearm    420917
Other weapon             104428
Unknown                   10189
Unarmed                    2803
Automatic firearm          2603
Name: weapon, dtype: int64
```

[95]:
```
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
```

[96]:
```
with open('data/pickled_dataframes/bias_clean.pickle', 'wb') as f:
    pickle.dump(df_bias, f)
```

[97]:
```
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
```

[98]:
```
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.

4.3.2 Combining Incident, Offense, Bias and Weapon DataFrames

[99]:
```
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
```

[100]:
```
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
```

[101]:
```
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
```

[102]:
```
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
```

[103]:
```
df_full.weapon.unique()
```

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

[104]:
```
df=df_full[df_full.weapon.isnull()]
df.offense_category_name.unique()
```

[104]: 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)

[105]:
```
# 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'
```

[106]:
```
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
```

[107]:
```
with open('data/pickled_dataframes/df_full_clean.pickle', 'wb') as f:
    pickle.dump(df_full, f)
```

5 EXPLORE

5.1 EDA

5.1.1 General information about the data

[108]:
```
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
```

[109]:
```
df_full.nunique()
```

[109]:
```
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

[110]:
```
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
```

[110]:

	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]

[111]:
```
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))
```

[112]:
```
fig = 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"
             )

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

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

[113]:
```
with open('images/pickled_figs/crime_cat.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

Number of Offenses in Weapon Categories

[114]:
```
df_weapon = df_full.groupby(['weapon']).count().sort_values(['offense_id'],␣
↪ascending=False).reset_index()
df_weapon = df_weapon[df_weapon ['weapon'] != 'NA']


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

fig.update_layout(width=1000,
                  height=700,
                  bargap=0.05)
fig.show()
```

[115]:
```
with open('images/pickled_figs/weapons.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

Crime rate per zip codes

[116]:
```
df_zip = df_full.groupby(['icpsr_zip']).count().sort_values(['offense_id'],␣
↪ascending=False).reset_index()


fig = px.bar(df_zip[:15], x='icpsr_zip',  y='offense_id', color='icpsr_zip',
            labels={"icpsr_zip": "Zip Codes",  "offense_id": "Number of␣
↪Offenses"},
             title='Zip Codes with the Highest Offense Numbers',
template="plotly_dark"
             )

fig.update_layout(width=1000,
                  height=700,
                  bargap=0.05)
fig.show()
```

[117]:
```
with open('images/pickled_figs/zips.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

Crime rate per county

[118]:
```
df_county = df_full.groupby(['primary_county']).count().
↪sort_values(['offense_id'], ascending=False).reset_index()
```

```
fig = px.bar(df_county[:15], y='primary_county',  x='offense_id',␣
↪color='primary_county',  orientation='h',
            labels={"primary_county": "County",  "offense_id": "Number of␣
↪Offenses"},
             title='Counties with the Highest Offense Numbers',
template="plotly_dark"
             )

fig.update_layout(width=1000,
                  height=700,
                  bargap=0.05)
fig.show()
```

[119]:
```
with open('images/pickled_figs/counties.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

Crime rate over day hours

[120]:
```
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]

fig = px.bar(df_hour, x='incident_hour',  y='offense_id',
            labels={"incident_hour": "Hour (24hr format)",  "offense_id":␣
↪"Number of Offenses"},
              title='Most Dangerous Hours',
template="plotly_dark"
             )

fig.update_layout(width=1000,
                  height=700,
                  bargap=0.05)
fig.show()
```

[121]:
```
with open('images/pickled_figs/hours.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

[122]:
```
fig=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')
```

[123]:
```
fig=map_choropleth_location(df_county, 'primary_county', 'County',␣
↪'offense_id', 'Number of Offenses',
                            CO_county_json, 'properties.name', 'Number of␣
↪Offenses per County')
```

[124]:
```python
with open('images/pickled_figs/county_map.pickle', 'wb') as f:
    pickle.dump(fig, f)
```

5.1.2 Setting up the DataFrame to continue with Time-Series analysis

[125]:
```python
# Setting up timestamp index
df_full_ts_full=df_full.copy()
df_full_ts=df_full_ts_full.loc[df_full_ts_full.timestamp >'2015']
df_full_ts.set_index('timestamp', drop=True, inplace=True)
df_full_ts.head()
```

[125]:
```
            offense_id  incident_id   location_name  \
timestamp
2015-09-13    90865054     83230679  Residence/Home
2015-09-27    90865110     83229845  Residence/Home
2015-09-26    90865082     83229813   Other/Unknown
2015-09-21    90865081     83230696     Field/Woods
2015-09-26    90865077     83229806  Residence/Home

                                           offense_name crime_against  \
timestamp
2015-09-13                              Motor Vehicle Theft      Property
2015-09-27                    Burglary/Breaking & Entering      Property
2015-09-26  Theft of Motor Vehicle Parts or Accessories      Property
2015-09-21                              Motor Vehicle Theft      Property
2015-09-26                         Theft From Motor Vehicle      Property

                   offense_category_name  agency_id  incident_hour  \
timestamp
2015-09-13           Motor Vehicle Theft       1971             25
2015-09-27  Burglary/Breaking & Entering       1971             16
2015-09-26        Larceny/Theft Offenses       1971             25
2015-09-21           Motor Vehicle Theft       1971             25
2015-09-26        Larceny/Theft Offenses       1971             25

           primary_county icpsr_zip bias_name weapon
timestamp
2015-09-13     Kit Carson     80807      None     NA
2015-09-27     Kit Carson     80807      None     NA
2015-09-26     Kit Carson     80807      None     NA
2015-09-21     Kit Carson     80807      None     NA
2015-09-26     Kit Carson     80807      None     NA
```

[126]:
```python
len(df_full_ts_full)
```

[126]: 3201143

[127]:
```
len(df_full_ts)
```

[127]:
```
1588675
```

[128]:
```
df_=df_full_ts.groupby('offense_category_name')['offense_id'].nunique().
 ↪sort_values(ascending=False)
df_
```

[128]:
```
offense_category_name
Larceny/Theft Offenses                         537725
Assault Offenses                               216625
Destruction/Damage/Vandalism of Property       215875
Drug/Narcotic Offenses                         156490
Fraud Offenses                                 114644
Burglary/Breaking & Entering                   107629
Motor Vehicle Theft                             99299
Sex Offenses                                    29977
Weapon Law Violations                           27470
Counterfeiting/Forgery                          25613
Robbery                                         17782
Stolen Property Offenses                        11268
Kidnapping/Abduction                             9220
Arson                                            4657
Pornography/Obscene Material                     3256
Prostitution Offenses                            2536
Embezzlement                                     2372
Animal Cruelty                                   2137
Extortion/Blackmail                              2108
Homicide Offenses                                1105
Bribery                                           671
Human Trafficking                                 178
Gambling Offenses                                  38
Name: offense_id, dtype: int64
```

[129]:
```
pd.crosstab(index = df_full_ts['offense_name'], columns =␣
 ↪df_full_ts['offense_category_name'])[:10]
```

[129]:
```
offense_category_name                       Animal Cruelty  Arson  \
offense_name
Aggravated Assault                                       0      0
All Other Larceny                                        0      0
Animal Cruelty                                        2137      0
Arson                                                    0   4657
Assisting or Promoting Prostitution                      0      0
Betting/Wagering                                         0      0
Bribery                                                  0      0
Burglary/Breaking & Entering                             0      0
```

```
Counterfeiting/Forgery                                 0      0
Credit Card/Automated Teller Machine Fraud             0      0

offense_category_name                       Assault Offenses  Bribery  \
offense_name
Aggravated Assault                                     50396        0
All Other Larceny                                          0        0
Animal Cruelty                                             0        0
Arson                                                      0        0
Assisting or Promoting Prostitution                        0        0
Betting/Wagering                                           0        0
Bribery                                                    0      671
Burglary/Breaking & Entering                               0        0
Counterfeiting/Forgery                                     0        0
Credit Card/Automated Teller Machine Fraud                 0        0

offense_category_name                       Burglary/Breaking & Entering  \
offense_name
Aggravated Assault                                                     0
All Other Larceny                                                      0
Animal Cruelty                                                         0
Arson                                                                  0
Assisting or Promoting Prostitution                                    0
Betting/Wagering                                                       0
Bribery                                                                0
Burglary/Breaking & Entering                                      107629
Counterfeiting/Forgery                                                 0
Credit Card/Automated Teller Machine Fraud                             0

offense_category_name                       Counterfeiting/Forgery  \
offense_name
Aggravated Assault                                               0
All Other Larceny                                                0
Animal Cruelty                                                   0
Arson                                                            0
Assisting or Promoting Prostitution                              0
Betting/Wagering                                                 0
Bribery                                                          0
Burglary/Breaking & Entering                                     0
Counterfeiting/Forgery                                       25613
Credit Card/Automated Teller Machine Fraud                       0

offense_category_name                       Destruction/Damage/Vandalism of
Property  \
offense_name
Aggravated Assault
0
```

```
All Other Larceny
0
Animal Cruelty
0
Arson
0
Assisting or Promoting Prostitution
0
Betting/Wagering
0
Bribery
0
Burglary/Breaking & Entering
0
Counterfeiting/Forgery
0
Credit Card/Automated Teller Machine Fraud
0

offense_category_name                         Drug/Narcotic Offenses  \
offense_name
Aggravated Assault                                                 0
All Other Larceny                                                  0
Animal Cruelty                                                     0
Arson                                                              0
Assisting or Promoting Prostitution                                0
Betting/Wagering                                                   0
Bribery                                                            0
Burglary/Breaking & Entering                                       0
Counterfeiting/Forgery                                             0
Credit Card/Automated Teller Machine Fraud                         0

offense_category_name                         Embezzlement  Extortion/Blackmail  \
offense_name
Aggravated Assault                                       0                    0
All Other Larceny                                        0                    0
Animal Cruelty                                           0                    0
Arson                                                    0                    0
Assisting or Promoting Prostitution                      0                    0
Betting/Wagering                                         0                    0
Bribery                                                  0                    0
Burglary/Breaking & Entering                             0                    0
Counterfeiting/Forgery                                   0                    0
Credit Card/Automated Teller Machine Fraud               0                    0

offense_category_name                         …  Human Trafficking  \
offense_name                                  …
```

```
Aggravated Assault                          …                 0
All Other Larceny                           …                 0
Animal Cruelty                              …                 0
Arson                                       …                 0
Assisting or Promoting Prostitution         …                 0
Betting/Wagering                            …                 0
Bribery                                     …                 0
Burglary/Breaking & Entering                …                 0
Counterfeiting/Forgery                      …                 0
Credit Card/Automated Teller Machine Fraud  …                 0

offense_category_name                       Kidnapping/Abduction  \
offense_name
Aggravated Assault                                            0
All Other Larceny                                             0
Animal Cruelty                                                0
Arson                                                         0
Assisting or Promoting Prostitution                           0
Betting/Wagering                                              0
Bribery                                                       0
Burglary/Breaking & Entering                                  0
Counterfeiting/Forgery                                        0
Credit Card/Automated Teller Machine Fraud                    0

offense_category_name                       Larceny/Theft Offenses  \
offense_name
Aggravated Assault                                              0
All Other Larceny                                          185716
Animal Cruelty                                                  0
Arson                                                           0
Assisting or Promoting Prostitution                             0
Betting/Wagering                                                0
Bribery                                                         0
Burglary/Breaking & Entering                                    0
Counterfeiting/Forgery                                          0
Credit Card/Automated Teller Machine Fraud                      0

offense_category_name                       Motor Vehicle Theft  \
offense_name
Aggravated Assault                                           0
All Other Larceny                                            0
Animal Cruelty                                               0
Arson                                                        0
Assisting or Promoting Prostitution                          0
Betting/Wagering                                             0
Bribery                                                      0
Burglary/Breaking & Entering                                 0
```

