bostonrobberies

December 12, 2019

1 Massachusetts and Boston Crime Analysis

1.1 Authors: Jenny Zhang, Anthony Castillo, Eisuke Okuda

2 Abstract

Crime data analytics have played a big role in improving law enforcement for the last many years. It has helped identify and analyze patterns and trends in crime and disorder. Thus it is important for us to understand the method and put it into practice. We studied the crime datasets in Boston and Massachusetts from 1960 to 2018 to answer our questions. We wanted to see if there were changes in different kinds of crimes, such as property and violent crime, and if we can accurately predict the rise (or fall) robberies in Boston specifically. Our project has two parts: exploratory data analysis on Massachussetts crime data and time series analysis on Boston robberies data.

3 Introduction

The main purpose of our analysis was to study the trends in crime data in Massachussetts over time and forecast robberies in Boston. In our exploratory data analysis, we wanted to study the Massachussetts dataset, which contains data on population, the amount of crime committed in the categories: Murder, Rape, Robbery, Aggravated Assault, Burglary, Larceny Theft and Vehicle Theft, the amount of violent and property crimes, and index: the amount of total (both violent and property) crimes. Property crimes consist of Robbery, Burglary, Larceny and Vehicle Theft while Violent crimes consist of Murder, Rape and Aggravated Assault. We wanted to study specific questions, such as what the most common crimes are, how violent crimes compare to property crimes, and how crimes changed throughout time. Additionally, we wanted to study the density of crimes per year using heatmaps.

The second part of our project consists of a time series analysis on robberies in Boston from 1960 to 2018. Here we took the time series data we were given and decomposed it for a preliminary analysis. Afterwards, we established factors for auto regression, differencing, moving average, and seasonality parameters. Then, we took our data and used one-step forecasting to predict Boston Robberies for the next 4 years.

Lastly we wanted to see if there were any correlations between crime in Massachussetts and crime in Boston using hypothesis testing.

```
In [21]: import pandas as pd
         from random import randrange
         import numpy as np
```

```
import seaborn as sns
from matplotlib import pyplot
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from pandas.plotting import autocorrelation_plot
from scipy import stats
import scipy
import os
import re
import datetime
import itertools
import warnings
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
```

This is just us loading the Massachusetts crime dataset, and then converting all fields in that dataset from string type to float type. This makes statistical analysis possible. We then take the theft-related data from Massachusetts and trim it to match up with our Boston data in terms of what time interval will be discussed. Lastly, we aggregate our monthly Boston data into an annual sum using resample, and then convert the new object into a list.

```
In [22]: mass = pd.read_csv('massachusettsdata.csv', header=0, index_col=0,parse_dates=True, s
         mass['Population'] = mass['Population'].str.replace(',','').astype(float)
         mass['Index'] = mass['Index'].str.replace(',','').astype(float)
         mass['Violent'] = mass['Violent'].str.replace(',','').astype(float)
         mass['Property'] = mass['Property'].str.replace(',','').astype(float)
         mass['Murder'] = mass['Murder'].str.replace(',','').astype(float)
         mass['Forcible Rape'] = mass['Forcible Rape'].str.replace(',','').astype(float)
         mass['Robbery'] = mass['Robbery'].str.replace(',','').astype(float)
         mass['Aggravated Assault'] = mass['Aggravated Assault'].str.replace(',','').astype(fl
         mass['Burglary'] = mass['Burglary'].str.replace(',','').astype(float)
         mass['Larceny - Theft'] = mass['Larceny - Theft'].str.replace(',','').astype(float)
         mass['Vehicle Theft'] = mass['Vehicle Theft'].str.replace(',','').astype(float)

In [23]: mass

Out[23]:
```

Year	Population	Index	Violent	Property	Murder	Forcible Rape	\
1960-01-01	5148578.0	62767.0	2512.0	60255.0	74.0	249.0	
1961-01-01	5234000.0	76560.0	2777.0	73783.0	77.0	291.0	
1962-01-01	5161000.0	84107.0	3207.0	80900.0	95.0	256.0	
1963-01-01	5218000.0	90056.0	3454.0	86602.0	101.0	236.0	
1964-01-01	5338000.0	106099.0	4559.0	101540.0	105.0	320.0	
1965-01-01	5348000.0	113455.0	5270.0	108185.0	129.0	290.0	
1966-01-01	5383000.0	122477.0	6201.0	116276.0	128.0	344.0	
1967-01-01	5421000.0	134557.0	6919.0	127638.0	154.0	411.0	
1968-01-01	5437000.0	166802.0	8916.0	157886.0	188.0	518.0	
1969-01-01	5467000.0	190173.0	10272.0	179901.0	191.0	592.0	

1970-01-01	5689170.0	213134.0	11542.0	201592.0	197.0	684.0
1971-01-01	5758000.0	250332.0	15317.0	235015.0	220.0	715.0
1972-01-01	5787000.0	237677.0	17086.0	220591.0	215.0	784.0
1973-01-01	5818000.0	263031.0	20475.0	242556.0	256.0	949.0
1974-01-01	5800000.0	312211.0	22545.0	289666.0	256.0	907.0
1975-01-01	5828000.0	354216.0	25793.0	328423.0	242.0	1121.0
1976-01-01	5809000.0	338136.0	23190.0	314946.0	194.0	1028.0
1977-01-01	5782000.0	312751.0	24593.0	288158.0	178.0	1203.0
1978-01-01	5774000.0	308933.0	26673.0	282260.0	216.0	1307.0
1979-01-01	5769000.0	341406.0	30650.0	310756.0	212.0	1428.0
1980-01-01	5728288.0	348231.0	34444.0	313787.0	232.0	1562.0
1981-01-01	5770000.0	336701.0	36273.0	300428.0	210.0	1580.0
1982-01-01	5781000.0	318171.0	33031.0	285140.0	219.0	1464.0
1983-01-01	5767000.0	288971.0	33264.0	255707.0	203.0	1495.0
1984-01-01	5798000.0	266037.0	30362.0	235675.0	211.0	1627.0
1985-01-01	5822000.0	276999.0	31334.0	245665.0	202.0	1734.0
1986-01-01	5832000.0	275465.0	32476.0	242989.0	208.0	1731.0
1987-01-01	5855000.0	277165.0	33060.0	244105.0	173.0	1868.0
1988-01-01	5871000.0	293015.0	36376.0	256639.0	208.0	1876.0
1989-01-01	5913000.0	303692.0	39912.0	263780.0	254.0	1881.0
1990-01-01	6016425.0	318742.0	44300.0	274442.0	243.0	2030.0
1991-01-01	5996000.0	319128.0	44138.0	274990.0	249.0	1926.0
1992-01-01	5998000.0	300071.0	46727.0	253344.0	214.0	2166.0
1993-01-01	6012000.0	294224.0	48393.0	245831.0	233.0	2006.0
1994-01-01	6041000.0	268281.0	42749.0	225532.0	214.0	1825.0
1995-01-01	6074000.0	263710.0	41739.0	221971.0	217.0	1759.0
1996-01-01	6092000.0	233758.0	39122.0	194636.0	157.0	1767.0
1997-01-01	6118000.0	224848.0	39411.0	185437.0	119.0	1647.0
1998-01-01	6147000.0	211203.0	38192.0	173011.0	124.0	1687.0
1999-01-01	6175169.0	201460.0	34023.0	167437.0	122.0	1663.0
2000-01-01	6349097.0	192131.0	30230.0	161901.0	125.0	1696.0
2001-01-01	6401164.0	197664.0	30585.0	167079.0	143.0	1856.0
2002-01-01	6421800.0	198890.0	31137.0	167753.0	173.0	1777.0
2003-01-01	6420357.0	194920.0	30377.0	164543.0	140.0	1848.0
2004-01-01	6407382.0	187639.0	29489.0	158150.0	171.0	1794.0
2005-01-01	6433367.0	180362.0	29644.0	151727.0	178.0	1751.0
2006-01-01	6437193.0	181688.0	28775.0	153913.0	186.0	1742.0
2007-01-01	6449755.0	181058.0	27832.0	154246.0	184.0	1634.0
2008-01-01	6543595.0	185971.0	29888.0	156083.0	167.0	1744.0
2009-01-01	6593587.0	183681.0	30503.0	153178.0	173.0	1734.0
2010-01-01	6555466.0	185233.0	30737.0	154496.0	214.0	1784.0
2011-01-01	6607003.0	177061.0	28232.0	148829.0	184.0	1654.0
2012-01-01	6645303.0	170372.0	27047.0	143325.0	121.0	1650.0
2013-01-01	6708874.0	164538.0	27264.0	137274.0	138.0	1722.0
2014-01-01	6755124.0	152170.0	26689.0	125481.0	133.0	1625.0
2015-01-01	6784240.0	141000.0	26453.0	114547.0	131.0	1538.0
2016-01-01	6811779.0	132016.0	25677.0	106339.0	134.0	1592.0
2017-01-01	6863246.0	122295.0	24318.0	97977.0	172.0	2199.0

```
2018-01-01   6902149.0  110533.0  23337.0   87196.0   136.0         2410.0

            Robbery  Aggravated Assault  Burglary  Larceny - Theft  \
Year
1960-01-01   1052.0              1137.0   15918.0          33469.0
1961-01-01   1066.0              1343.0   19683.0          39885.0
1962-01-01   1331.0              1525.0   21181.0          44051.0
1963-01-01   1409.0              1708.0   23121.0          44382.0
1964-01-01   1636.0              2498.0   28278.0          49129.0
1965-01-01   2139.0              2712.0   29655.0          49997.0
1966-01-01   2474.0              3255.0   33326.0          53244.0
1967-01-01   2818.0              3536.0   36621.0          54837.0
1968-01-01   4039.0              4171.0   47210.0          66823.0
1969-01-01   4955.0              4534.0   56450.0          76501.0
1970-01-01   5658.0              5003.0   64523.0          87114.0
1971-01-01   8069.0              6313.0   77145.0         101161.0
1972-01-01   8840.0              7247.0   71894.0          92425.0
1973-01-01  10586.0              8684.0   77395.0         100605.0
1974-01-01  12317.0              9065.0   89891.0         120572.0
1975-01-01  13229.0             11201.0   99802.0         137058.0
1976-01-01  10466.0             11502.0   96554.0         142135.0
1977-01-01   9822.0             13390.0   88594.0         133642.0
1978-01-01   9947.0             15203.0   87482.0         131519.0
1979-01-01  11724.0             17286.0   92570.0         152135.0
1980-01-01  13492.0             19158.0   99697.0         153849.0
1981-01-01  15633.0             18850.0   95080.0         147691.0
1982-01-01  12359.0             18989.0   82212.0         146933.0
1983-01-01  12023.0             19543.0   72291.0         133883.0
1984-01-01  10122.0             18402.0   63756.0         123214.0
1985-01-01  10974.0             18424.0   65231.0         130088.0
1986-01-01  11239.0             19298.0   62455.0         127668.0
1987-01-01  10379.0             20640.0   62056.0         127939.0
1988-01-01  10352.0             23940.0   62307.0         141933.0
1989-01-01  11980.0             25797.0   63004.0         146925.0
1990-01-01  13062.0             28965.0   66942.0         151933.0
1991-01-01  11669.0             30294.0   69977.0         149930.0
1992-01-01  11059.0             33288.0   64318.0         141610.0
1993-01-01  10563.0             35591.0   60220.0         136548.0
1994-01-01  10160.0             30550.0   53222.0         129962.0
1995-01-01   9137.0             30626.0   49669.0         135586.0
1996-01-01   7778.0             29420.0   42896.0         119562.0
1997-01-01   6676.0             30969.0   40491.0         115494.0
1998-01-01   5938.0             30443.0   37333.0         109275.0
1999-01-01   5931.0             26307.0   32964.0         108845.0
2000-01-01   5815.0             22594.0   30600.0         105425.0
2001-01-01   6476.0             22110.0   32430.0         106821.0
2002-01-01   7169.0             22018.0   33243.0         107922.0
2003-01-01   8011.0             20378.0   34963.0         104069.0
```