```
Counterfeiting/Forgery                                        0
Credit Card/Automated Teller Machine Fraud                    0

offense_category_name                        Pornography/Obscene Material  \
offense_name
Aggravated Assault                                                      0
All Other Larceny                                                       0
Animal Cruelty                                                          0
Arson                                                                   0
Assisting or Promoting Prostitution                                     0
Betting/Wagering                                                        0
Bribery                                                                 0
Burglary/Breaking & Entering                                            0
Counterfeiting/Forgery                                                  0
Credit Card/Automated Teller Machine Fraud                              0

offense_category_name                        Prostitution Offenses  Robbery  \
offense_name
Aggravated Assault                                               0        0
All Other Larceny                                                0        0
Animal Cruelty                                                   0        0
Arson                                                            0        0
Assisting or Promoting Prostitution                            561        0
Betting/Wagering                                                 0        0
Bribery                                                          0        0
Burglary/Breaking & Entering                                     0        0
Counterfeiting/Forgery                                           0        0
Credit Card/Automated Teller Machine Fraud                       0        0

offense_category_name                        Sex Offenses  \
offense_name
Aggravated Assault                                      0
All Other Larceny                                       0
Animal Cruelty                                          0
Arson                                                   0
Assisting or Promoting Prostitution                     0
Betting/Wagering                                        0
Bribery                                                 0
Burglary/Breaking & Entering                            0
Counterfeiting/Forgery                                  0
Credit Card/Automated Teller Machine Fraud              0

offense_category_name                        Stolen Property Offenses  \
offense_name
Aggravated Assault                                                  0
All Other Larceny                                                   0
Animal Cruelty                                                      0
```

```
Arson                                                           0
Assisting or Promoting Prostitution                             0
Betting/Wagering                                                0
Bribery                                                         0
Burglary/Breaking & Entering                                    0
Counterfeiting/Forgery                                          0
Credit Card/Automated Teller Machine Fraud                      0

offense_category_name                       Weapon Law Violations
offense_name
Aggravated Assault                                            0
All Other Larceny                                             0
Animal Cruelty                                                0
Arson                                                         0
Assisting or Promoting Prostitution                           0
Betting/Wagering                                              0
Bribery                                                       0
Burglary/Breaking & Entering                                  0
Counterfeiting/Forgery                                        0
Credit Card/Automated Teller Machine Fraud                    0

[10 rows x 23 columns]
```

[130]:
```
TS_crime_category=create_ts_dict('offense_category_name', df_full_ts)

TS_crime_against=create_ts_dict('crime_against', df_full_ts)

TS_crime_location=create_ts_dict('location_name', df_full_ts)
```

[132]:
```
with open('data/pickled_ts/TS_crime_category.pickle', 'wb') as f:
    pickle.dump(TS_crime_category, f)

with open('data/pickled_ts/TS_crime_against.pickle', 'wb') as f:
    pickle.dump(TS_crime_against, f)

with open('data/pickled_ts/TS_crime_location.pickle', 'wb') as f:
    pickle.dump(TS_crime_location, f)
```

[133]:
```
TS_crime_category.keys()
```

[133]:
```
dict_keys(['Motor Vehicle Theft', 'Burglary/Breaking & Entering', 'Larceny/Theft
Offenses', 'Fraud Offenses', 'Counterfeiting/Forgery', 'Assault Offenses',
'Destruction/Damage/Vandalism of Property', 'Arson', 'Drug/Narcotic Offenses',
'Weapon Law Violations', 'Sex Offenses', 'Stolen Property Offenses',
'Kidnapping/Abduction', 'Robbery', 'Extortion/Blackmail', 'Pornography/Obscene
Material', 'Prostitution Offenses', 'Bribery', 'Embezzlement', 'Homicide
Offenses', 'Human Trafficking', 'Gambling Offenses', 'Animal Cruelty'])
```

[134]:
```
df_crime_against=pd.concat(TS_crime_against,axis=1)
df_crime_against.loc[(df_crime_against['Not a Crime'].isna()),'Not a Crime']=0
df_crime_against=df_crime_against.astype({'Not a Crime': 'int64'})
df_crime_against.head()
```

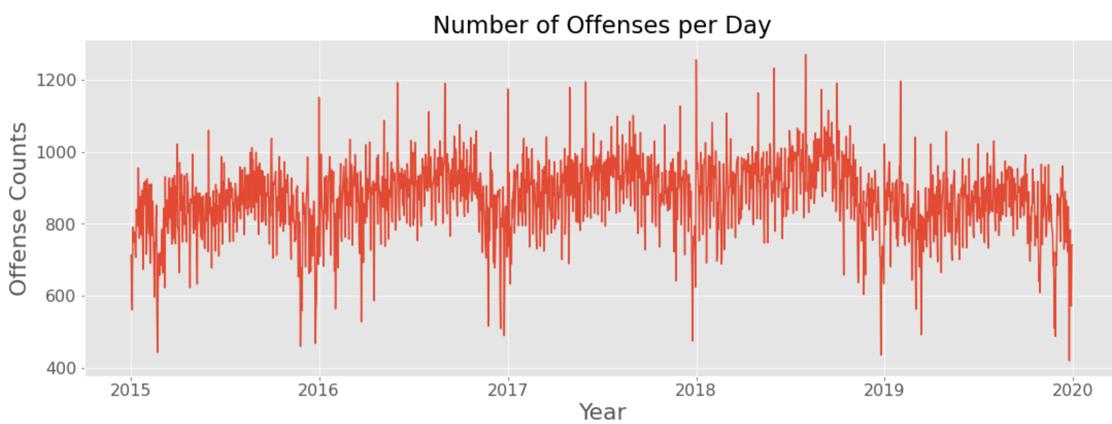
[134]:

	Property	Person	Society	Not a Crime
timestamp				
2015-01-04	1369	326	190	0
2015-01-11	3954	779	548	0
2015-01-18	4288	839	711	1
2015-01-25	4040	792	761	0
2015-02-01	4331	871	690	1

5.1.3 Exploring time-series plots

[135]:
```
# Creating a time-series
ts=df_full_ts.resample('D').count()['offense_id']
```

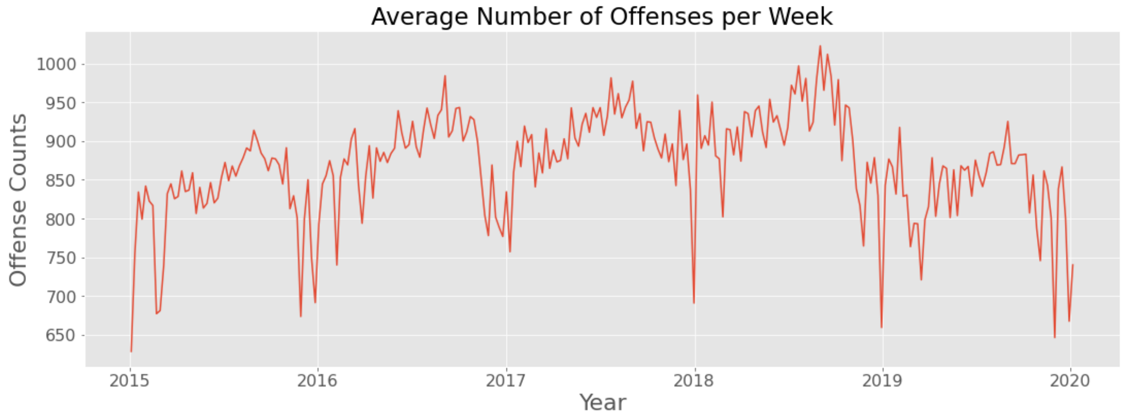
[136]:
```
with plt.style.context('ggplot'):
    fig, ax = plt.subplots(figsize=(18,6))

    ax.plot(ts.index, ts.values)
    ax.set_title('Number of Offenses per Day', fontsize=23);
    ax.set_ylabel('Offense Counts', fontsize=22);
    ax.set_xlabel('Year', fontsize=22);
    ax.tick_params(axis='y', labelsize=16)
    ax.tick_params(axis='x', labelsize=16)
    plt.show()
```



[137]:
```
ts_weekly=ts.resample('W').mean()
```

[138]:
```
with open('data/pickled_ts/ts_weekly.pickle', 'wb') as f:
    pickle.dump(ts_weekly, f)
```

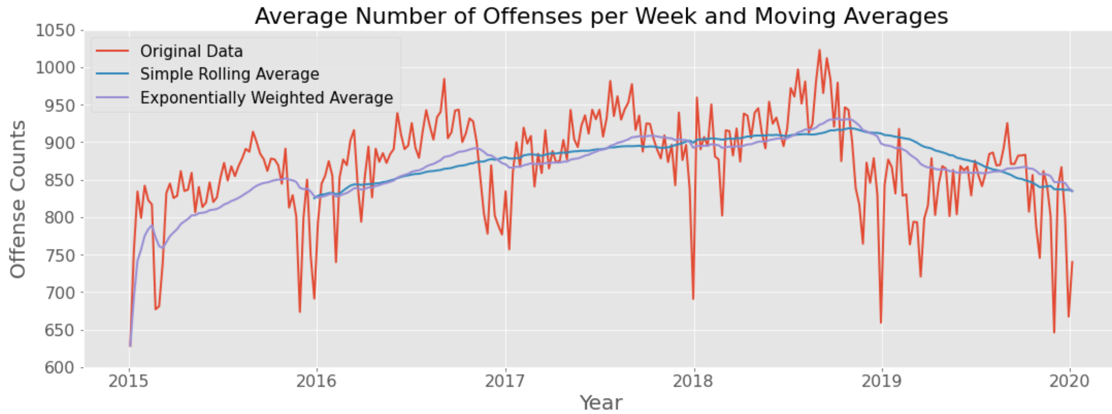
[139]:
```
with plt.style.context('ggplot'):
    fig, ax = plt.subplots(figsize=(18,6))

    ax.plot(ts_weekly.index, ts_weekly.values)
    ax.set_title('Average Number of Offenses per Week', fontsize=23);
    ax.set_ylabel('Offense Counts', fontsize=22);
    ax.set_xlabel('Year', fontsize=22);
    ax.tick_params(axis='y', labelsize=16)
    ax.tick_params(axis='x', labelsize=16)
    plt.show()
```



[140]:
```
ts_ma=ts_weekly.rolling(52).mean()
```

[141]:
```
ts_ewm=ts_weekly.ewm(span=52).mean()
```

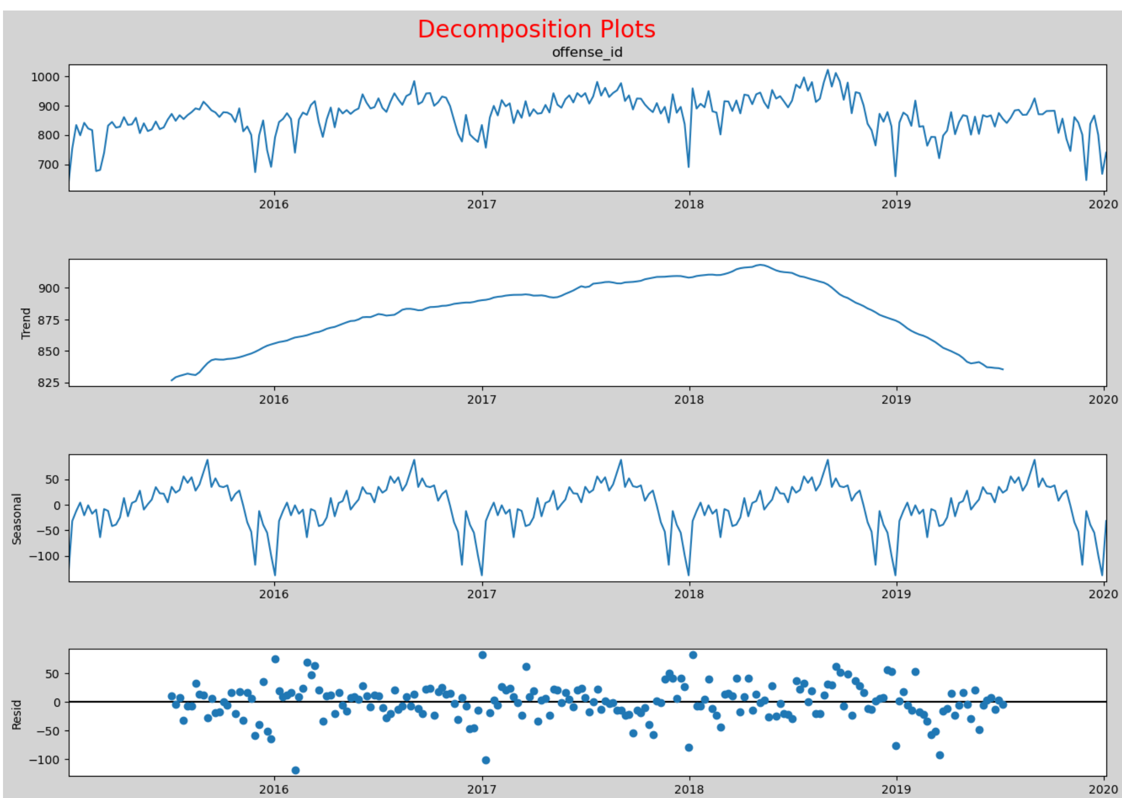
[142]:
```
fig=display_figure_w_TSs(ts_weekly, ts_ma, 'Original Data', 'Simple Rolling␣
↪Average',
                         'Average Number of Offenses per Week and Moving Averages',␣
↪n=3,
                         ts3=ts_ewm, ts4=None, label3='Exponentially Weighted␣
↪Average', label4=None)
```



EWA displays a clear upward trend with a seasonality while SRA does not pick up the seasonal fluctuation tendency. The seasonality is quite pronounced and is of an additive nature. The problematic range of dates is a period from the late 2018 till the end of 2019 when the trend changes to a downward trend. Unfortunately, cutting off a test set with the latest dates will make it impossible to predict the overall trend correctly.

[143]:
```
decomposing(ts_weekly)
```

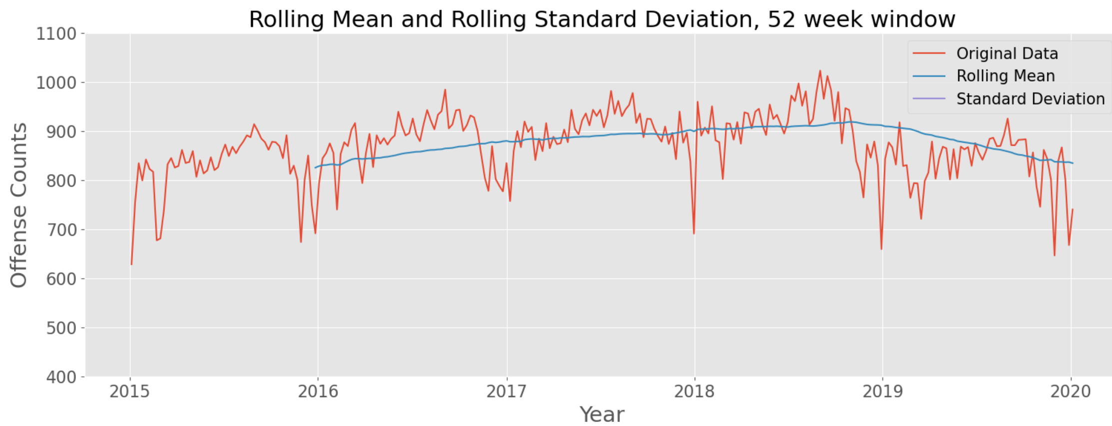
```
<Figure size 640x480 with 0 Axes>
```



The time series displays a clear trend along with seasonal fluctuations. Seasonality is comparable with the overall trend values (>10%).

5.1.4 Testing for Stationarity

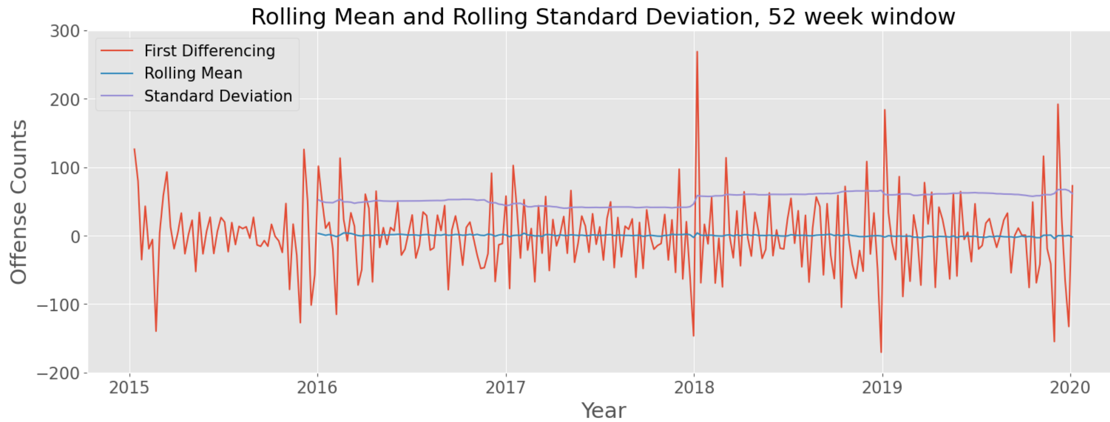
[148]:
```
check_stationarity(ts_weekly, 'Original Data', min_=400, max_=1100)
```



[148]:
```
                              T_value   P_value  Lags  Observations  \
Dickey-Fuller test results -3.224157  0.018628     3           258

                           Critical value, 1%  Critical value, 5%  \
Dickey-Fuller test results           -3.455953           -2.872809

                           Critical value, 10%  Stationary?
Dickey-Fuller test results            -2.572775         True
```

The time-series is more or less **stationary**, p-value is 0.02 (<0.05). Visually it is not very stationary, the trend is somewhat visible. Since critical value -3.22 > -3.46, but < -2.87 (t-values at 1% and 5% confidence intervals), null hypothesis is rejected. However, the TS might benefit from stationarization

[149]:
```
ts_diff1=ts_weekly.diff().dropna()

check_stationarity(ts_diff1, 'First Differencing', min_=-200, max_=300)
```



[149]:

```
                                T_value   P_value  Lags  Observations  \
Dickey-Fuller test results -4.872453  0.000039    12           248

                            Critical value, 1%  Critical value, 5%  \
Dickey-Fuller test results           -3.456996           -2.873266

                            Critical value, 10%  Stationary?
Dickey-Fuller test results            -2.573019         True
```

The first differenceing time-series is **stationary**, p-value is 3.9e-5 (well below 0.05). Also the critical value -4.87 < -3.46,-2.87 (t-values at 1% and 5% confidence intervals); null hypothesis is rejected.

[150]:

```
decomposing(ts_diff1)
```

```
<Figure size 640x480 with 0 Axes>
```

The first differencing time-series decomposition displays clear seasonality.

6 MODEL

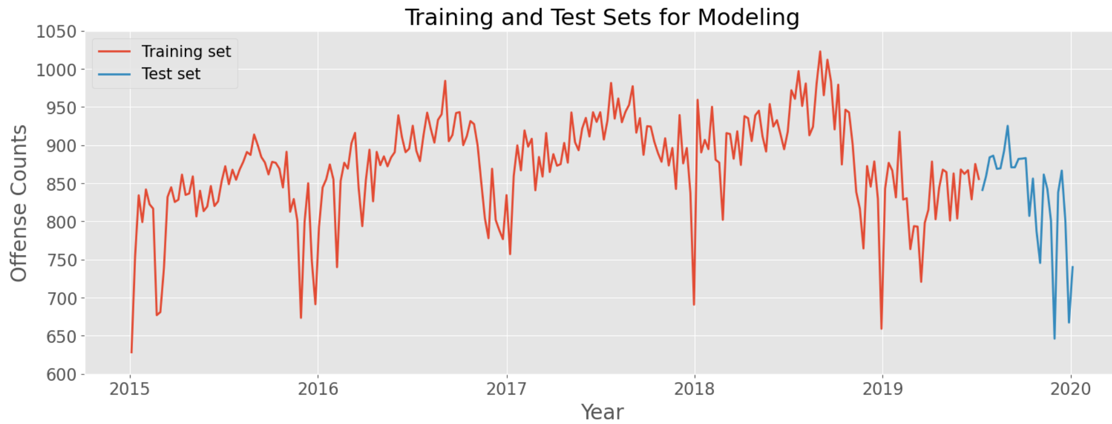
6.1 General Crime Rate Modeling

6.1.1 Splitting into a training and a test sets

I am cutting off a ~10% tail of my data to create a test set because I want the downswing of the data in the last year to be included in the training dataset

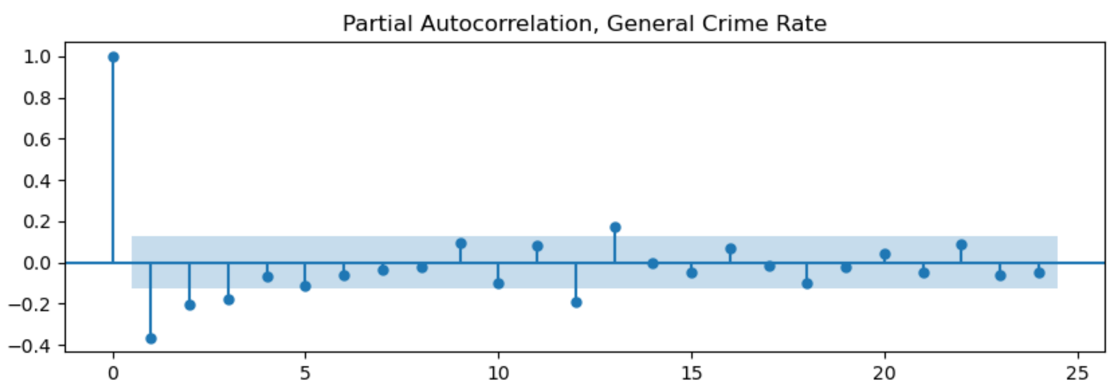
[151]:
```
train_size = round(len(ts_weekly) * 0.90)
ts_train, ts_test = ts_weekly[:train_size], ts_weekly[train_size:]
print('Observations: %d weeks' % (len(ts_weekly)))
print('Training Observations: %d weeks' % (len(ts_train)))
print('Testing Observations: %d weeks' % (len(ts_test)))

fig=display_figure_w_TSs(ts_train, ts_test, 'Training set', 'Test set',␣
 ↪'Training and Test Sets for Modeling')
```