2004-01-01	7484.0	20040.0	34497.0	101605.0
2005-01-01	7837.0	19878.0	34728.0	98079.0
2006-01-01	8047.0	18800.0	35181.0	100771.0
2007-01-01	7006.0	19008.0	35662.0	103592.0
2008-01-01	7071.0	20906.0	36260.0	107048.0
2009-01-01	7467.0	21129.0	34515.0	106799.0
2010-01-01	6897.0	21842.0	37903.0	105124.0
2011-01-01	6768.0	19626.0	36403.0	101644.0
2012-01-01	6555.0	18721.0	34635.0	99453.0
2013-01-01	6705.0	18118.0	30716.0	97437.0
2014-01-01	6077.0	18277.0	24951.0	92226.0
2015-01-01	5266.0	18974.0	21826.0	84676.0
2016-01-01	5365.0	18050.0	19193.0	79088.0
2017-01-01	4874.0	17073.0	16916.0	73573.0
2018-01-01	4143.0	16648.0	13862.0	66728.0

Year	Vehicle Theft
1960-01-01	10868.0
1961-01-01	14215.0
1962-01-01	15668.0
1963-01-01	19099.0
1964-01-01	24133.0
1965-01-01	28533.0
1966-01-01	29706.0
1967-01-01	36180.0
1968-01-01	43853.0
1969-01-01	46950.0
1970-01-01	49955.0
1971-01-01	56709.0
1972-01-01	56272.0
1973-01-01	64556.0
1974-01-01	79203.0
1975-01-01	91563.0
1976-01-01	76257.0
1977-01-01	65922.0
1978-01-01	63259.0
1979-01-01	66051.0
1980-01-01	60241.0
1981-01-01	57657.0
1982-01-01	55995.0
1983-01-01	49533.0
1984-01-01	48705.0
1985-01-01	50346.0
1986-01-01	52866.0
1987-01-01	54110.0
1988-01-01	52399.0
1989-01-01	53851.0

```
1990-01-01        55567.0
1991-01-01        55083.0
1992-01-01        47416.0
1993-01-01        49063.0
1994-01-01        42348.0
1995-01-01        36716.0
1996-01-01        32178.0
1997-01-01        29452.0
1998-01-01        26403.0
1999-01-01        25628.0
2000-01-01        25876.0
2001-01-01        27828.0
2002-01-01        26588.0
2003-01-01        25511.0
2004-01-01        22048.0
2005-01-01        18920.0
2006-01-01        17961.0
2007-01-01        14992.0
2008-01-01        12775.0
2009-01-01        11864.0
2010-01-01        11469.0
2011-01-01        10782.0
2012-01-01         9237.0
2013-01-01         9121.0
2014-01-01         8304.0
2015-01-01         8045.0
2016-01-01         8058.0
2017-01-01         7488.0
2018-01-01         6606.0
```

```
In [24]: pop = mass['Population'].loc['1966-01-01':'1975-10-01']
         ind = mass['Index'].loc['1966-01-01':'1975-10-01']
         rob = mass['Robbery'].loc['1966-01-01':'1975-10-01']
         lar = mass['Larceny - Theft'].loc['1966-01-01':'1975-10-01']
         veh = mass['Vehicle Theft'].loc['1966-01-01':'1975-10-01']
         bur = mass['Burglary'].loc['1966-01-01':'1975-10-01']
```

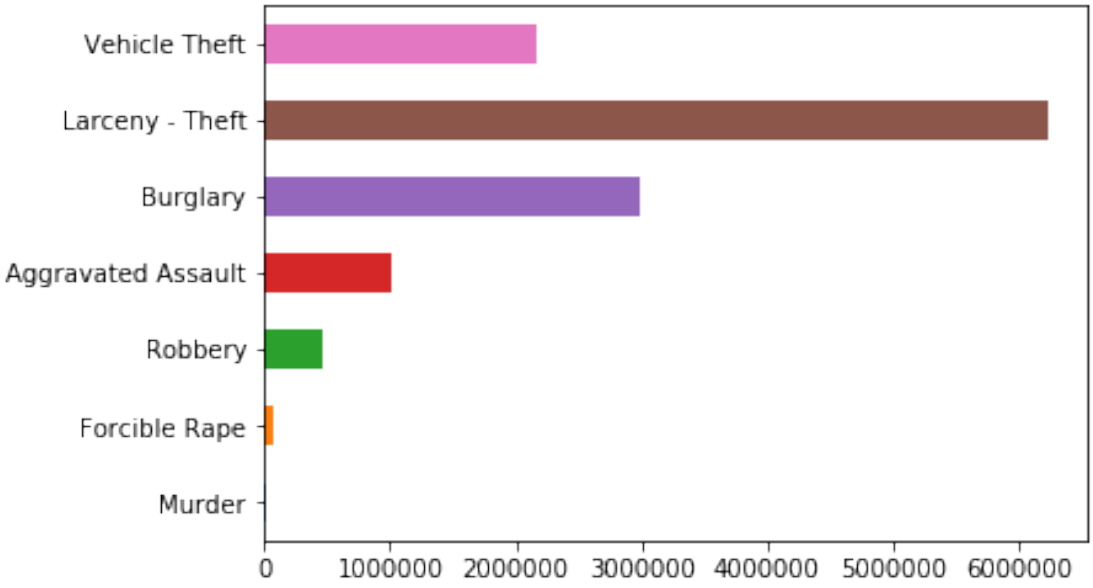
4 Exploratory Data Analysis

4.1 What are the most common crimes?

4.1.1 Barplot of all the individual crimes:

```
In [25]: #write code for barplot of all the individual crimes
         #(sum the columns for murder, forcible rape, robbery, aggravated assault, burglary, l
         # vehicle theft, and plot a bar of the columns as x axis vs the total sum on y axis)
         #mass.loc["Sum"]=mass.sum(axis=0)
         mysum = mass.sum(axis=0)
```

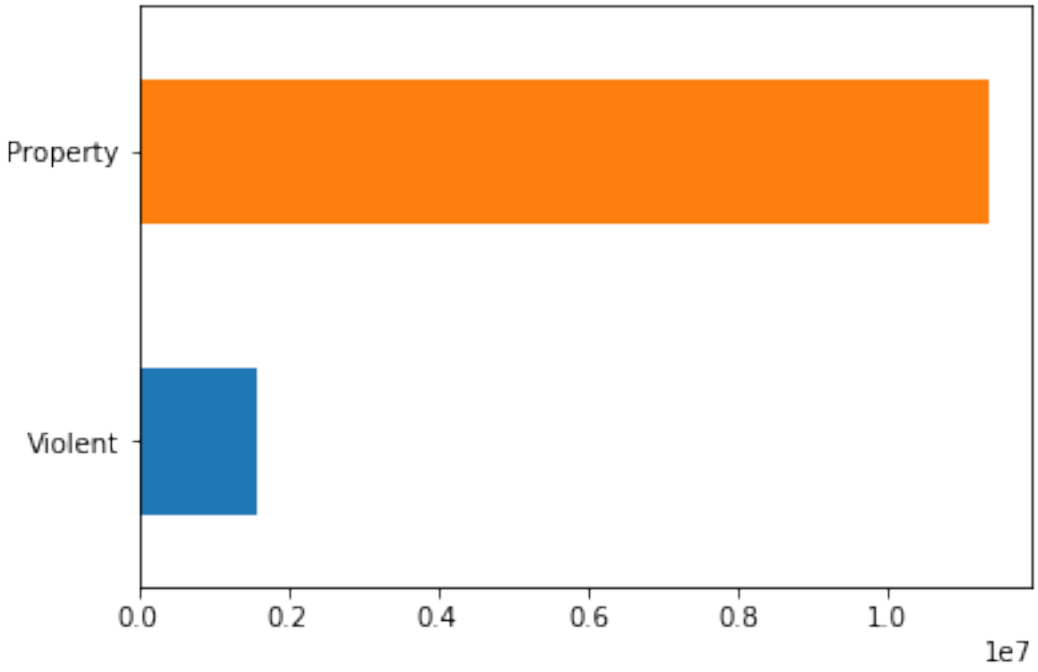
```
mysum
sum1 = mysum[4:]
sum1
ax = sum1.plot.barh(rot=0)
```



This bar plot depicts all the individual crimes committed throughout all the years combined and how many occurences there were for each crime. Out of all the crimes, larceny theft had the highest occurence, followed by burglary and then vehicle theft. Collectively, there is a very small amount of murders.