```
Observations: 262 weeks
Training Observations: 236 weeks
Testing Observations: 26 weeks
```



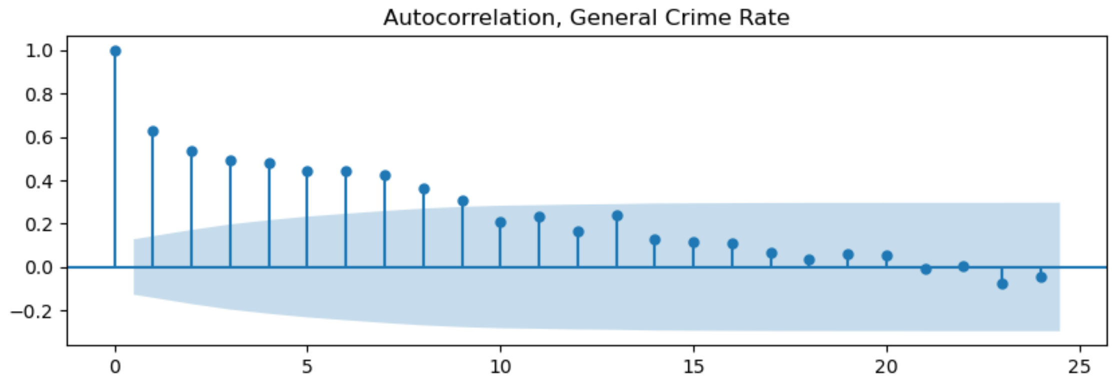
6.1.2 Partial Autocorrelation and Autocorrelation Functions

[152]:
```
matplotlib.rc_file_defaults()
plt.rc("figure", figsize=(10,3))
plot_pacf(ts_train.diff().dropna(), title='Partial Autocorrelation, General␣
↪Crime Rate');
```



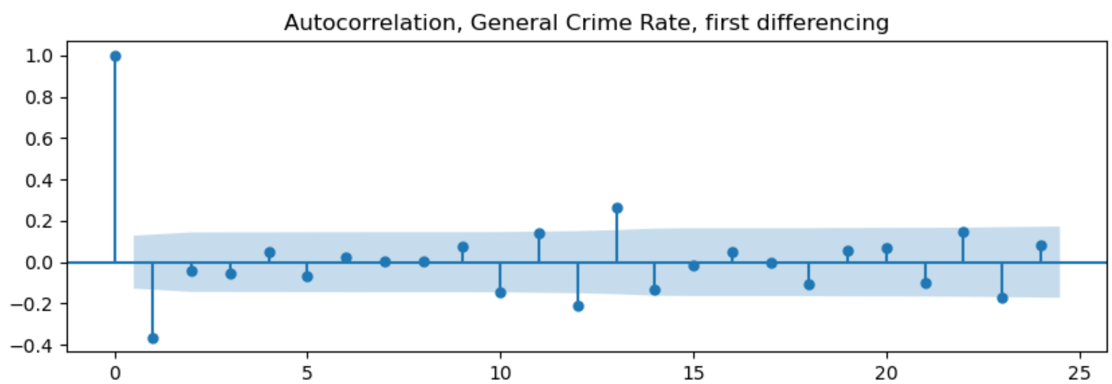
Partial autocorrelation function of the first ts differencing indicates the importance of the first 3 lags.

[153]:
```
matplotlib.rc_file_defaults()
plt.rc("figure", figsize=(10,3))
plot_acf(ts_train, title='Autocorrelation, General Crime Rate');
```



The ACF shows a long persistent autocorrelation up to the 9th lag. That is a strong indicator that the differencing should be taken to stationarize the TS.

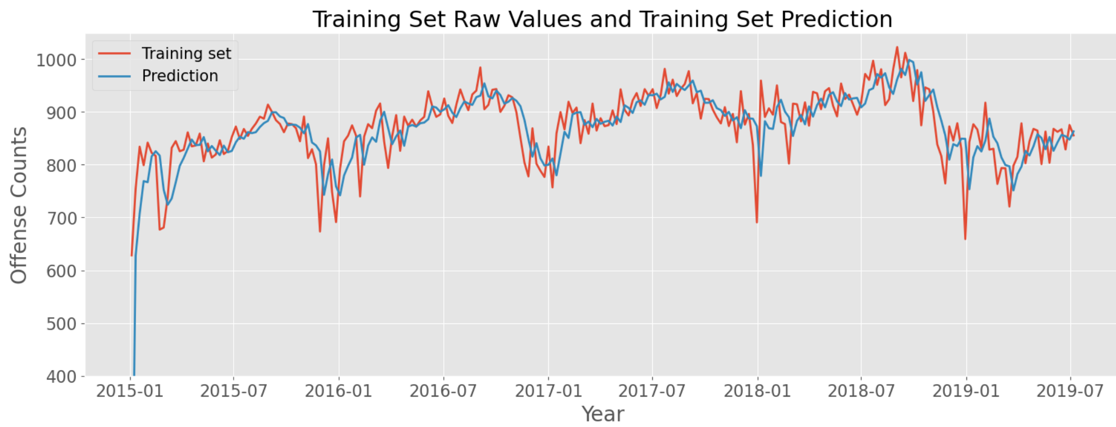
[154]:
```
matplotlib.rc_file_defaults()
plt.rc("figure", figsize=(10,3))
plot_acf(ts_train.diff().dropna(), title='Autocorrelation, General Crime Rate,␣
↪first differencing');
```



The ACF of the differenced series displays the sharp cut-off and the negative lag1 correlation and therefore one MA term could be added to the model.

6.1.3 Baseline Model

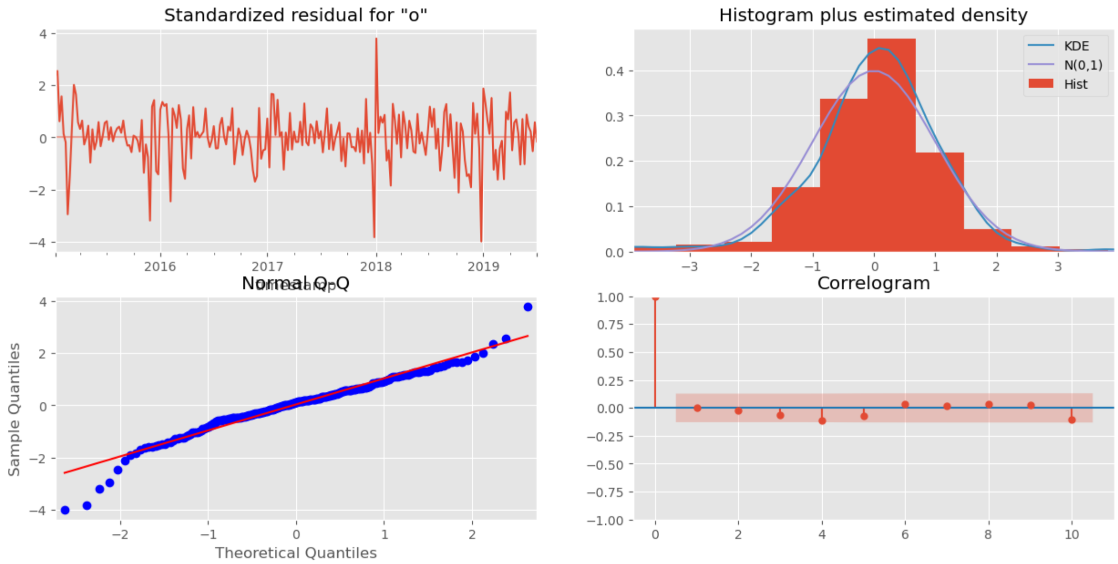
[155]:
```
arima_1=ARIMA(ts_train, order=(3,1,0)).fit()
y_hat_train=arima_1.predict(typ='levels')


fig=display_figure_w_TSs(ts_train, y_hat_train, 'Training set', 'Prediction',
                        'Training Set Raw Values and Training Set Prediction',␣
 ↪min_=400)
```



[156]:
```
diagnostics(arima_1)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:             offense_id   No. Observations:                  236
Model:                 ARIMA(3, 1, 0)   Log Likelihood               -1240.793
Date:                Sun, 18 Jul 2021   AIC                           2489.586
Time:                        19:16:03   BIC                           2503.424
Sample:                    01-04-2015   HQIC                          2495.165
                         - 07-07-2019
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5013      0.046    -10.995      0.000      -0.591      -0.412
ar.L2         -0.3149      0.063     -4.999      0.000      -0.438      -0.191
ar.L3         -0.1955      0.059     -3.317      0.001      -0.311      -0.080
sigma2      2253.0732    148.340     15.189      0.000    1962.332    2543.815
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                67.77
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.99   Skew:                            -0.53
```

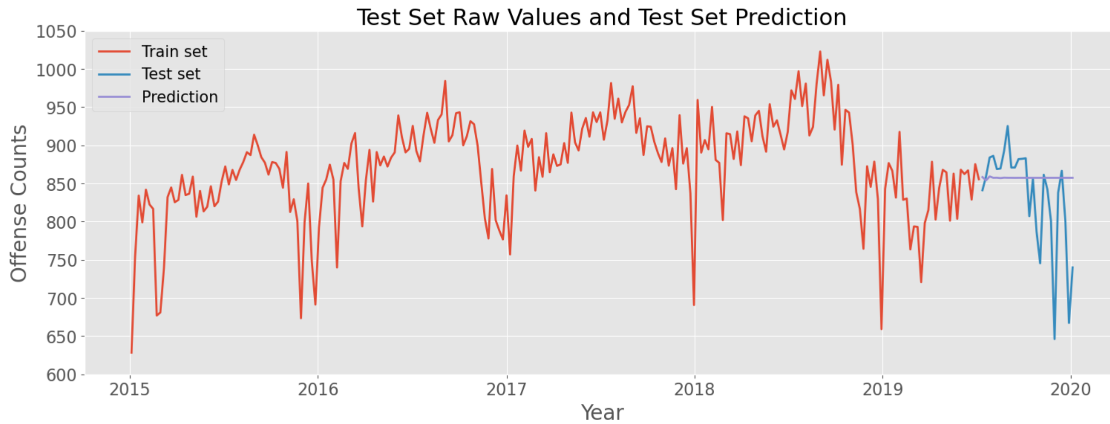
```
Prob(H) (two-sided):                  0.95   Kurtosis:                    5.41
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```



Testing the model

[157]:
```
y_hat_test=arima_1.predict(start=ts_test.index[0], end=ts_test.index[-1],␣
 ↪typ='levels')

rmse = np.sqrt(mean_squared_error(ts_test, y_hat_test))
print('RMSE of the Baseline model is {}'.format(round(rmse,2)))

fig=display_figure_w_TSs(ts_train, ts_test, 'Train set', 'Test set', 'Test Set␣
 ↪Raw Values and Test Set Prediction',
                         n=3, ts3=y_hat_test,
                         label3='Prediction')
```

```
RMSE of the Baseline model is 71.21
```



The ARIMA model above is our Baseline model

6.1.4 SARIMAX models

No exogenous regressors It's a manual grid search section. I tried several combinations of pdq/PDQs and it seems that the most appropriate tryout ranges for MA terms and for AR terms in both trend and seasonal parts of the models are 0-1 and 0-3 respectively.

[158]:
```
p=range(0,4)
q=range(0,2)
pdq=list(itertools.product(p,[1],q))

P=range(0,4)
Q=range(0,2)
seasonal_pdq=[(x[0], x[1], x[2], 52) for x in list(itertools.product(P,[1],Q))]

for i in pdq:
    for s in seasonal_pdq:
        print('SARIMAX combination: {}x{}'.format(i,s))
```

```
SARIMAX combination: (0, 1, 0)x(0, 1, 0, 52)
SARIMAX combination: (0, 1, 0)x(0, 1, 1, 52)
SARIMAX combination: (0, 1, 0)x(1, 1, 0, 52)
SARIMAX combination: (0, 1, 0)x(1, 1, 1, 52)
SARIMAX combination: (0, 1, 0)x(2, 1, 0, 52)
SARIMAX combination: (0, 1, 0)x(2, 1, 1, 52)
SARIMAX combination: (0, 1, 0)x(3, 1, 0, 52)
SARIMAX combination: (0, 1, 0)x(3, 1, 1, 52)
SARIMAX combination: (0, 1, 1)x(0, 1, 0, 52)
SARIMAX combination: (0, 1, 1)x(0, 1, 1, 52)
SARIMAX combination: (0, 1, 1)x(1, 1, 0, 52)
SARIMAX combination: (0, 1, 1)x(1, 1, 1, 52)
SARIMAX combination: (0, 1, 1)x(2, 1, 0, 52)
```

```
SARIMAX combination: (0, 1, 1)x(2, 1, 1, 52)
SARIMAX combination: (0, 1, 1)x(3, 1, 0, 52)
SARIMAX combination: (0, 1, 1)x(3, 1, 1, 52)
SARIMAX combination: (1, 1, 0)x(0, 1, 0, 52)
SARIMAX combination: (1, 1, 0)x(0, 1, 1, 52)
SARIMAX combination: (1, 1, 0)x(1, 1, 0, 52)
SARIMAX combination: (1, 1, 0)x(1, 1, 1, 52)
SARIMAX combination: (1, 1, 0)x(2, 1, 0, 52)
SARIMAX combination: (1, 1, 0)x(2, 1, 1, 52)
SARIMAX combination: (1, 1, 0)x(3, 1, 0, 52)
SARIMAX combination: (1, 1, 0)x(3, 1, 1, 52)
SARIMAX combination: (1, 1, 1)x(0, 1, 0, 52)
SARIMAX combination: (1, 1, 1)x(0, 1, 1, 52)
SARIMAX combination: (1, 1, 1)x(1, 1, 0, 52)
SARIMAX combination: (1, 1, 1)x(1, 1, 1, 52)
SARIMAX combination: (1, 1, 1)x(2, 1, 0, 52)
SARIMAX combination: (1, 1, 1)x(2, 1, 1, 52)
SARIMAX combination: (1, 1, 1)x(3, 1, 0, 52)
SARIMAX combination: (1, 1, 1)x(3, 1, 1, 52)
SARIMAX combination: (2, 1, 0)x(0, 1, 0, 52)
SARIMAX combination: (2, 1, 0)x(0, 1, 1, 52)
SARIMAX combination: (2, 1, 0)x(1, 1, 0, 52)
SARIMAX combination: (2, 1, 0)x(1, 1, 1, 52)
SARIMAX combination: (2, 1, 0)x(2, 1, 0, 52)
SARIMAX combination: (2, 1, 0)x(2, 1, 1, 52)
SARIMAX combination: (2, 1, 0)x(3, 1, 0, 52)
SARIMAX combination: (2, 1, 0)x(3, 1, 1, 52)
SARIMAX combination: (2, 1, 1)x(0, 1, 0, 52)
SARIMAX combination: (2, 1, 1)x(0, 1, 1, 52)
SARIMAX combination: (2, 1, 1)x(1, 1, 0, 52)
SARIMAX combination: (2, 1, 1)x(1, 1, 1, 52)
SARIMAX combination: (2, 1, 1)x(2, 1, 0, 52)
SARIMAX combination: (2, 1, 1)x(2, 1, 1, 52)
SARIMAX combination: (2, 1, 1)x(3, 1, 0, 52)
SARIMAX combination: (2, 1, 1)x(3, 1, 1, 52)
SARIMAX combination: (3, 1, 0)x(0, 1, 0, 52)
SARIMAX combination: (3, 1, 0)x(0, 1, 1, 52)
SARIMAX combination: (3, 1, 0)x(1, 1, 0, 52)
SARIMAX combination: (3, 1, 0)x(1, 1, 1, 52)
SARIMAX combination: (3, 1, 0)x(2, 1, 0, 52)
SARIMAX combination: (3, 1, 0)x(2, 1, 1, 52)
SARIMAX combination: (3, 1, 0)x(3, 1, 0, 52)
SARIMAX combination: (3, 1, 0)x(3, 1, 1, 52)
SARIMAX combination: (3, 1, 1)x(0, 1, 0, 52)
SARIMAX combination: (3, 1, 1)x(0, 1, 1, 52)
SARIMAX combination: (3, 1, 1)x(1, 1, 0, 52)
SARIMAX combination: (3, 1, 1)x(1, 1, 1, 52)
SARIMAX combination: (3, 1, 1)x(2, 1, 0, 52)
```

```
SARIMAX combination: (3, 1, 1)x(2, 1, 1, 52)
SARIMAX combination: (3, 1, 1)x(3, 1, 0, 52)
SARIMAX combination: (3, 1, 1)x(3, 1, 1, 52)
```

[ ]:
```
# for param in pdq:
#     for param_seasonal in seasonal_pdq:
#         try:
#             sarimax_mod=SARIMAX(ts_train,
#                                 order=param,
#                                 seasonal_order=param_seasonal,
#                                 enforce_invertibility=False)
#             results=sarimax_mod.fit()
#             print('ARIMA{}x{}-AIC:{}:'.format(param, param_seasonal,results.
↪aic))
#         except:
#             print('Error!')
#             continue
```

ARIMA(3, 1, 0)x(3, 1, 0, 52)-AIC:256.74: is our best model. it took 55 minutes to complete this search. Therefore I am commenting out this snippet.

[ ]:
```
# sarimax_mod1=SARIMAX(ts_train,
#                      order=(3, 1, 0),
#                      seasonal_order=(3, 1, 0, 52),
#                      enforce_invertibility=False).fit()
```

The model above took 44 seconds to fit but just in case it is pickled to be used forward.

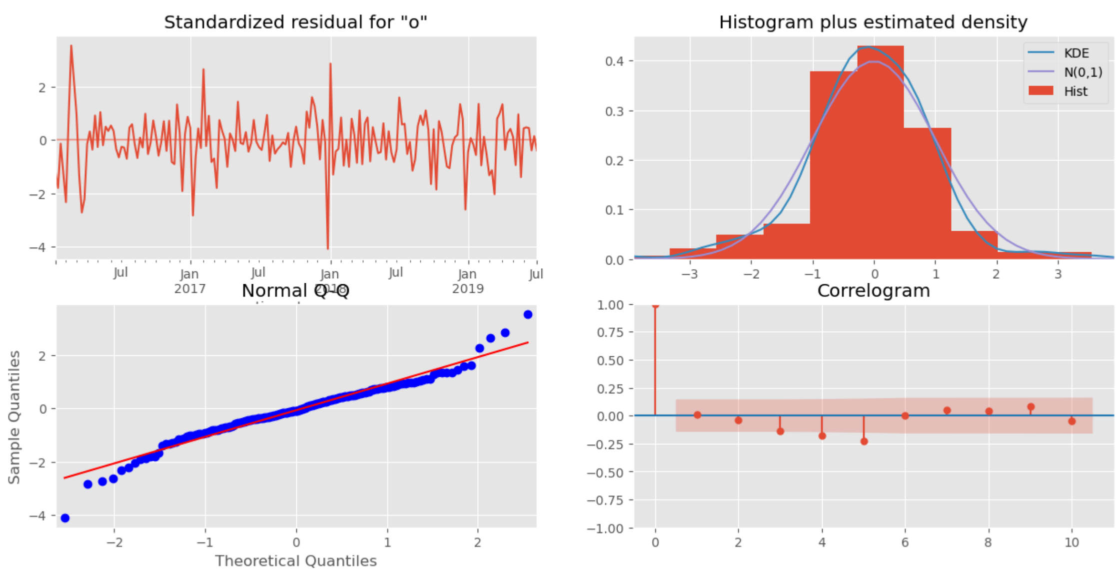
[ ]:
```
# with open('data/pickled_models/sarimax_mod1.pickle', 'wb') as f:
#     pickle.dump(sarimax_mod1, f)
```

[159]:
```
with open('data/pickled_models/sarimax_mod1.pickle', 'rb') as f:
    sarimax_mod1=pickle.load(f)
```

[160]:
```
diagnostics(sarimax_mod1)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                          offense_id   No. Observations:                  236
Model:             SARIMAX(3, 1, 0)x(3, 1, 0, 52)   Log Likelihood                -971.969
Date:                             Sun, 18 Jul 2021   AIC                           1957.937
Time:                                     19:16:20   BIC                           1980.403
Sample:                                 01-04-2015   HQIC                          1967.044
                                      - 07-07-2019
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
ar.L1         -0.6356      0.056    -11.337      0.000      -0.745      -0.526
ar.L2         -0.3886      0.066     -5.877      0.000      -0.518      -0.259
ar.L3         -0.1715      0.075     -2.292      0.022      -0.318      -0.025
ar.S.L52      -0.6286      0.092     -6.800      0.000      -0.810      -0.447
ar.S.L104     -0.4472      0.151     -2.960      0.003      -0.743      -0.151
ar.S.L156     -0.2381      0.165     -1.439      0.150      -0.562       0.086
sigma2      2033.5387    269.943      7.533      0.000    1504.460    2562.617
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                39.08
Prob(Q):                              0.86   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                            -0.24
Prob(H) (two-sided):                  0.06   Kurtosis:                         5.21
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```

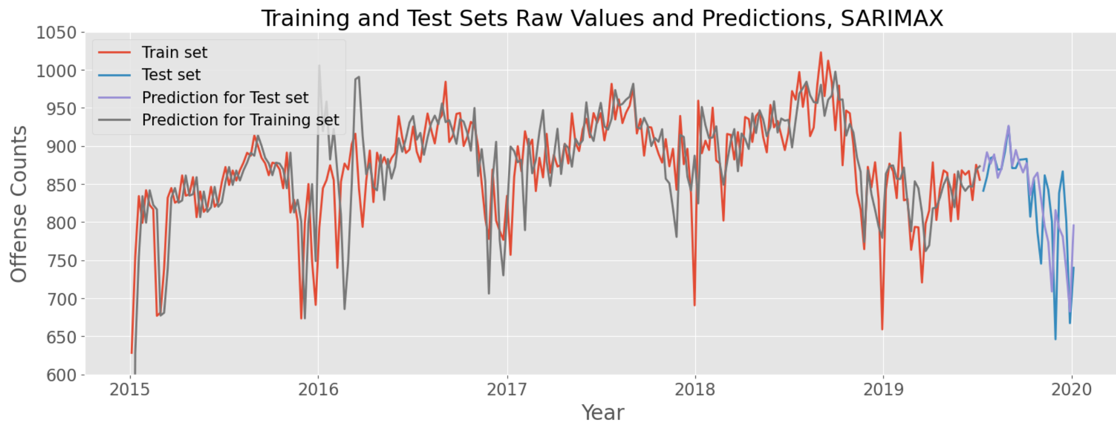


Testing

[161]:
```
y_hat_train=sarimax_mod1.predict(typ='levels')
y_hat_test=sarimax_mod1.predict(start=ts_test.index[0], end=ts_test.index[-1],␣
↪typ='levels')

rmse = np.sqrt(mean_squared_error(ts_test, y_hat_test))
print('RMSE of the SARIMAX model is {}'.format(round(rmse,2)))
```

```
fig=display_figure_w_TSs(ts_train, ts_test, 'Train set', 'Test set',
                         'Training and Test Sets Raw Values and Predictions,␣
↪SARIMAX',
                          n=4, ts3=y_hat_test,
                          ts4=y_hat_train, label3='Prediction for Test set',␣
↪label4='Prediction for Training set')
```

```
RMSE of the SARIMAX model is 55.53
```



The RMSE value is significantly better than the RMSE for the Baseline model. Though visually the prediction for the train and test sets are not perfect.

Adding US holidays as exogenous regressors I will add holiday exogenous variables as an additional argument to the SARIMAX model to see if it helps improving the performance.

[162]:
```
ts_holidays_weekly=us_holidays_predictors_TS(start='1/1/2015',
    end='12/31/2019',
    years=range(2015, 2020),
    freq='W')
```

[163]:
```
# Spliting exogenous TS into the training and the test parts

train_size_holidays = round(len(ts_holidays_weekly) * 0.90)
ts_train_holiday, ts_test_holiday = ts_holidays_weekly[:train_size],␣
↪ts_holidays_weekly[train_size:]
```

[164]:
```
# sarimax_mod2=SARIMAX(ts_train, exog=ts_train_holiday,
#                      order=(3, 1, 0),
#                      seasonal_order=(3, 1, 0, 52),
#                      enforce_invertibility=False).fit()
```

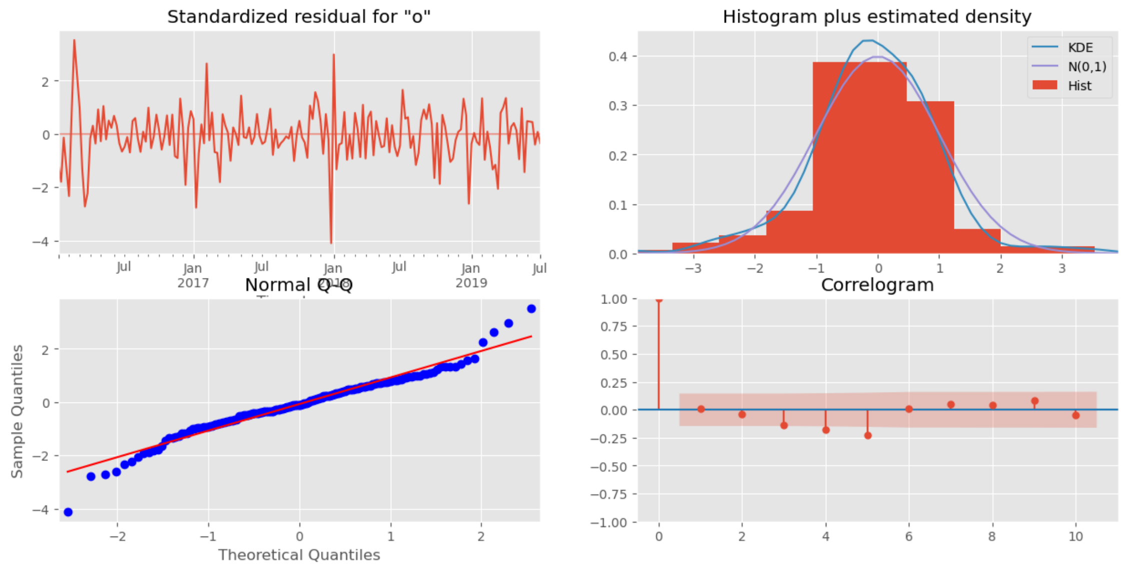
The model above took 1.5 minute to fit, therefore it is pickled to be used forward.

[165]:
```
# with open('data/pickled_models/sarimax_mod2.pickle', 'wb') as f:
#     pickle.dump(sarimax_mod2, f)
```

[166]:
```
with open('data/pickled_models/sarimax_mod2.pickle', 'rb') as f:
    sarimax_mod2=pickle.load(f)
```

[167]:
```
diagnostics(sarimax_mod2)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                           offense_id   No. Observations:                236
Model:             SARIMAX(3, 1, 0)x(3, 1, 0, 52)   Log Likelihood                -971.795
Date:                              Sun, 18 Jul 2021   AIC                           1959.590
Time:                                      19:16:44   BIC                           1985.266
Sample:                                  01-04-2015   HQIC                          1969.998
                                       - 07-07-2019
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Holiday       -5.5068     12.288     -0.448      0.654     -29.591      18.577
ar.L1         -0.6332      0.058    -10.933      0.000      -0.747      -0.520
ar.L2         -0.3928      0.067     -5.833      0.000      -0.525      -0.261
ar.L3         -0.1735      0.076     -2.276      0.023      -0.323      -0.024
ar.S.L52      -0.6326      0.092     -6.841      0.000      -0.814      -0.451
ar.S.L104     -0.4542      0.150     -3.020      0.003      -0.749      -0.159
ar.S.L156     -0.2450      0.163     -1.501      0.133      -0.565       0.075
sigma2      2021.2538    270.058      7.485      0.000    1491.949    2550.558
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                40.31
Prob(Q):                              0.87   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                            -0.23
Prob(H) (two-sided):                  0.06   Kurtosis:                         5.25
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```