4.1.2 Barplot of violent vs property crimes:

```
In [26]: #write code for barplot of violent vs property crimes
         #sum the columns for violent and property and plot a bar of the violent and property
         sum2 = mysum[2:4]
         sum2
         ax = sum2.plot.barh(rot=0)
```



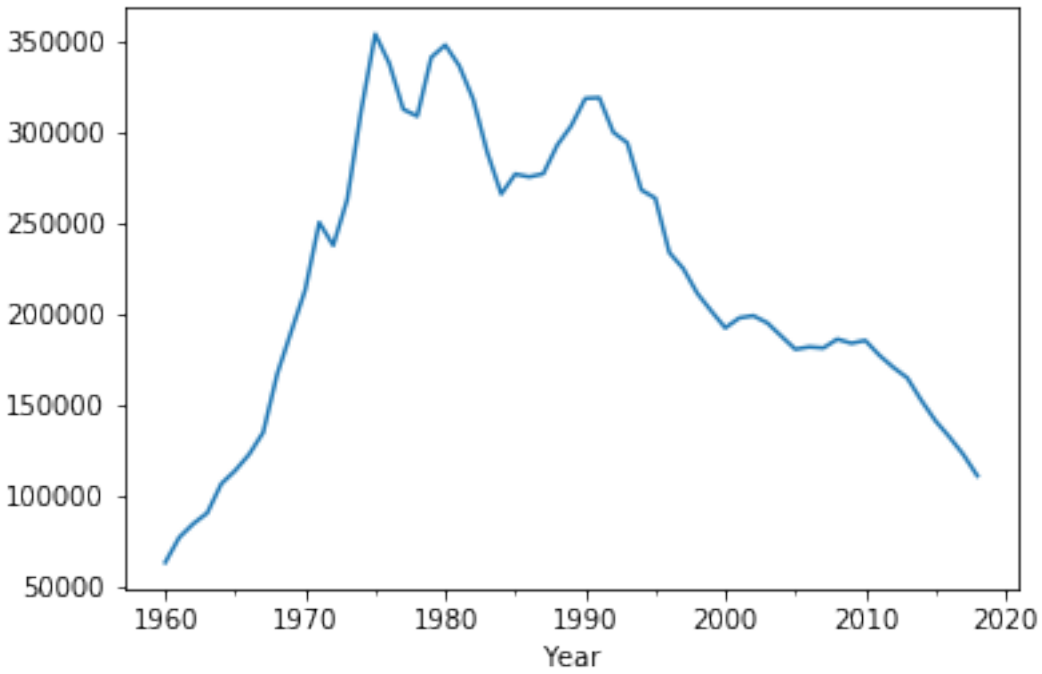
The bar plot depicts the occurences of property and violent crimes for all the years combined. Property crimes out count violent crimes by more than four times. This is likely due to violent crimes having harsher punishments, and property crimes having a higher unsolved rate.

4.2 What years had the most crimes?

4.2.1 How was overall crime throughout the years?

```
In [27]: mass.iloc[:,1].plot()
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1fa50c18>
```

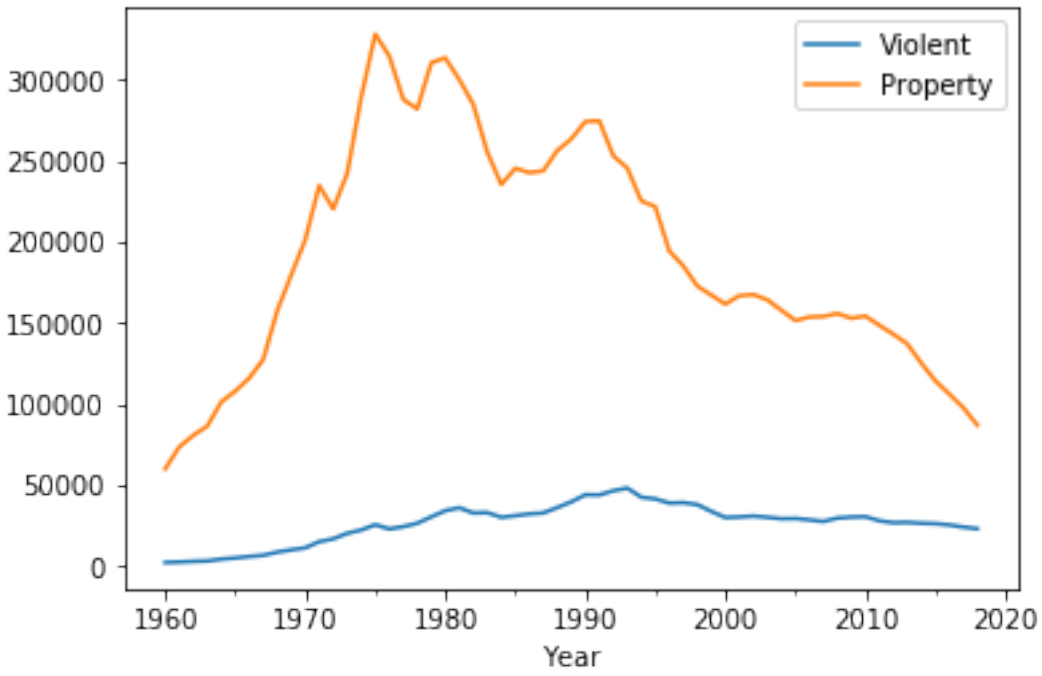


Here, we created a line plot of total crimes to show the transition throughout the years. The number kept going up from 1960 and around 1975 we had the most crimes. Then around 1990, the crime rate suddenly starts to drop without spiking back up again.

4.2.2 How did violent and property crimes change throughout the years?

```
In [28]: #write code for a bar plot of violent and property crimes for every year. Each year s
         mass.iloc[:,2:4].plot()
```

```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1fb42f28>
```

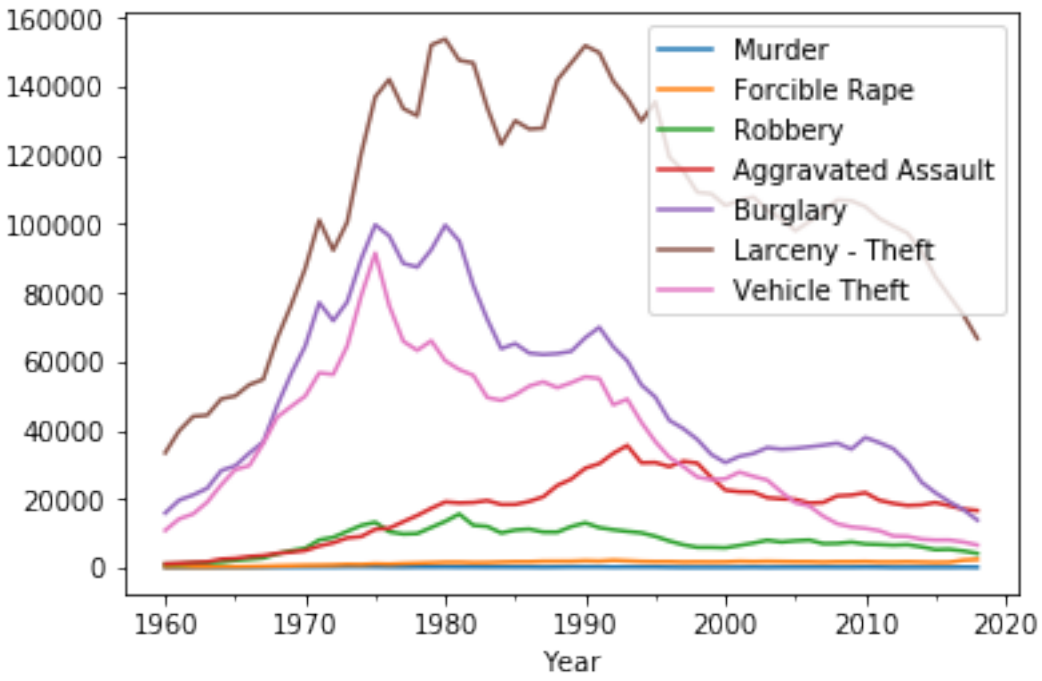


We created a line plot to see how violent and property crimes changed over the years. You can see that property crimes follows the same trend as total crime. However, it is because the number of property crimes is much bigger than that of violent. Propert crime spikes a lot throughout the years while violent crime tends to have a more stable and straight line.