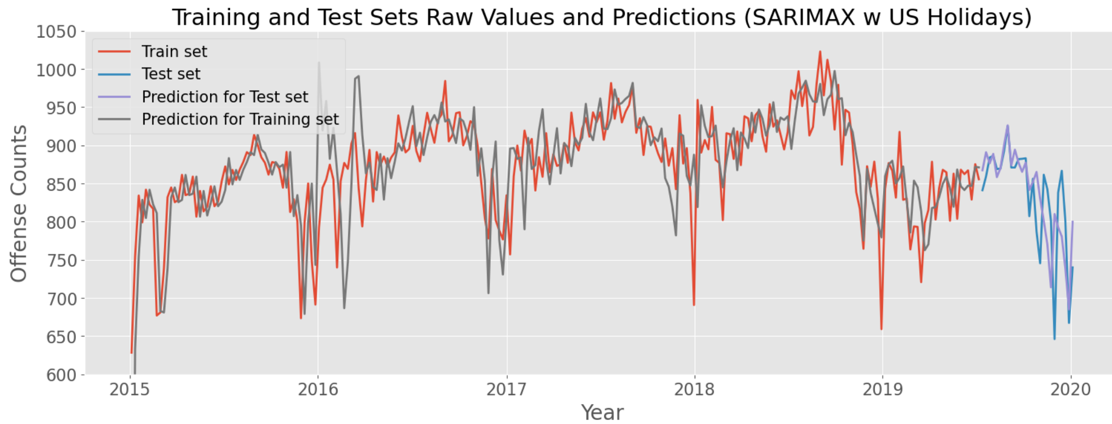


This model performed very slightly worse than the one without exogenous regressors in terms of AIC value.

Testing

[168]:
```
plt.style.use('ggplot')
y_hat_train=sarimax_mod2.predict(typ='levels')
y_hat_test=sarimax_mod2.predict(start=ts_test.index[0], end=ts_test.index[-1], exog=ts_test_holiday,typ='levels')

rmse = np.sqrt(mean_squared_error(ts_test, y_hat_test))
print('RMSE of the SARIMAX model (w US holidays) is {}'.format(round(rmse,2)))


fig=display_figure_w_TSs(ts_train, ts_test, 'Train set', 'Test set',
                         'Training and Test Sets Raw Values and Predictions (SARIMAX w US Holidays)',
                          n=4, ts3=y_hat_test,
                          ts4=y_hat_train, label3='Prediction for Test set', label4='Prediction for Training set')
```

```
RMSE of the SARIMAX model (w US holidays) is 54.65
```

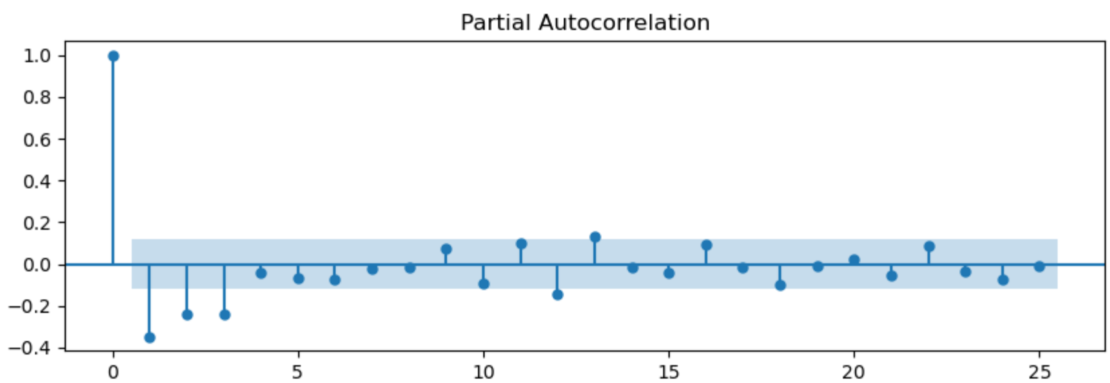


The RMSE value of this model is better than the one from the model w/o US holidays regressors.

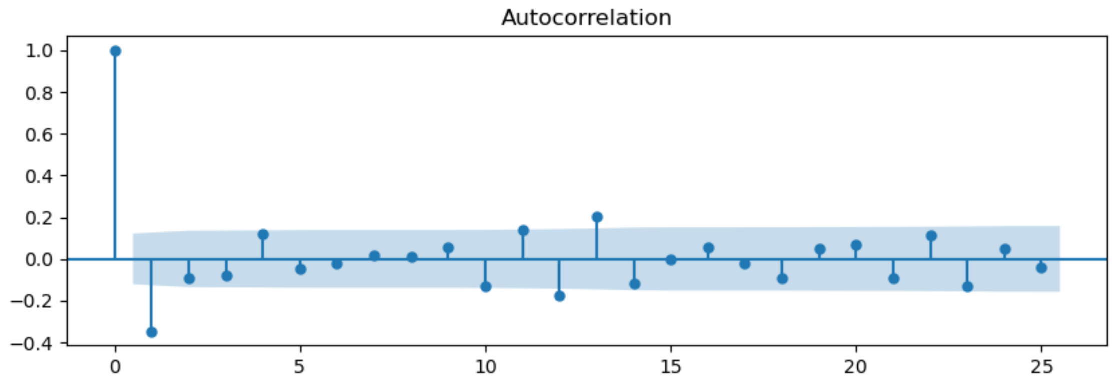
6.1.5 Forecasting

SARIMAX w/o US holidays ACF and PACF

[169]:
```
matplotlib.rc_file_defaults()
plt.rc("figure", figsize=(10,3))
plot_pacf(ts_weekly.diff().dropna());
```



[170]:
```
matplotlib.rc_file_defaults()
plt.rc("figure", figsize=(10,3))
plot_acf(ts_weekly.diff().dropna());
```



Based on PACF and ACF plots the same model would work best for the full dataset as well (not just the training subset).

Fitting the model to the full dataset

[171]:
```
# sarimax_mod1_for=SARIMAX(ts_weekly,
#                          order=(3, 1, 0),
#                          seasonal_order=(3, 1, 0, 52),
#                          enforce_invertibility=False).fit()
```

The model above took 43 seconds to fit, therefore it is pickled to be used forward.

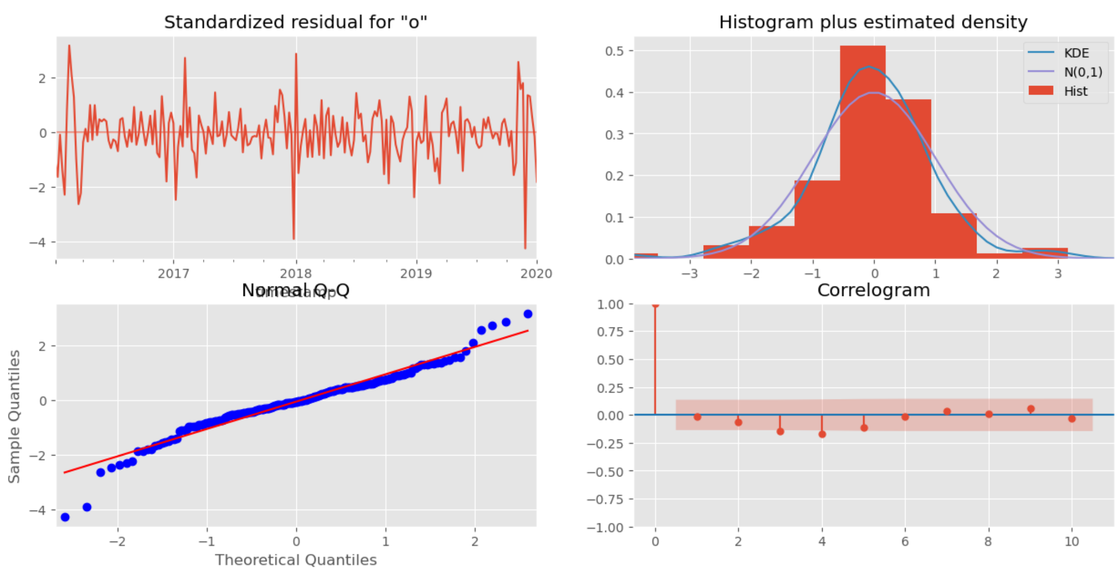
[172]:
```
# with open('data/pickled_models/sarimax_mod1_for.pickle', 'wb') as f:
#     pickle.dump(sarimax_mod1_for, f)
```

[173]:
```
with open('data/pickled_models/sarimax_mod1_for.pickle', 'rb') as f:
    sarimax_mod1_for=pickle.load(f)
```

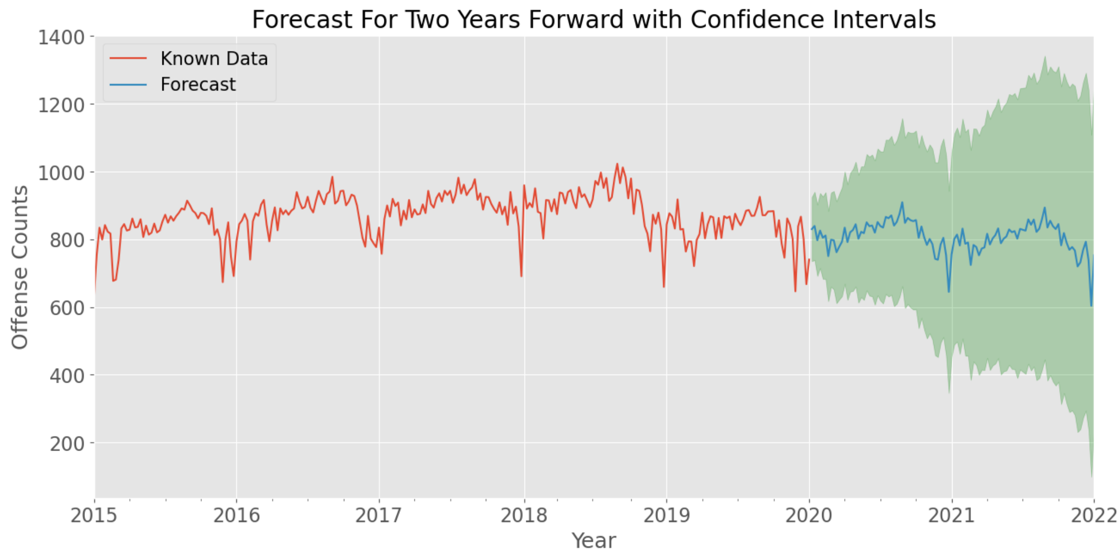
[174]:
```
diagnostics(sarimax_mod1_for)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                    SARIMAX Results
==========================================================================================
Dep. Variable:                          offense_id   No. Observations:                  262
Model:             SARIMAX(3, 1, 0)x(3, 1, 0, 52)   Log Likelihood               -1119.803
Date:                             Sun, 18 Jul 2021   AIC                           2253.605
Time:                                     19:17:15   BIC                           2277.002
Sample:                                 01-04-2015   HQIC                          2263.065
                                      - 01-05-2020
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.6327      0.050    -12.567      0.000      -0.731      -0.534
```

```
ar.L2          -0.4433      0.061     -7.311      0.000      -0.562      -0.324
ar.L3          -0.2734      0.058     -4.704      0.000      -0.387      -0.160
ar.S.L52       -0.5999      0.088     -6.819      0.000      -0.772      -0.427
ar.S.L104      -0.4110      0.124     -3.310      0.001      -0.654      -0.168
ar.S.L156      -0.1834      0.140     -1.312      0.189      -0.457       0.091
sigma2       2326.7001    217.935     10.676      0.000    1899.555    2753.846
===================================================================================
Ljung-Box (L1) (Q):                   0.06   Jarque-Bera (JB):                65.24
Prob(Q):                              0.81   Prob(JB):                         0.00
Heteroskedasticity (H):               1.01   Skew:                            -0.42
Prob(H) (two-sided):                  0.97   Kurtosis:                         5.60
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```



[175]:
```
fig=plot_predictions(ts_weekly, sarimax_mod1_for, 'Forecast For Two Years␣
↪Forward with Confidence Intervals',
                     steps=104, xmin='2015')
```



SARIMAX with US holidays Fitting the model to the full dataset with full US holiday schedule

[176]:
```
# sarimax_mod2_for=SARIMAX(ts_weekly, exog=ts_holidays_weekly,
#                          order=(3, 1, 0),
#                          seasonal_order=(3, 1, 0, 52),
#                          enforce_invertibility=False).fit()
```

The fitting of the model took 2.5 minutes therefore I am saving it to a pickle file.

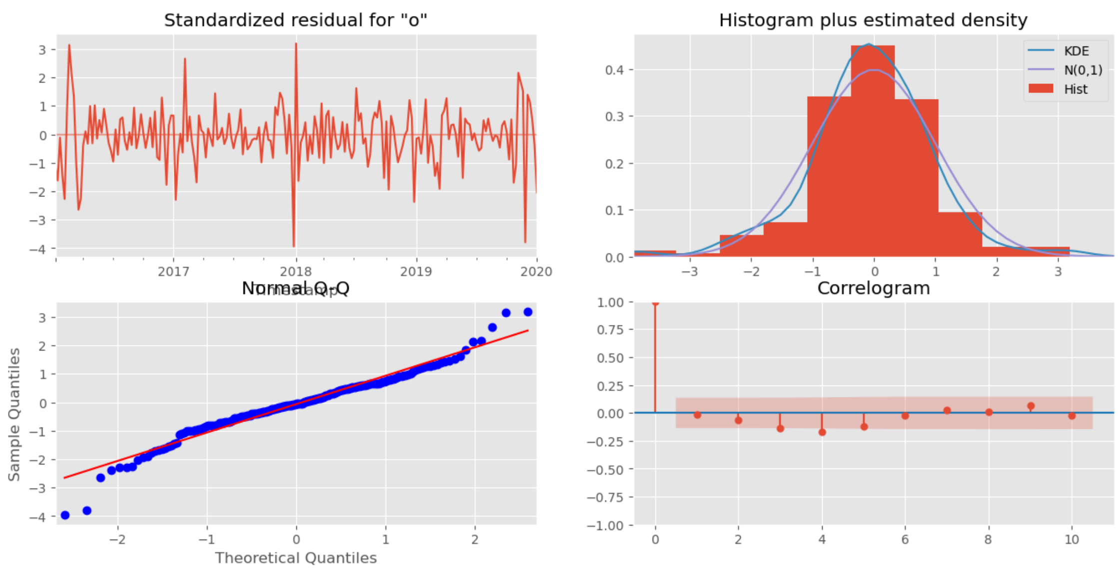
[177]:
```
# with open('data/pickled_models/sarimax_mod2_for.pickle', 'wb') as f:
#     pickle.dump(sarimax_mod2_for, f)
```

[178]:
```
with open('data/pickled_models/sarimax_mod2_for.pickle', 'rb') as f:
    sarimax_mod2_for=pickle.load(f)
```

[179]:
```
diagnostics(sarimax_mod2_for)
```

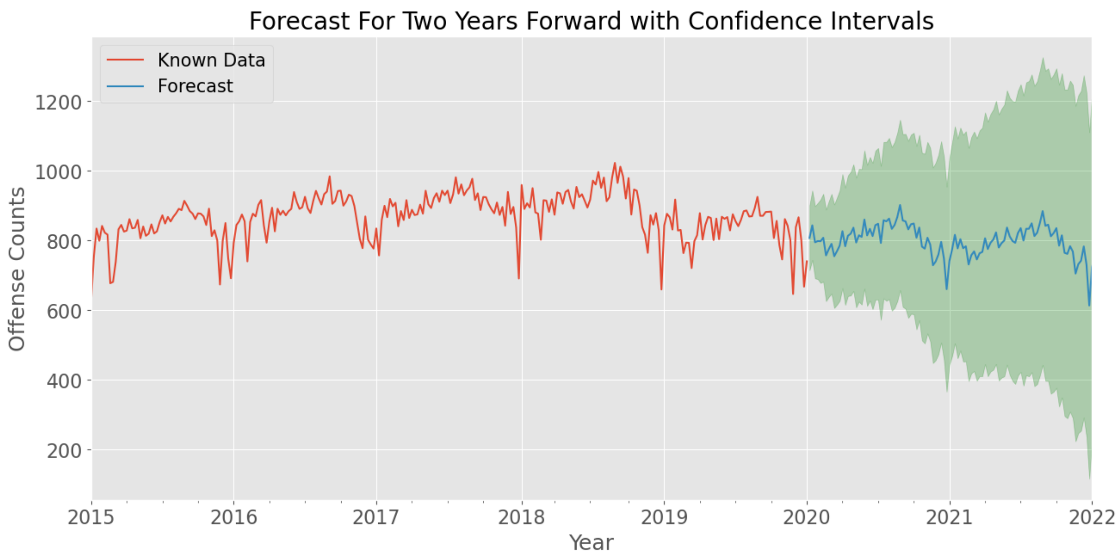
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                          offense_id   No. Observations:                 262
Model:             SARIMAX(3, 1, 0)x(3, 1, 0, 52)   Log Likelihood               -1117.886
Date:                             Sun, 18 Jul 2021   AIC                           2251.771
Time:                                     19:17:33   BIC                           2278.510
Sample:                                 01-04-2015   HQIC                          2262.582
                                      - 01-05-2020
Covariance Type:                               opg
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Holiday      -16.8103      6.897     -2.437      0.015     -30.328      -3.292
ar.L1         -0.6228      0.049    -12.602      0.000      -0.720      -0.526
ar.L2         -0.4621      0.061     -7.581      0.000      -0.582      -0.343
ar.L3         -0.2657      0.067     -3.936      0.000      -0.398      -0.133
ar.S.L52      -0.6029      0.091     -6.594      0.000      -0.782      -0.424
ar.S.L104     -0.4262      0.126     -3.372      0.001      -0.674      -0.178
ar.S.L156     -0.1967      0.144     -1.368      0.171      -0.478       0.085
sigma2      2271.3878    236.961      9.585      0.000    1806.952    2735.823
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                45.46
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):               0.95   Skew:                            -0.33
Prob(H) (two-sided):                  0.84   Kurtosis:                         5.19
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```



[180]:
```
# I am extending the period to 105 weeks to cover all the span of the
↪US_holidays TS above
# (it includes 53 weeks+52 weeks next year)
```

```
fig=plot_predictions(ts_weekly, sarimax_mod2_for, 'Forecast For Two Years␣
↪Forward with Confidence Intervals',
                  steps=105, xmin='2015',
                  egog_flag=True, exog=exog_reg_timeframe('1/1/2020', '1/1/
↪2022'))
```



Auto ARIMA search for the best parameters Training and testing

[181]:
```
# auto_model_train = pmd.auto_arima(ts_train,
#                               start_p=0,start_q=0, d=1,
#                               max_p=3,max_q=1,
#                               max_P=3,max_Q=1,
#                               start_P=0, start_Q=0, D=1,
#                               m=52,trace=True,
#                               verbose=2)
```

executed in 29m 56s, finished 23:29:20 2021-07-14

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[52]             : AIC=2030.878, Time=0.14 sec
 ARIMA(1,1,0)(1,1,0)[52]             : AIC=1981.679, Time=2.04 sec
 ARIMA(0,1,1)(0,1,1)[52]             : AIC=inf, Time=4.15 sec
 ARIMA(1,1,0)(0,1,0)[52]             : AIC=2002.205, Time=0.30 sec
 ARIMA(1,1,0)(2,1,0)[52]             : AIC=1974.602, Time=8.17 sec
 ARIMA(1,1,0)(3,1,0)[52]             : AIC=inf, Time=31.73 sec
 ARIMA(1,1,0)(2,1,1)[52]             : AIC=inf, Time=18.88 sec
 ARIMA(1,1,0)(1,1,1)[52]             : AIC=inf, Time=12.71 sec
 ARIMA(1,1,0)(3,1,1)[52]             : AIC=1976.784, Time=44.05 sec
 ARIMA(0,1,0)(2,1,0)[52]             : AIC=2010.941, Time=6.07 sec
 ARIMA(2,1,0)(2,1,0)[52]             : AIC=1960.136, Time=11.07 sec
 ARIMA(2,1,0)(1,1,0)[52]             : AIC=1966.722, Time=3.15 sec
 ARIMA(2,1,0)(3,1,0)[52]             : AIC=inf, Time=38.62 sec
 ARIMA(2,1,0)(2,1,1)[52]             : AIC=inf, Time=23.66 sec
 ARIMA(2,1,0)(1,1,1)[52]             : AIC=inf, Time=7.88 sec
 ARIMA(2,1,0)(3,1,1)[52]             : AIC=1963.287, Time=53.10 sec
 ARIMA(3,1,0)(2,1,0)[52]             : AIC=1957.526, Time=14.56 sec
 ARIMA(3,1,0)(1,1,0)[52]             : AIC=1963.767, Time=3.88 sec
 ARIMA(3,1,0)(3,1,0)[52]             : AIC=inf, Time=45.00 sec
 ARIMA(3,1,0)(2,1,1)[52]             : AIC=inf, Time=30.04 sec
 ARIMA(3,1,0)(1,1,1)[52]             : AIC=inf, Time=19.36 sec
 ARIMA(3,1,0)(3,1,1)[52]             : AIC=1959.937, Time=62.02 sec
 ARIMA(3,1,1)(2,1,0)[52]             : AIC=1940.570, Time=29.35 sec
 ARIMA(3,1,1)(1,1,0)[52]             : AIC=1945.851, Time=8.29 sec
 ARIMA(3,1,1)(3,1,0)[52]             : AIC=1941.304, Time=78.07 sec
 ARIMA(3,1,1)(2,1,1)[52]             : AIC=inf, Time=62.33 sec
 ARIMA(3,1,1)(1,1,1)[52]             : AIC=inf, Time=21.50 sec
 ARIMA(3,1,1)(3,1,1)[52]             : AIC=1943.304, Time=87.79 sec
 ARIMA(2,1,1)(2,1,0)[52]             : AIC=1938.628, Time=23.60 sec
 ARIMA(2,1,1)(1,1,0)[52]             : AIC=1943.953, Time=6.60 sec
 ARIMA(2,1,1)(3,1,0)[52]             : AIC=1939.309, Time=62.31 sec
 ARIMA(2,1,1)(2,1,1)[52]             : AIC=inf, Time=32.64 sec
 ARIMA(2,1,1)(1,1,1)[52]             : AIC=inf, Time=21.80 sec
 ARIMA(2,1,1)(3,1,1)[52]             : AIC=1941.309, Time=65.30 sec
 ARIMA(1,1,1)(2,1,0)[52]             : AIC=1936.683, Time=15.17 sec
 ARIMA(1,1,1)(1,1,0)[52]             : AIC=1941.970, Time=4.47 sec
 ARIMA(1,1,1)(3,1,0)[52]             : AIC=1937.403, Time=47.51 sec
 ARIMA(1,1,1)(2,1,1)[52]             : AIC=inf, Time=24.39 sec
 ARIMA(1,1,1)(1,1,1)[52]             : AIC=inf, Time=19.71 sec
 ARIMA(1,1,1)(3,1,1)[52]             : AIC=1939.403, Time=56.63 sec
 ARIMA(0,1,1)(2,1,0)[52]             : AIC=1936.612, Time=10.39 sec
 ARIMA(0,1,1)(1,1,0)[52]             : AIC=1942.505, Time=2.69 sec
 ARIMA(0,1,1)(3,1,0)[52]             : AIC=1936.962, Time=34.36 sec
 ARIMA(0,1,1)(2,1,1)[52]             : AIC=inf, Time=17.49 sec
 ARIMA(0,1,1)(1,1,1)[52]             : AIC=inf, Time=13.34 sec
 ARIMA(0,1,1)(3,1,1)[52]             : AIC=1938.962, Time=44.74 sec
 ARIMA(0,1,1)(2,1,0)[52] intercept   : AIC=1933.412, Time=14.70 sec
 ARIMA(0,1,1)(1,1,0)[52] intercept   : AIC=1940.285, Time=5.96 sec
 ARIMA(0,1,1)(3,1,0)[52] intercept   : AIC=1933.673, Time=52.78 sec
 ARIMA(0,1,1)(2,1,1)[52] intercept   : AIC=inf, Time=53.25 sec
 ARIMA(0,1,1)(1,1,1)[52] intercept   : AIC=inf, Time=18.68 sec
 ARIMA(0,1,1)(3,1,1)[52] intercept   : AIC=inf, Time=65.89 sec
 ARIMA(0,1,0)(2,1,0)[52] intercept   : AIC=2012.806, Time=11.98 sec
 ARIMA(1,1,1)(2,1,0)[52] intercept   : AIC=1932.341, Time=28.83 sec
 ARIMA(1,1,1)(1,1,0)[52] intercept   : AIC=1938.748, Time=7.02 sec
 ARIMA(1,1,1)(3,1,0)[52] intercept   : AIC=1933.069, Time=65.45 sec
 ARIMA(1,1,1)(2,1,1)[52] intercept   : AIC=inf, Time=35.05 sec
 ARIMA(1,1,1)(1,1,1)[52] intercept   : AIC=inf, Time=21.06 sec
 ARIMA(1,1,1)(3,1,1)[52] intercept   : AIC=inf, Time=97.09 sec
 ARIMA(1,1,0)(2,1,0)[52] intercept   : AIC=1976.309, Time=18.49 sec
 ARIMA(2,1,1)(2,1,0)[52] intercept   : AIC=1934.090, Time=37.89 sec
 ARIMA(2,1,0)(2,1,0)[52] intercept   : AIC=1961.684, Time=28.25 sec

Best model:  ARIMA(1,1,1)(2,1,0)[52] intercept
Total fit time: 1794.188 seconds
```