4.2.3 Do different crimes have different patterns throughout the years?

```
In [29]: #draw a line plot of all the columns for murder, forcible rape, robbery, aggravated a
         # vehicle theft over the years. One line for each column.
         mass.iloc[:,4:].plot()
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1f94ac50>
```



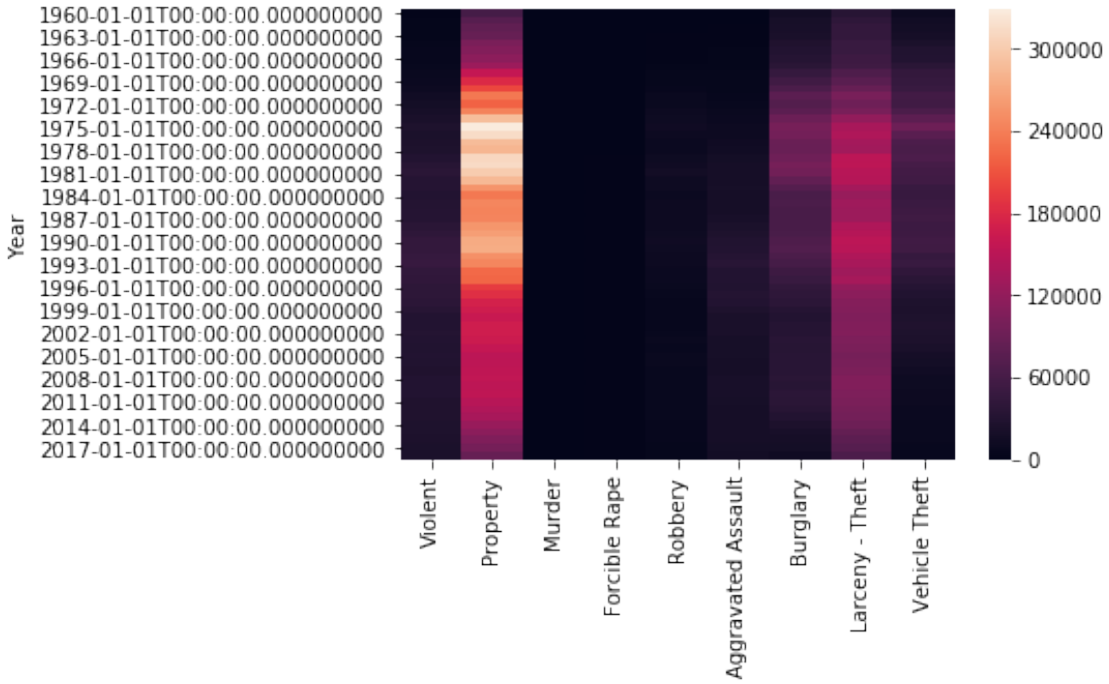
We generated a plot to see if different crimes have different patterns over time. We can observe that top 3 crimes: larceny theft, burglary and vehicle theft are all following the same pattern. The crimes are very likely correlated strongly with each other, and all have many spikes in time where it is occuring more often. On the other hand, crimes like forcible rape and murder are practically straight lines throughout the years. These crimes seem to not follow any pattern. Aggravated assualt and robbery seem to spike a very tiny amount compared to the top three property crimes but generally dont change too much either.

4.2.4 Heatmap of crimes vs years:

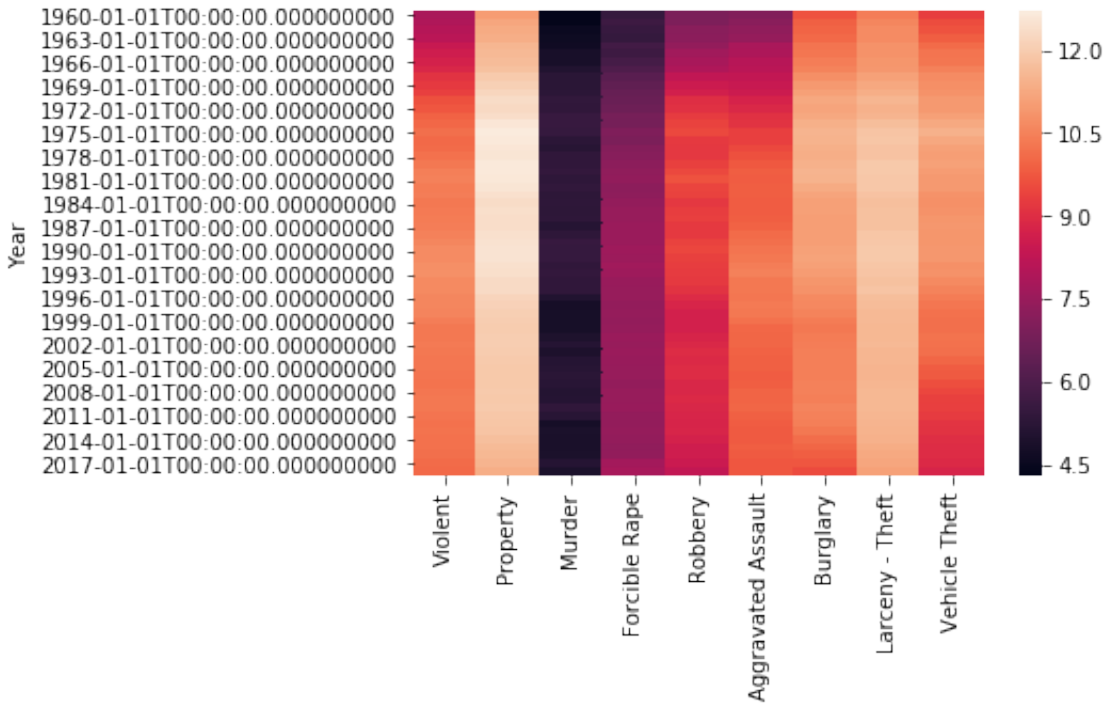
Below, we have three heat maps showing the volume of Massachusetts crime over time. In the first heat map, property-related crimes take up the highest density of crimes from 1960 until present, so I zoomed in on burglaries, robberies, larceny, and vehicular theft in a second heat map. There, I found larceny to have the highest volume of the four theft-related crimes. I then zoomed in again to those four crimes within the time interval when the Boston data was gathered, and this conclusion was held true even in the time era from 1966 to 1975. We also repeated this for log-transformations of the data to achieve clearer results, and we found the exact same things as without the log- transformations. New findings from the log-transform include the fact that of all crimes, murder and forcible rape had the lowest and second lowest volumes.

```
In [30]: mass_l = mass.apply(np.log)
         rob_l = rob.apply(np.log)
         lar_l = lar.apply(np.log)
         veh_l = veh.apply(np.log)
         bur_l = bur.apply(np.log)
```

```
In [31]: mass_heat_1 = mass.iloc[:,2:11]
         sns.heatmap(mass_heat_1)

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1fc79dd8>
```

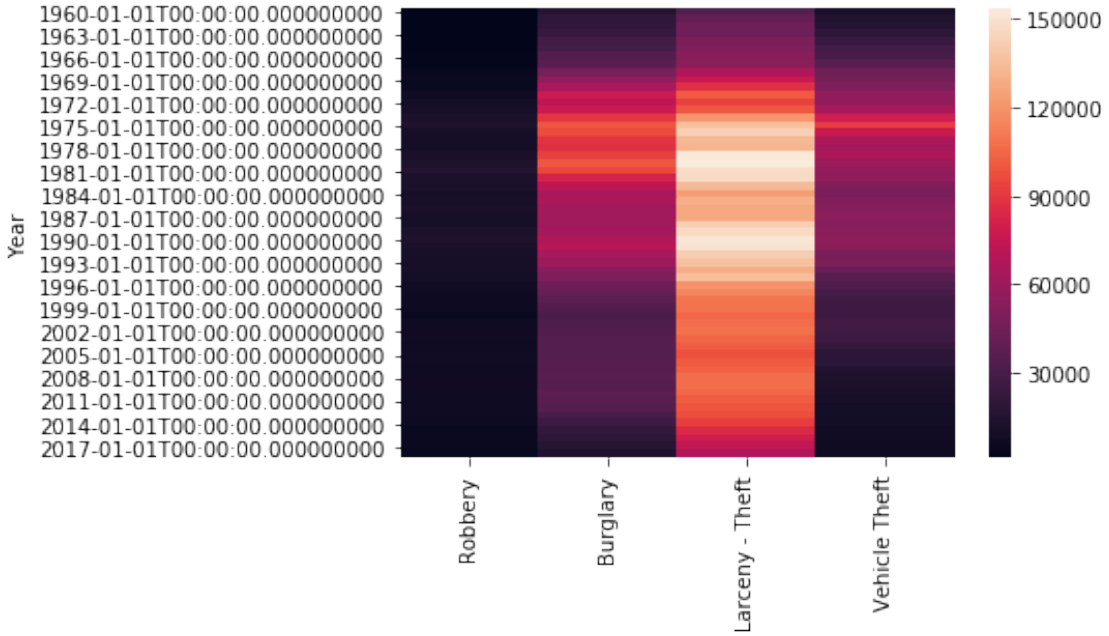



```
In [32]: mass_l_heat_1 = mass_l.iloc[:,2:11]
         sns.heatmap(mass_l_heat_1)

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1fd77240>
```



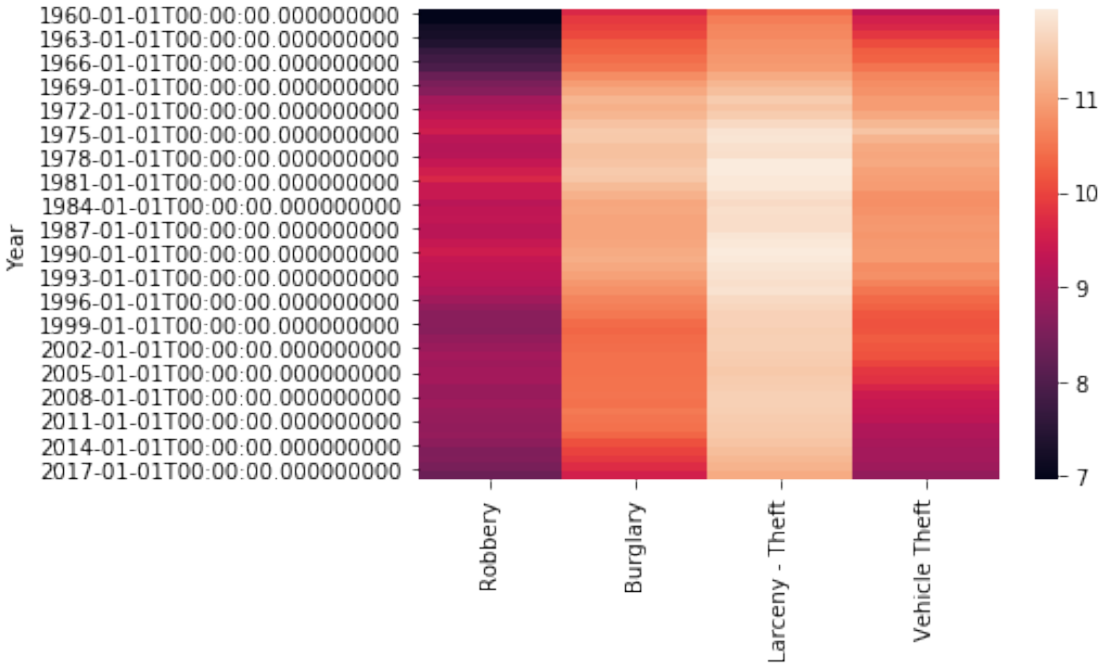

```
In [33]: mass_heat_2 = mass.iloc[:,[6,8,9,10]]
         sns.heatmap(mass_heat_2)
```

```
Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1feff438>
```