The search above took 29.5 minutes to run, therefore it is pickled to be used forward.

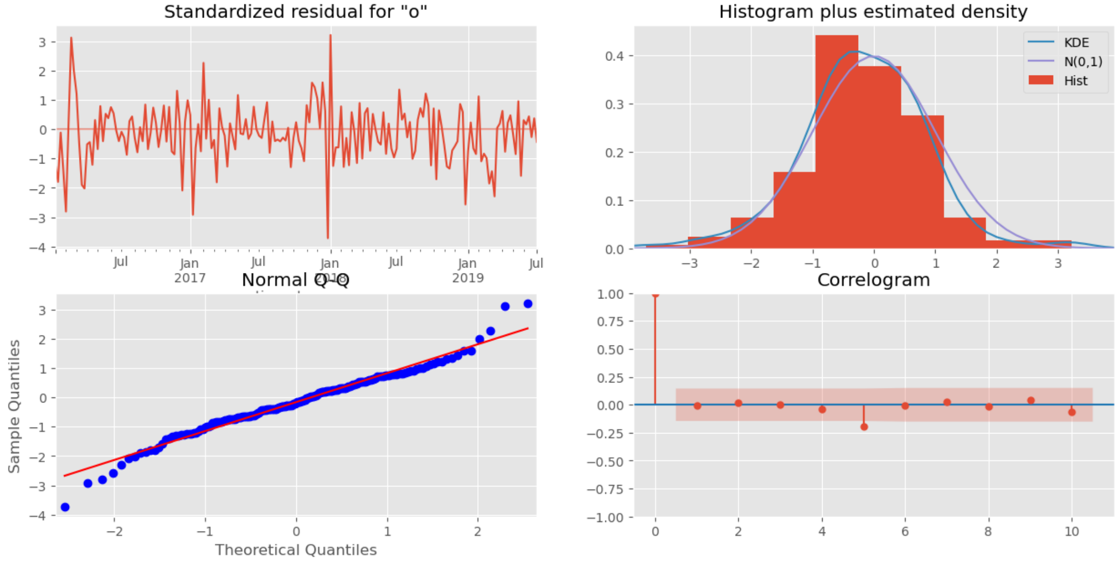
[182]:
```
# with open('data/pickled_models/auto_model_train.pickle', 'wb') as f:
#     pickle.dump(auto_model_train, f)
```

[183]:
```
with open('data/pickled_models/auto_model_train.pickle', 'rb') as f:
    auto_model_train=pickle.load(f)
```

[184]:
```
model_auto_train = tsa.SARIMAX(ts_train, order=auto_model_train.order,
                seasonal_order=auto_model_train.seasonal_order,
                enforce_invertibility=False, freq='W').fit()
```

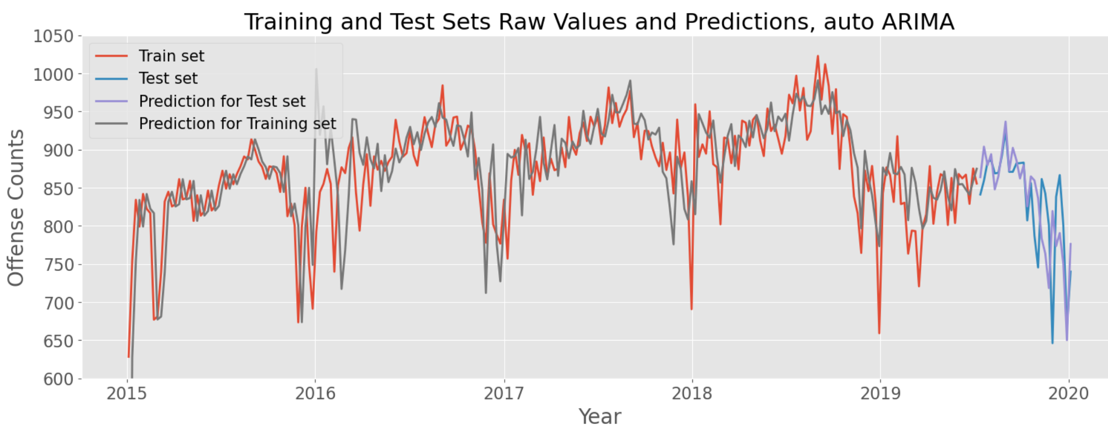
[185]:
```
diagnostics(model_auto_train)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
===========================================================================================
Dep. Variable:                             offense_id   No. Observations:                236
Model:             SARIMAX(1, 1, 1)x(2, 1, [], 52)   Log Likelihood              -963.341
Date:                            Sun, 18 Jul 2021   AIC                         1936.683
Time:                                    19:18:12   BIC                         1952.730
Sample:                                01-04-2015   HQIC                        1943.188
                                     - 07-07-2019
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1223      0.065      1.885      0.059      -0.005       0.249
ma.L1         -0.8601      0.041    -20.853      0.000      -0.941      -0.779
ar.S.L52      -0.4910      0.061     -8.081      0.000      -0.610      -0.372
ar.S.L104     -0.2871      0.093     -3.091      0.002      -0.469      -0.105
sigma2      1979.7726    194.117     10.199      0.000    1599.309    2360.236
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                19.00
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.70   Skew:                            -0.10
Prob(H) (two-sided):                  0.16   Kurtosis:                         4.57
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```



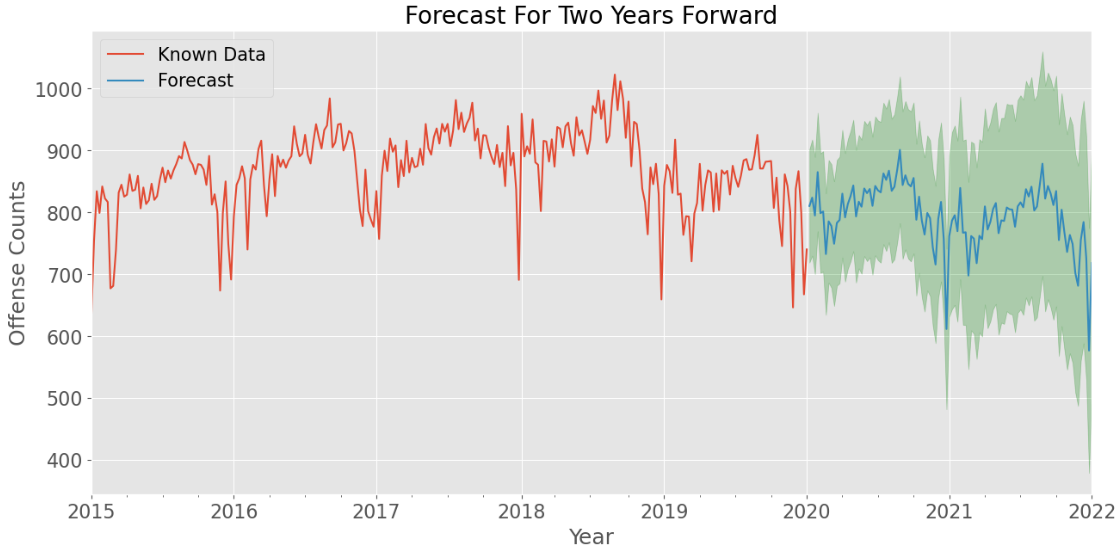
[186]:
```
y_hat_train=model_auto_train.predict(typ='levels')
y_hat_test=model_auto_train.predict(start=ts_test.index[0], end=ts_test.
 ↪index[-1], typ='levels')

rmse = np.sqrt(mean_squared_error(ts_test, y_hat_test))
print('RMSE of the auto ARIMA model is {}'.format(round(rmse,2)))

fig=display_figure_w_TSs(ts_train, ts_test, 'Train set', 'Test set',
                         'Training and Test Sets Raw Values and Predictions, auto␣
 ↪ARIMA',
                          n=4, ts3=y_hat_test,
                          ts4=y_hat_train, label3='Prediction for Test set',␣
 ↪label4='Prediction for Training set')
```

```
RMSE of the auto ARIMA model is 56.07
```



[187]:
```
model_auto_for = tsa.SARIMAX(ts_weekly, order=auto_model_train.order,
                seasonal_order=auto_model_train.seasonal_order,
                enforce_invertibility=False, freq='W').fit()
```

[195]:
```
fig=plot_predictions(ts_weekly, model_auto_for, 'Forecast For Two Years␣
↪Forward', steps=104, xmin='2015')
```



It takes ~3 minutes to run this notebook

[215]:
```
plot_predictions_px(ts_weekly, model_auto_for, 'Test', xmin='2015')
```

6.2 Crime Rate per Offense Category Modeling

All Crime rate modeling for various crime categories are located in part III notebook. The reason is to make all notebook manageable.

7 iNTERPRET

8 CONCLUSIONS & RECOMMENDATIONS

Summarize your conclusions and bullet-point your list of recommendations, which are based on your modeling results.

9 TO DO/FUTURE WORK

-

```
[ ]:
```