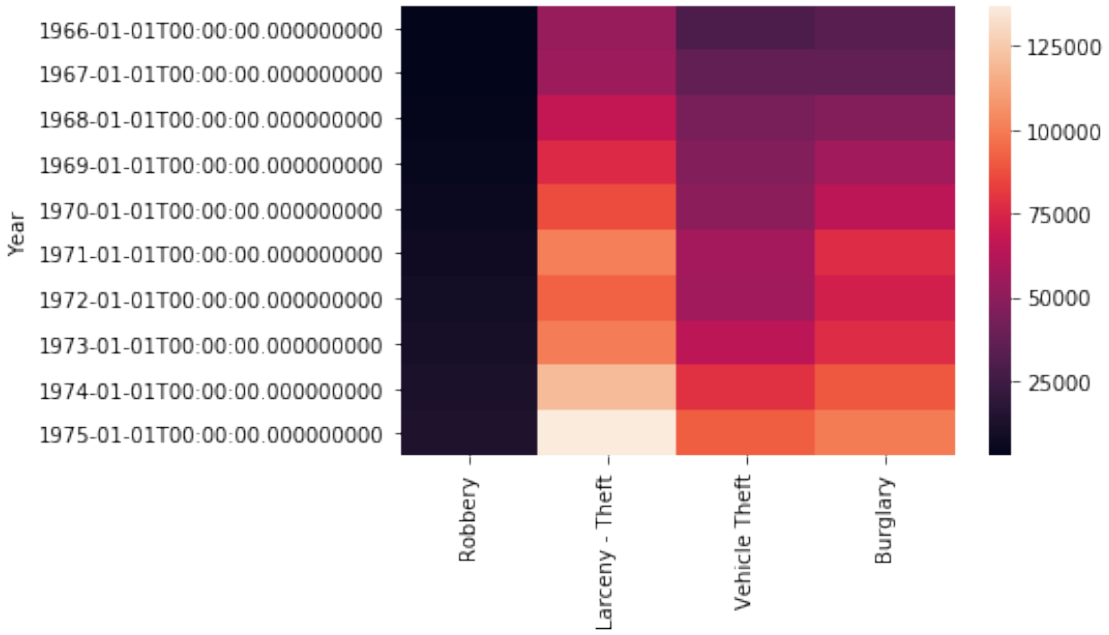
In [34]:
```
mass_l_heat_2 = mass_l.iloc[:,[6,8,9,10]]
sns.heatmap(mass_l_heat_2)
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1c20086908>



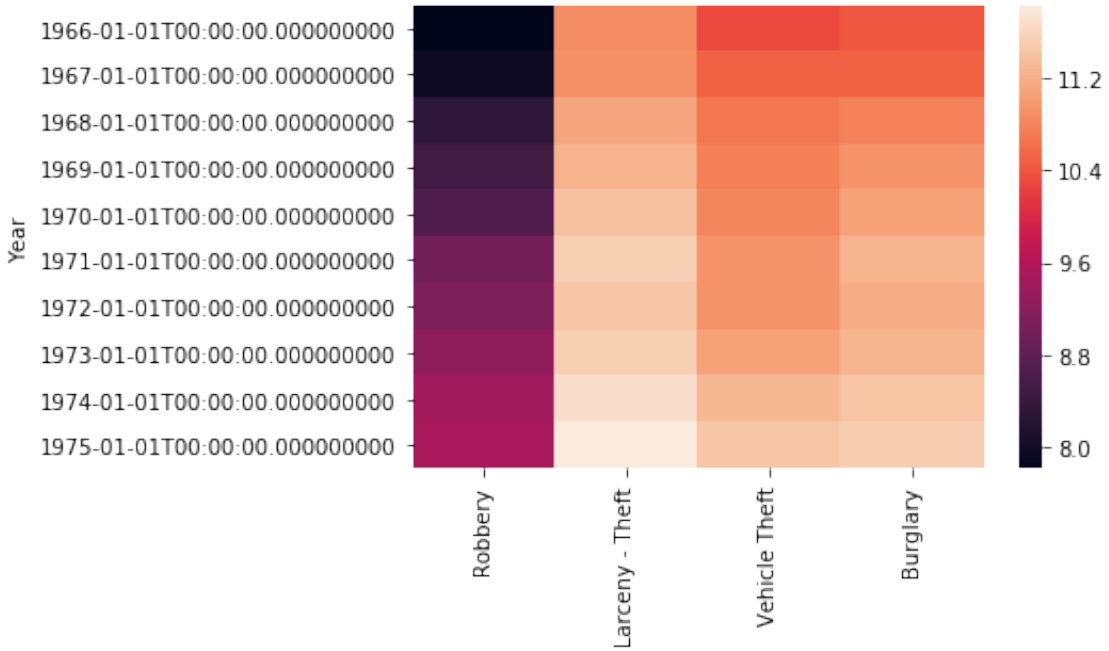
In [35]:
```
df = pd.DataFrame([rob,lar,veh,bur]).transpose()
sns.heatmap(df)
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1c201ecf28>



In [36]:
```
df = pd.DataFrame([rob_l,lar_l,veh_l,bur_l]).transpose()
sns.heatmap(df)
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1c20351160>



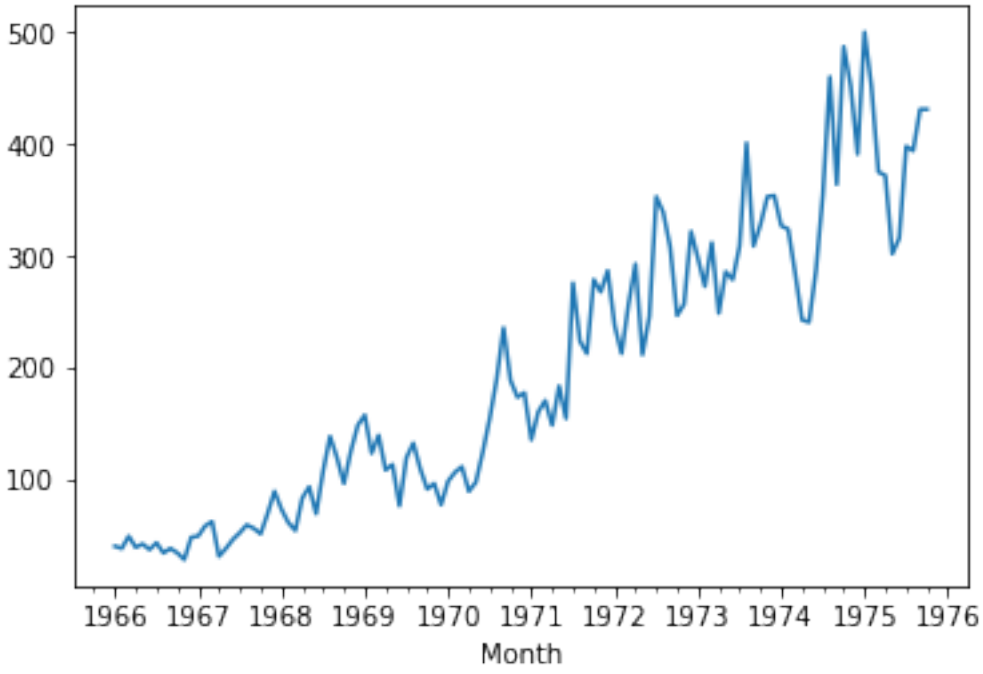
5 Time Series

First, we upload a csv file of our data. Set header and index_col to 0 so no rows are skipped and the data column is used as the index for our analysis. We also want to ignore warnings in the future so we can focus on our code and our results.

```
In [37]: robberies = pd.read_csv('robberies.csv', header=0, index_col=0, parse_dates=True, squ
         warnings.filterwarnings("ignore")
```

After uploading our dataset to the kernel, we plotted our time series as seen below. From what we can gather from the visual, it seems as though the volume of robberies is increasing over time, with fluctuations as time passes.

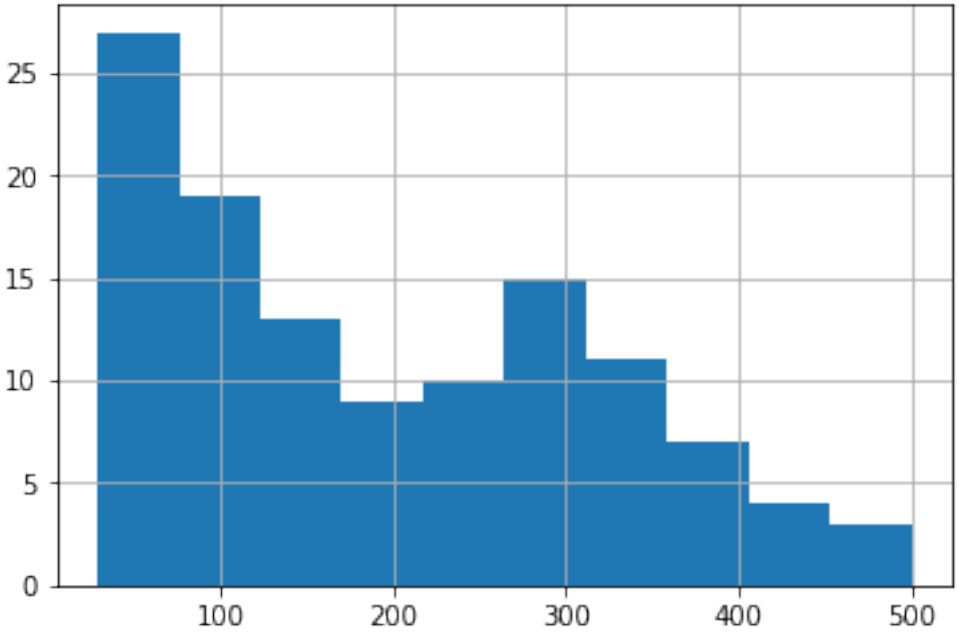
```
In [38]: robberies.plot()
         pyplot.show()
```



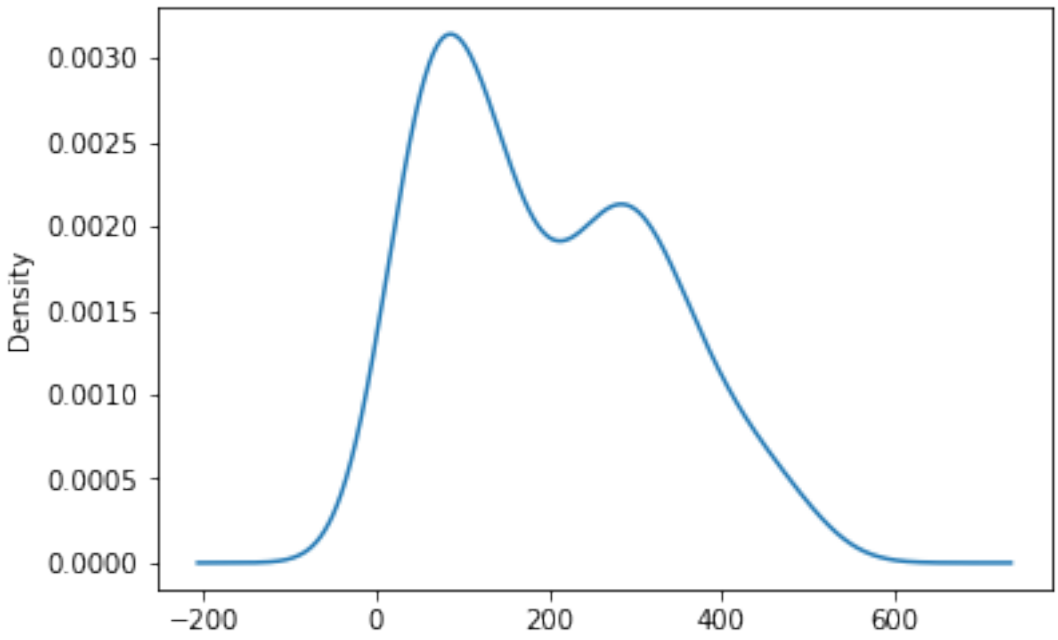
After plotting the time series proper, we then plot a barplot of the time series data, which illustrates how frequencies for armed robberies in Boston are frontloaded in that our largest quantities are in the first and second bins. Afterwards, the next highest point in our data is at around the 300-crime mark. Immediately afterwards, we convert this barplot into a kernel density estimate of our time series, which shows what is likely a bimodal distribution on the grounds that peaks do show up at around the 100-crime mark and 300-crime mark as discussed earlier. This all means that there are more months that had less than 100 robberies committed in the month of Boston than there is any other number of robberies in the set of months we are looking at. This is significant because it shows how a good portion of our time series (and thus much of the interval between 1966 and 1975) is spent within a marginally low number of robberies compared to the rest of the

study time. By looking at the time series, we see the number of robberies hover at a volume of about 100 robberies per month, and then spike up to about 300 robberies per month at about the year of 1970. Such is then reflected in the barplot and kernel density estimate, which then means that if we were to pull a month from our data set based on its robbery volume, we would most likely end up pulling a month with a low number of robberies (less than 100) or a month with around 300 robberies than any other month in our dataset.

```
In [39]: robberies.hist()
         pyplot.show()
```




```
In [40]: robberies.plot(kind='kde')
         pyplot.show()
```



After deriving a kernel density estimate of our data from our barplot, we then resort to using the Augmented Dickey-Fuller Unit Root Test and the Kwiatkowski-Phillips-Schmidt-Shin Test to see whether or not our data is seasonal. The ADF test concludes a lack of stationarity, and the KPSS concludes the existence of stationarity. Thus, we wil perform further tests and what not to sort this out. The ADF test examines whether or not a unit root is present in the time series sample. By us failing to reject the null hypothesis, we are saying this dataset is probably not stationary (and thus there likely does not exists a unit root in the time series characteristic equation). The KPSS test concludes the exact opposite, and thus further exploration is necessary.

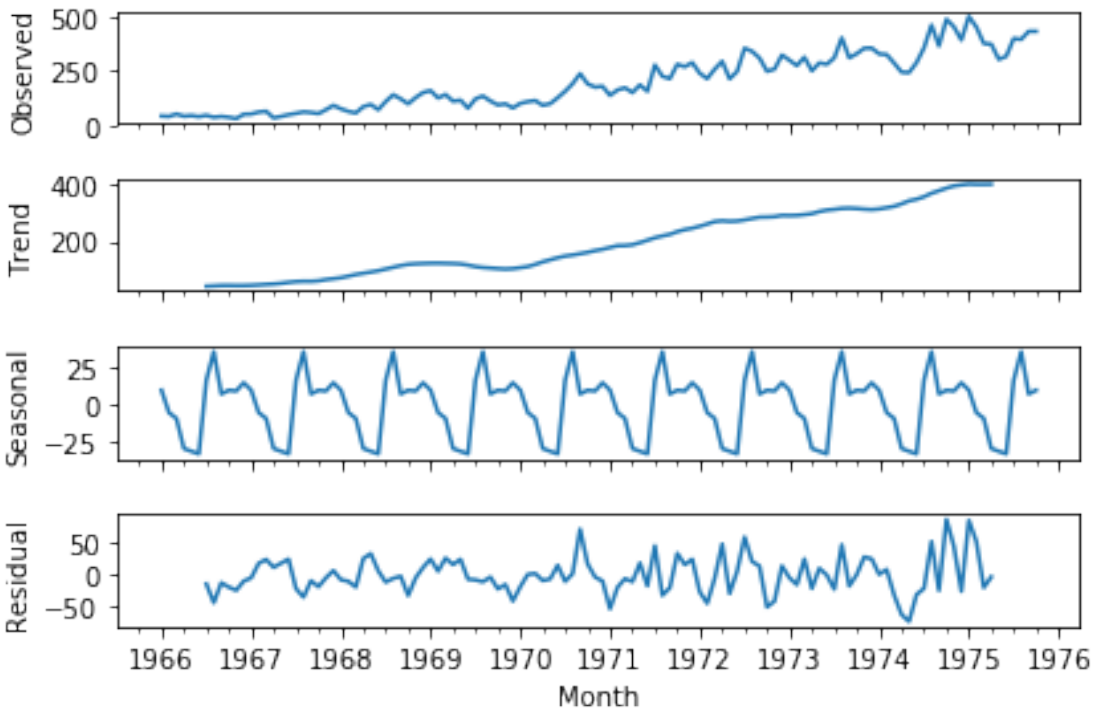
```
In [41]: def ts(data):
             print('Augmented Dickey-Fuller Unit Root Test')
             stat, p, lags, obs, crit, t = adfuller(data)
             print('stat=%.3f, p=%.3f' % (stat, p))

             if p > 0.05:
                 print('Fail to Reject H0, Probably not Stationary')
             else:
                 print('Reject H0, Probably Stationary')

             print('----------------------------------------')
             print(' ')

             print('Kwiatkowski-Phillips-Schmidt-Shin Test')
             stat, p, lags, crit = kpss(data)
             print('stat=%.3f, p=%.3f' % (stat, p))
```

```
            if p > 0.05:
                print('Fail to Reject H0, Probably not Stationary')
            else:
                print('Reject H0, Probably Stationary')

        ts(robberies)
```

```
Augmented Dickey-Fuller Unit Root Test
stat=1.001, p=0.994
Fail to Reject H0, Probably not Stationary
---------------------------------------

Kwiatkowski-Phillips-Schmidt-Shin Test
stat=0.922, p=0.010
Reject H0, Probably Stationary
```

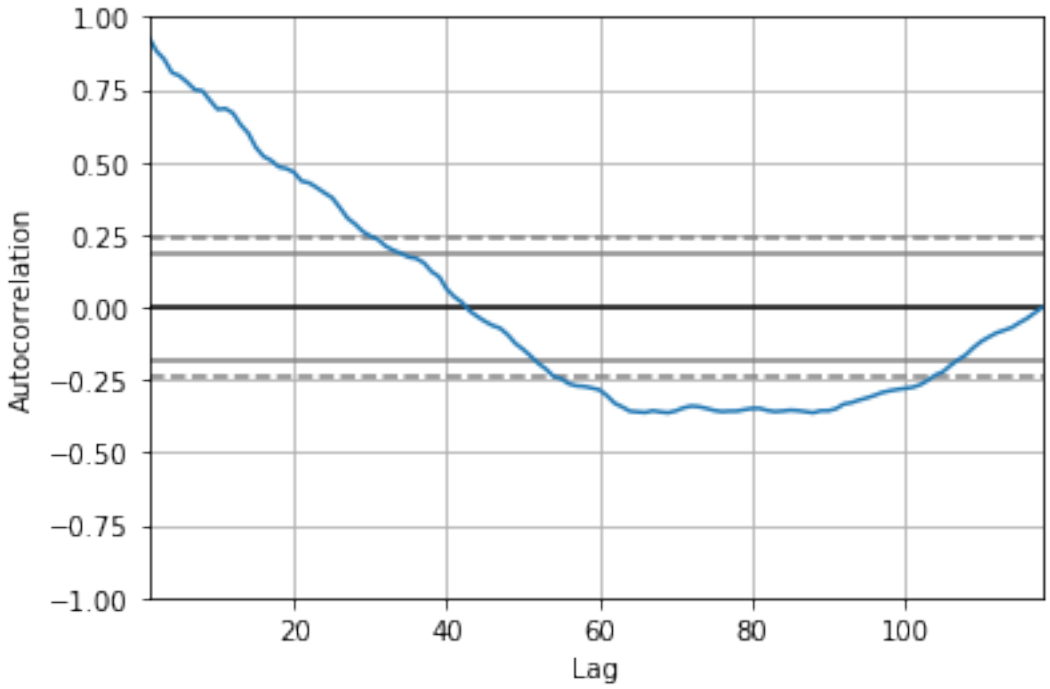
Below, we decompose our time series to it's bare components. First, we set our model to additive since we believe the components to this model (observed, trend, seasonality, and residual) are being added together due to the fact that our data is based on monthly rotations of our data, and thus our seasonality is running on the same width and amplitudes for its cycles. Moreover, our decomposition shows the time series itself, along with a positive trend, a repeating seasonality that cycles at about every 12 months, and residuals which we will test later. The seasonality plot shows a clear dip during the winter and spring months, followed by a spike that begins in the June-July date range in each year (this could just be visually examined by using the years and tick marks on the x-axis of our plots as reference). My best guess for as to why this is happening relies on the fact that there are multiple universities in the Boston area along with much travel and a highly-mobile population that needs to commute to get to work. Thus we see a spike in armed robberies when summer is about to happen since a large sum of people might be traveling to or through the area to see family, only for crime to partially decrease as students go back to studying and then drop off during the winter months when it is usually too cold to be outside.

```
In [42]: result = seasonal_decompose(robberies, model='additive', freq=12)
         result.plot()
         pyplot.show()
```



Below, we see our autocorrelation plot for our time series. Our autocorrelation plot shows values that start off close to one, but then dip down close to zero at about lag 30, actually reach zero at about lag 42, exit our confidence interval at about lag 47, re-enter the confidence interval at about lag 105, and then reach zero again at abiut lag 120. This means that there is not much of a correlation between time series values for observations between lags 30 and 42, along with observations after lag 105. Observations up to lag 30 and between lags 47 and 105 do have a correlation between time series values. Do keep in mind that every ten lags maps out to about a year in time. Thus, we can say that this time series is mostly uncorrelated, and this is actually important because it opens up an avenue where one can say data in this time series is actually independent and identically distributed, which is important for later tests.

```
In [43]: autocorrelation_plot(robberies)
         pyplot.show()
```



After plotting out the autocorrelation visual, we then resort to actually building our model. We set up our code such that the autocorrelation, moving averge, and differencing factors are all between 0 and 3. By using SARIMAX, we are able to compute Aikake information criterion (AIC) for all possible models. After creating our for-loop and then accounting for our parameters, we arrive at the fact that ARIMA(2,2,2) * (2,2,1,12)12 has the lowest AIC quantity at about 729.3. This means our autoregressive, differencing, and moving average parameters are all the same. Specifically, our model would then have exactly two autoregressive terms, two lagged forecast errors, and two instances where we must difference our data in order to achieve stationarity. Earlier, we were in conflict with whether or not our data was stationary. The presence of a non-zero differencing factor indicates our data was not stationary. Lastly, the (2,2,1,12)12 component accounts for seasonality in our data on a 12-month rotation. Thus, we move forward with this as our model.

```
In [44]: p = d = q = range(0, 3)
         pdq = list(itertools.product(p, d, q))
         seasonal_pdq = [(x[0], x[1], x[2], 12) for x in pdq]

         for param in pdq:
             for param_seasonal in seasonal_pdq:
                 try:
                     mod = sm.tsa.statespace.SARIMAX(robberies,
                                                     order=param,
                                                     seasonal_order=param_seasonal,
                                                     enforce_stationarity=False,
                                                     enforce_invertibility=False)
                     results = mod.fit()
```

```
                #print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal,results.aic))
            except:
                continue

    # Using MS Word and Control+F, I found the following model to have the lowest AIC:
    # ARIMA(2, 2, 2)x(2, 2, 1, 12)12 - AIC:729.3023027121975
```

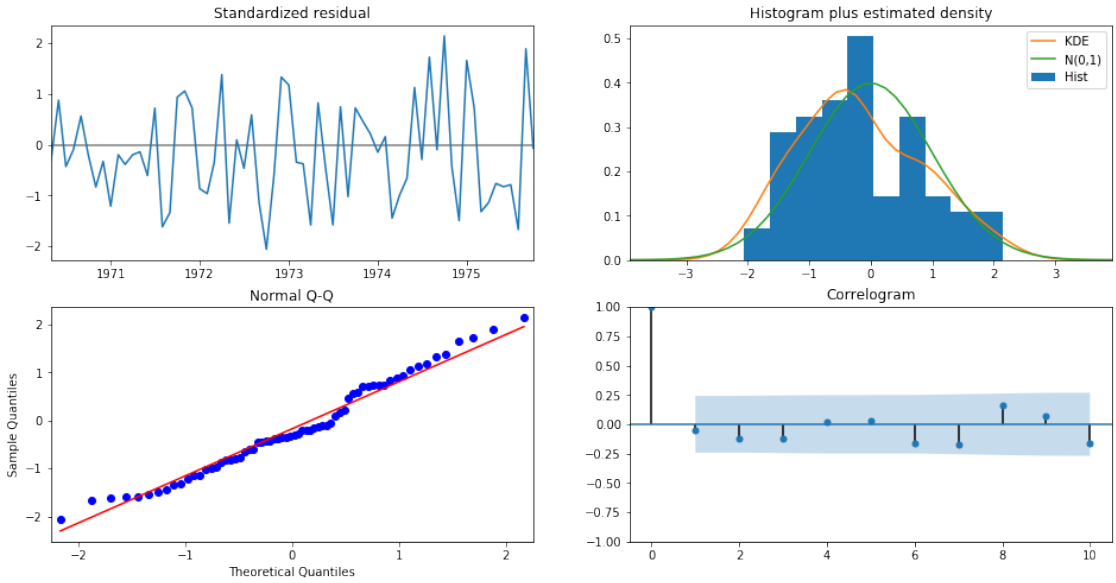
After deriving the model we would like to test, we actually go ahead and test it. From our results below, we can see confidence intervals and z-tests for our different parameters. All autoregressive terms are statistically significant, and all moving average terms are statistically insignificant. I did independently test a few other models to see whether the moving-average factors could be made statistically significant. While they could be made to have a p-value of less than .05, the AIC and BIC criterion spike up too high for me to be comfortable using those as our model, and thus we wil proceed with the chosen model anyhow.

```
In [45]: mod = sm.tsa.statespace.SARIMAX(robberies,
                                         order=(2, 2, 2),
                                         seasonal_order=(2, 2, 1, 12),
                                         enforce_stationarity=False,
                                         enforce_invertibility=False)
         results = mod.fit()
         print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.0181      0.292     -3.483      0.000      -1.591      -0.445
ar.L2         -0.5189      0.135     -3.834      0.000      -0.784      -0.254
ma.L1         -0.4958    566.283     -0.001      0.999   -1110.390    1109.399
ma.L2         -0.5043    285.431     -0.002      0.999    -559.939     558.930
ar.S.L12      -0.5904      0.187     -3.162      0.002      -0.956      -0.224
ar.S.L24      -0.4002      0.169     -2.372      0.018      -0.731      -0.069
ma.S.L12      -1.0001    566.163     -0.002      0.999   -1110.659    1108.659
sigma2      1971.0198      0.033      6e+04      0.000    1970.955    1971.084
==============================================================================
```

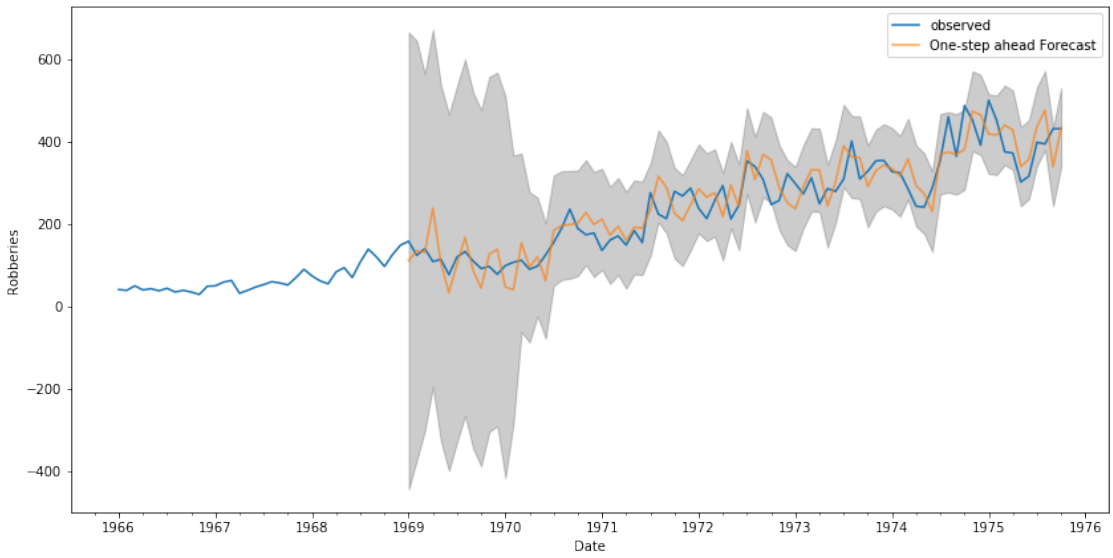
After verifying our model above, we then verify the residuals of that model and see if they fit a normal distribution. From what we can see on the histrogram and normal Q-Q plot below, it seems as if our residuals do follow a normal distribution, and thus our model checks out in this regard.

```
In [46]: results.plot_diagnostics(figsize=(16, 8))
         plt.show()
```



After verifying our model, we then do a one-step ahead forecast on the original time series. To do this, we set up our code to use data starting at January 1, 1966 and ending on January 1, 1969 to make these predictions. Afterwards, we project out the rest of the time series and find that the projected and observed data actually do match up totally when the confidence interval is accounted for. This becomes useful when we go and forecast ahead of the actual time series itself as we know what kind of model works.

```
In [47]: pred = results.get_prediction(start=pd.to_datetime('1969-01-01'), dynamic=False)
         pred_ci = pred.conf_int()
         ax = robberies['1966-01-01':].plot(label='observed')
         pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, figsize=(14
         ax.fill_between(pred_ci.index,
                         pred_ci.iloc[:, 0],
                         pred_ci.iloc[:, 1], color='k', alpha=.2)

         ax.set_xlabel('Date')
         ax.set_ylabel('Robberies')
         plt.legend()
         plt.show()
```



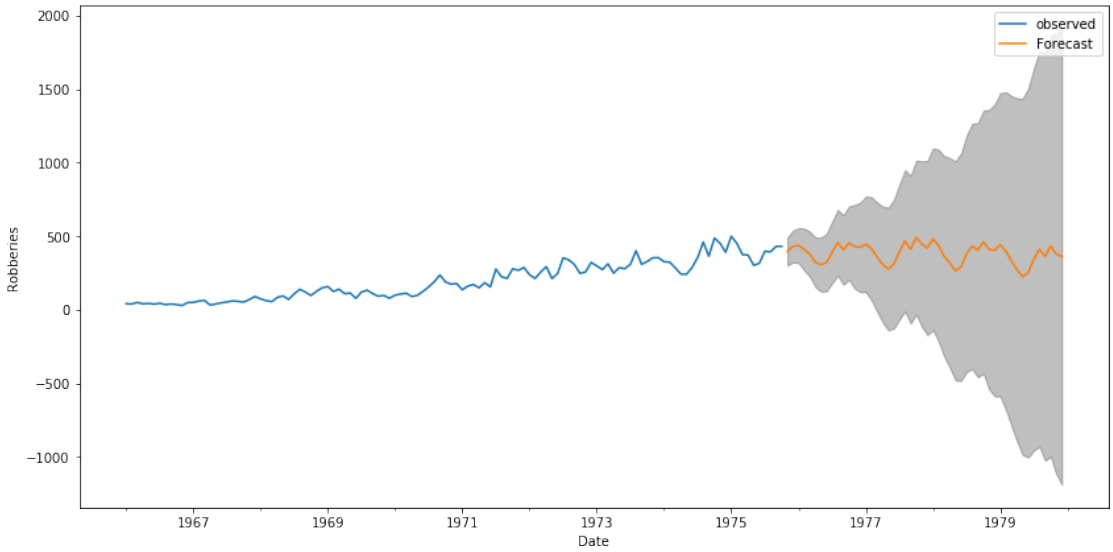
After making an initial projection from only part of our data, we are then able to gather statistics about that projection. Specifically, we have our mean squared error and root mean squared error below to show the predictive potency of a one-step ahead forecast. Values closer to zero are usually better.

```
In [48]: robberies_forecasted = pred.predicted_mean
         robberies_truth = robberies['1970-01-01':]
         mse = ((robberies_forecasted - robberies_truth) ** 2).mean()
         print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
         print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse),

The Mean Squared Error of our forecasts is 2708.48
The Root Mean Squared Error of our forecasts is 52.04
```

We wrap up our time series analysis by actually projecting 50 months of armed robbery data in the city of Boston. We can see a repeating pattern in our projection similar to that of the seasonality plot in our decomposition model from earlier. Overall, it seems as though the number of armed robberies in Boston stabilizes over time.

```
In [49]: pred_uc = results.get_forecast(steps=50)
         pred_ci = pred_uc.conf_int()
         ax = robberies.plot(label='observed', figsize=(14, 7))
         pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
         ax.fill_between(pred_ci.index,
         pred_ci.iloc[:, 0],
         pred_ci.iloc[:, 1], color='k', alpha=.25)

         ax.set_xlabel('Date')
         ax.set_ylabel('Robberies')
```

```
7

plt.legend()
plt.show()
```



Lastly, we test for any relationships between Boston robberies and burglaries, robberies, larceny, and vehicle theft in Massachusetts. All tests show some form of relationship between Boston and Massachusetts crime in that era. This means either Boston was a large enough chunk of Massachusett's population at the time to be a leading statistical parameter in terms of predicting crime, or the rest of the state was just behaving like Boston in that time interval. Specifically, the Pearson's, Spearman's, and Kendall's correlation tests are all to decide whether or not two sets of data have some form of correlation. Pearson's and Spearman's tests both rely on the covariance of both datasets divided by the product of their standard deviations as test statistics. The closer the test statistic is to the absolute value of one, the stronger the correlation is between both datasets. In our tests below, every instance of both of these tests has a high correlation statistics (all greater than 0.90), and thus we can infer a strong correlation between Boston robberies and theft-related crimes in all of Massachusetts. Lastly, Kendall's correlation test decides the same thing but in a completely different way. Specifically, we calculated tau statistics from subtracting the number of discordant pairs (where ranks between both samples do not correspond) from the number of concordant pairs, and then dividing by the binomial coeffient "n choose 2." We then utilized the p-value to draw the same conclusion as we did with the Pearson' and Spearman's tests.

```
In [50]: b_group = robberies.resample('AS').sum().tolist()

In [51]: def test(data1, data2):
             print('Pearsons Correlation Coefficient Test')
             stat, p = scipy.stats.pearsonr(data1, data2)
             print('stat=%.3f, p=%.3f' % (stat, p))
```

```
            if p > 0.05:
                print('Fail to reject H0, probably independent')
            else:
                print('Reject H0, probably dependent')

            print('                                  ')
            print('Spearmans Rank Correlation Test')
            stat2, p2 = scipy.stats.spearmanr(data1, data2)
            print('stat=%.3f, p=%.3f' % (stat2, p2))

            if p2 > 0.05:
                print('Fail to reject H0, probably independent')
            else:
                print('Reject H0, probably dependent')

            print('                                  ')
            print('Kendalls Rank Correlation Test')
            stat3, p3, dof, expected = scipy.stats.chi2_contingency([data1, data2])
            print('stat=%.3f, p=%.3f' % (stat3, p3))

            if p3 > 0.05:
                print('Fail to reject H0, probably independent')
            else:
                print('Reject H0, probably dependent')

            print('_____________________________________')
            print('                                  ')

        test(b_group, rob)
        test(b_group, lar)
        test(b_group, veh)
        test(b_group, bur)
```

```
Pearsons Correlation Coefficient Test
stat=0.986, p=0.000
Reject H0, probably dependent

Spearmans Rank Correlation Test
stat=0.988, p=0.000
Reject H0, probably dependent

Kendalls Rank Correlation Test
stat=296.121, p=0.000
Reject H0, probably dependent
---------------------------------

Pearsons Correlation Coefficient Test
stat=0.927, p=0.000
```

```
Reject H0, probably dependent

Spearmans Rank Correlation Test
stat=0.952, p=0.000
Reject H0, probably dependent

Kendalls Rank Correlation Test
stat=2664.934, p=0.000
Reject H0, probably dependent
---------------------------------

Pearsons Correlation Coefficient Test
stat=0.923, p=0.000
Reject H0, probably dependent

Spearmans Rank Correlation Test
stat=0.976, p=0.000
Reject H0, probably dependent

Kendalls Rank Correlation Test
stat=2368.351, p=0.000
Reject H0, probably dependent
---------------------------------

Pearsons Correlation Coefficient Test
stat=0.945, p=0.000
Reject H0, probably dependent

Spearmans Rank Correlation Test
stat=0.976, p=0.000
Reject H0, probably dependent

Kendalls Rank Correlation Test
stat=2121.361, p=0.000
Reject H0, probably dependent
---------------------------------
```

6 Conclusion

In conclusion, we tested Massachusetts data for crime volume over time, and then zoomed in on the city of Boston to further examine the dynamic armed robberies and theft-related crimes in that era took as time progressed. Specifically, we modeled a time series that specifically covered armed robberies from the city of Boston from 1966 to 1975, and then statistically and visually related it to the corresponding Massachusetts data. Our results show there was some kind of a relationship between the urban areas of Massachusetts (like Boston) and the rural areas that aren't as densely

populated. Whether this relationship is that the rest of the state behaved like Boston, or that Boston was simply so highly populated that it statistically accounted for too much of the state to not have a relationship remains to be seen. This does speak to the urban-rural dynamic here in America, where a state might seem violent when all the crime happens in its metropolitan areas only for the rural areas to actually be peaceful. In that scenario, it would be that the cities would be so highly populated that in order to perform further meaningful analysis, we would need data on those rural areas. This, in addition to not having any economic, social, cultural, or educational data, are all ways in which this project could have been improved. With the above in mind, we would like to thank Prof. Oh for giving us the requisite material to complete this project.

In [ ]:

In [ ]: