Zixiang Tang zt2292 & Chenxi Jiang cj2706

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Tutorial On Commonly Used Plots

Load Dataset Index Crimes by County and Agency: Beginning 1990

In [3]:

```
df = pd.read_csv("https://data.ny.gov/api/views/ca8h-8gjq/rows.csv")
df.head(5) # show first 5 rows
```

Out[3]:

	County	Agency	Year	Months Reported	Index Total	Violent Total	Murder	Rape	Robbery	Aggravated Assault	Property Total	Burglary	Larceny	Moto Vehicl Thef
0	Albany	Albany City PD	1990	NaN	6635.0	1052.0	9.0	82.0	386.0	575.0	5583.0	1884.0	3264.0	435.
1	Albany	Albany City PD	1991	NaN	7569.0	1201.0	11.0	71.0	487.0	632.0	6368.0	1988.0	3878.0	502.
2	Albany	Albany City PD	1992	NaN	7791.0	1150.0	8.0	77.0	467.0	598.0	6641.0	2246.0	3858.0	537.
3	Albany	Albany City PD	1993	NaN	7802.0	1238.0	6.0	59.0	481.0	692.0	6564.0	2063.0	4030.0	471.
4	Albany	Albany City PD	1994	NaN	8648.0	1380.0	13.0	79.0	542.0	746.0	7268.0	2227.0	4502.0	539.

Useful Functions

Groupby

Group by a column and apply a function to the rest of columns

DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=NoDefault.no_default, observed=False, dropna=True)

Often uses by = label or list of labels

Follow by a function (such as sum, mean, etc.)

In [4]:

```
df_group = df.groupby(by=['Year'],dropna=True).sum() # group by year, sum up number in r
```

```
df_group = df.groupby(by = 'Year').sum() # group by Year, sum up number in the
est columns
df_group.reset_index(level=0, inplace=True)
df_group
```

Out[4]:

	Year	Months Reported	Index Total	Violent Total	Murder	Rape	Robbery	Aggravated Assault	Property Total	Burglary	Larceny	Motor Vehicle Theft
0	1990	0.0	2701049.0	461171.0	5573.0	12918.0	236450.0	206230.0	2239878.0	501980.0	1324811.0	413087.0
1	1991	0.0	2698494.0	459696.0	5517.0	12353.0	238429.0	203397.0	2238798.0	499787.0	1335032.0	403979.0
2	1992	0.0	2547943.0	440821.0	5151.0	12563.0	225588.0	197519.0	2107122.0	473771.0	1255525.0	377826.0
3	1993	0.0	2430768.0	424203.0	5212.0	12110.0	213742.0	193139.0	2006565.0	443994.0	1219537.0	343034.0
4	1994	0.0	2226215.0	388386.0	4379.0	11428.0	187051.0	185528.0	1837829.0	403214.0	1144019.0	290596.0
5	1995	0.0	2039612.0	340991.0	3475.0	10330.0	158293.0	168893.0	1698621.0	366455.0	1097087.0	235079.0
6	1996	0.0	1869331.0	297041.0	3007.0	10146.0	135859.0	148029.0	1572290.0	328489.0	1034747.0	209054.0
7	1997	0.0	1771085.0	281864.0	2489.0	10097.0	123696.0	145582.0	1489221.0	303091.0	998824.0	187306.0
8	1998	0.0	1623925.0	262007.0	2136.0	9765.0	108288.0	141818.0	1361918.0	265740.0	937085.0	159093.0
9	1999	0.0	1509190.0	241513.0	2074.0	8606.0	99096.0	131737.0	1267677.0	234058.0	886714.0	146905.0
10	2000	0.0	1467524.0	237722.0	2195.0	8413.0	92226.0	134888.0	1229802.0	218069.0	874644.0	137089.0
11	2001	0.0	1408230.0	227387.0	2264.0	8811.0	84945.0	131367.0	1180843.0	205832.0	851389.0	123622.0
12	2002	7226.0	1366099.0	219954.0	2128.0	9633.0	83609.0	124584.0	1146145.0	199383.0	828842.0	117920.0
13	2003	7747.0	1321013.0	208623.0	2154.0	9701.0	81345.0	115423.0	1112390.0	197958.0	802652.0	111780.0
14	2004	7691.0	1275044.0	197110.0	2100.0	9255.0	75677.0	110078.0	1077934.0	183687.0	792982.0	101265.0
15	2005	7739.0	1251065.0	201949.0	2083.0	9433.0	80290.0	110143.0	1049116.0	180196.0	780696.0	88224.0
16	2006	7845.0	1228079.0	199362.0	2185.0	8424.0	79632.0	109121.0	1028717.0	183324.0	765065.0	80328.0
17	2007	7848.0	921990.0	159636.0	1616.0	5848.0	62072.0	90100.0	762354.0	129668.0	576740.0	55946.0
18	2008	7340.0	730963.0	106452.0	1149.0	4684.0	41300.0	59319.0	624511.0	111161.0	475596.0	37754.0
19	2009	7071.0	711581.0	103303.0	1097.0	4298.0	37529.0	60379.0	608278.0	106574.0	468832.0	32872.0
20	2010	7136.0	712760.0	103185.0	1196.0	4498.0	37202.0	60289.0	609575.0	111464.0	467612.0	30499.0
21	2011	7157.0	706162.0	103725.0	1025.0	4454.0	36933.0	61313.0	602437.0	112021.0	462540.0	27876.0
22	2012	7146.0	708969.0	105607.0	957.0	4498.0	36907.0	63245.0	603362.0	109523.0	467625.0	26214.0
23	2013	7159.0	671035.0	100578.0	955.0	4120.0	35188.0	60315.0	570457.0	95750.0	451123.0	23584.0
24	2014	7102.0	635167.0	95286.0	899.0	4004.0	31407.0	58976.0	539881.0	85416.0	430723.0	23742.0
25	2015	7033.0	602946.0	100226.0	888.0	10058.0	30830.0	58450.0	502720.0	74040.0	405382.0	23298.0
26	2016	7115.0	581572.0	99124.0	927.0	10118.0	29028.0	59051.0	482448.0	67451.0	392690.0	22307.0
27	2017	6996.0	555028.0	94955.0	810.0	10391.0	26139.0	57615.0	460073.0	58640.0	380430.0	21003.0
28	2018	6927.0	523984.0	91013.0	839.0	10632.0	23420.0	56122.0	432971.0	51249.0	361036.0	20686.0
29	2019	6839.0	503506.0	91917.0	821.0	10504.0	22748.0	57844.0	411589.0	45352.0	346355.0	19882.0
30	2020	6743.0	507587.0	93127.0	1204.0	8967.0	21830.0	61126.0	414460.0	50572.0	333948.0	29940.0

group_by is very useful in processing data. It is commonly used to group a column and apply a function to the rest column. For example, in this dataset, to find the crime number each year, use group_by to group Year column and apply sum to the rest to find the crime number each year.

Filter Rows by a specific condition

In [5]:

```
df2=df[df['Year']==2000] # filter all crime taken place in 2000
```

```
df2.head(5)
```

Out[5]:

	County	Agency	Year	Months Reported	Index Total	Violent Total	Murder	Rape	Robbery	Aggravated Assault	Property Total	Burglary	Larceny	M Ve
10	Albany	Albany City PD	2000	NaN	7088.0	1110.0	11.0	66.0	396.0	637.0	5978.0	1513.0	4012.0	
66	Albany	Albany County Park PD	2000	NaN	4.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	
100	Albany	Albany County Sheriff	2000	NaN	128.0	16.0	0.0	0.0	0.0	16.0	112.0	34.0	62.0	
131	Albany	Albany County State Police	2000	NaN	493.0	17.0	0.0	1.0	4.0	12.0	476.0	75.0	392.0	
162	Albany	Altamont Vg PD	2000	NaN	6.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	6.0	

Select Columns by names

In [6]:

```
df2 = df[['County','Rape','Larceny','Burglary','Murder']]
df2.head(5)
```

Out[6]:

	County	Rape	Larceny	Burglary	Murder
0	Albany	82.0	3264.0	1884.0	9.0
1	Albany	71.0	3878.0	1988.0	11.0
2	Albany	77.0	3858.0	2246.0	8.0
3	Albany	59.0	4030.0	2063.0	6.0
4	Albany	79.0	4502.0	2227.0	13.0

Plot

1. Line plot

matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)

Parameters: y vs x

x:float or array-like, x coordinates of plot

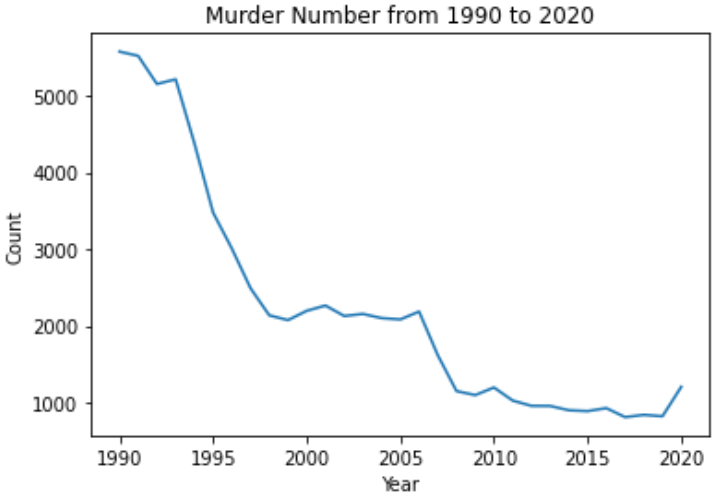
y:float or array-like, y coordinates of plot

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

In [7]:

```
plt.plot(df_group['Year'],df_group['Murder'])
```

```
plt.title("Murder Number from 1990 to 2020")
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```



1. **Scatter Plot**

matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, *, edgecolors=None, plotnonfinite=False, data=None, **kwargs)

Parameters: y vs x

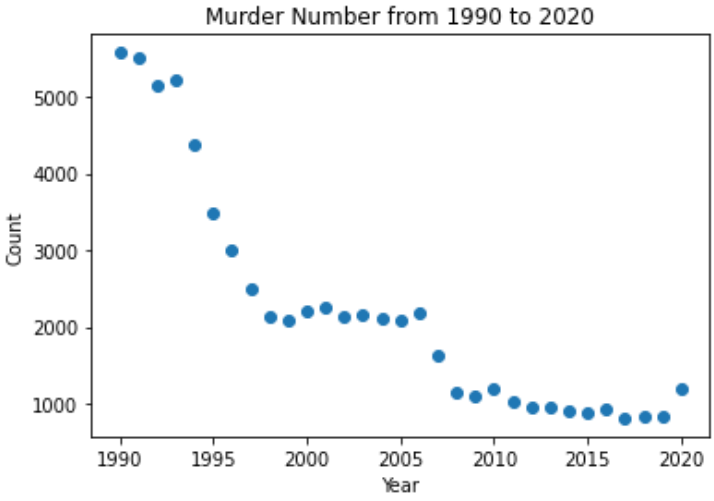
x:float or array-like, x coordinates of plot

y:float or array-like, y coordinates of plot

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

In [8]:

```
plt.scatter(df_group['Year'],df_group['Murder'])
plt.title("Murder Number from 1990 to 2020")
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```



We can see the line plot can indicate the amount of the quantity and clearly shows the situation of the increase or decrease of the quantity. Scatter plot reflects the change pattern of the relationship between variables, which is convenient for deciding which mathematical expression to use to simulate the relationship between variables. Also, scatter plot can transfer the type of relationship between variables Information and the degree of clarity of the relationship between the reflected variables. When we specificly look at the scatter plot for the relationship of year vs. Count. We cannot easily get they are in a linear relationship, but we can conclude they crime count is descending obviously. However, when we look at the line plot, we can observe the distinct decreasing murder in around 1993 to 1998 and 2005 to 2008. From this information, we can explain this situation by aspect of change

of politics, economy, or culture.

1. **Bar plot**

matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)

Parameters:

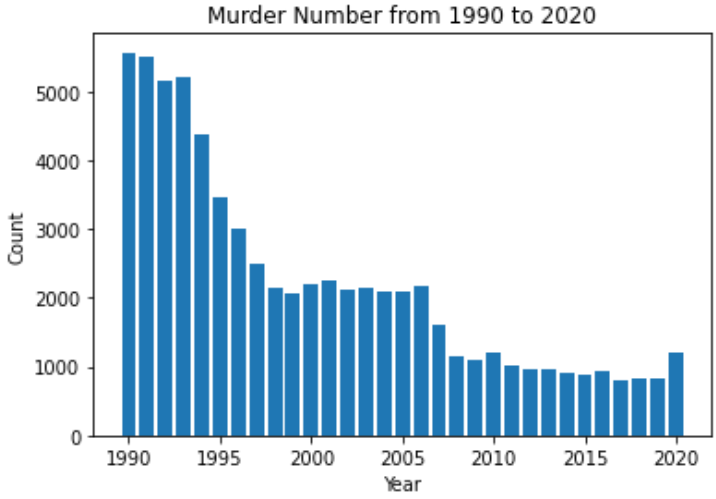
x:float or array-like, x coordinates of plot

y:float or array-like, height of the bar

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

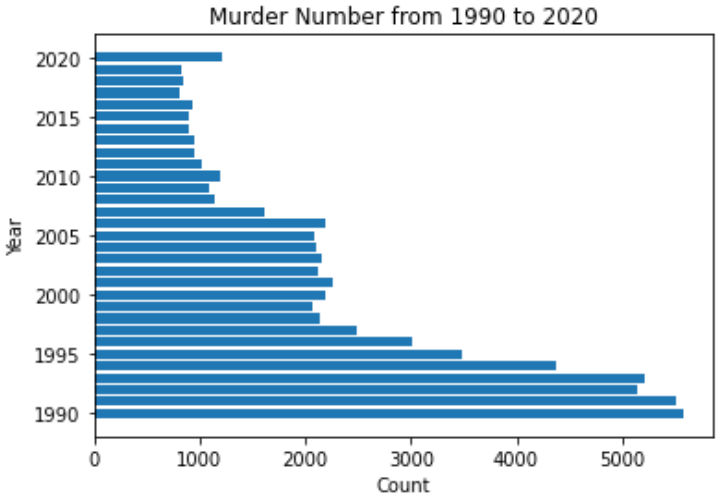
In [9]:

```
plt.bar(df_group['Year'],df_group['Murder'])
plt.title("Murder Number from 1990 to 2020")
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```



In [10]:

```
plt.barh(df_group['Year'],df_group['Murder'])
plt.title("Murder Number from 1990 to 2020")
plt.xlabel('Count')
plt.ylabel('Year')
plt.show()
```



Bar plot is generally used to describe categorical variables. Here, the variable of the horizontal axis is the year, and the variable of the vertical axis is the count of Murder number. An interval of the Bar plot represents a categorical variable, and the interval does not need to be next to each other. Because they are parallel to each other, there is no comparison of size. Bar plot displays the relative numbers or proportions of multiple categories and summarize a large data set in visual form.

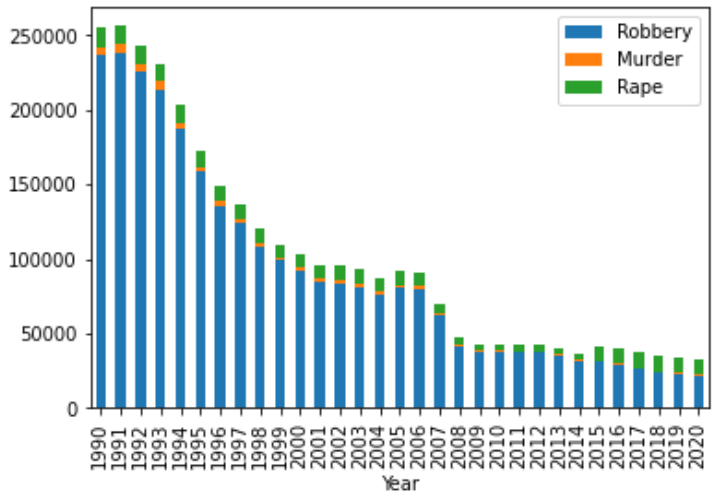
and summarize a large data set in visual form.

Stacked Bar Plot

Stacked Bar Plot is commonly used to compare categories distribution within each type ploted across x coordinates. The plot below shows three crimes distribution each year.

In [11]:

```
df_group.plot.bar(x='Year',y=['Robbery','Murder','Rape'],stacked=True)
plt.show()
```



1. **Histogram**

matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

Parameters:

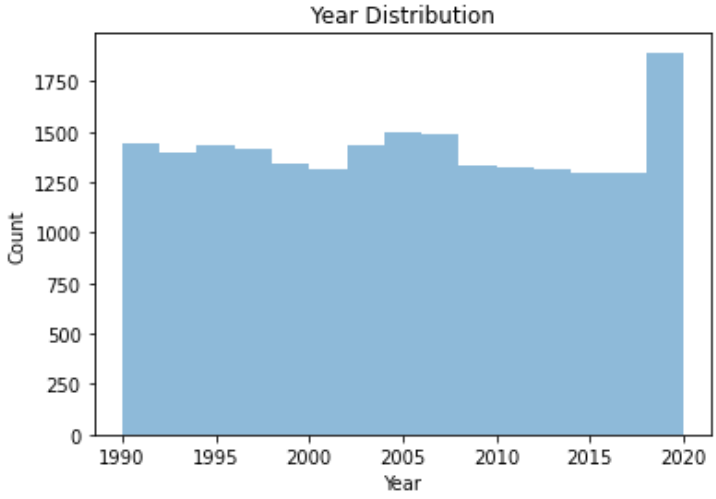
x: array or sequence

bins: can be assigned or use default

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

In [12]:

```
plt.hist(df['Year'],bins=15, alpha=0.5)
plt.title("Year Distribution")
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```



Histogram is generally used to describe numerical variables. The horizontal axis of Histogram uses bins to divide

the variables into a specific interval. Here, year is used as the variable, separated by a length of five years, so the length of a bin is 5, so each interval in the Histogram is next to each other, and each interval can be compared. Like Bar plot, the vertical axis of Histogram also represents the number of variables. Histograms are usually used to present the distribution of variables. Here, the histogram is used to present the year distribution of murder number.

1. **Box Plot**

matplotlib.pyplot.boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, patch_artist=None, bootstrap=None, usermedians=None, conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None, showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None, meanprops=None, capprops=None, whiskerprops=None, manage_ticks=True, autorange=False, zorder=None, *, data=None)

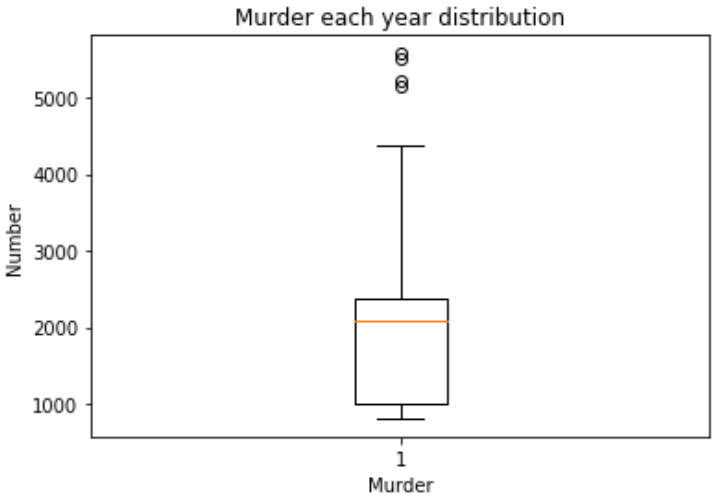
Parameters:

x: array or sequence

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

In [13]:

```
plt.boxplot(df_group['Murder'])
plt.xlabel('Murder')
plt.ylabel('Number')
plt.title('Murder each year distribution')
plt.show()
```



Using box plots makes it easy for us to investigate multiple continuous variables at the same time, or to investigate a single continuous variable in groups, so it is more flexible in use than histograms and has a wider range of uses. The box plot can tell us the value of: Lower limit, Lower Quartile, Median, Upper Quartile, Upper limit, Mild Outlier and Extreme Outlier. Box plot can summarize variation in large datasets visually, show outliers, compares multiple distributions, Indicate symmetry and skewness to a degree.

1. **Violin Plot**

Axes.violinplot(dataset, positions=None, vert=True, widths=0.5, showmeans=False, showextrema=True, showmedians=False, quantiles=None, points=100, bw_method=None, *, data=None)

Parameters:

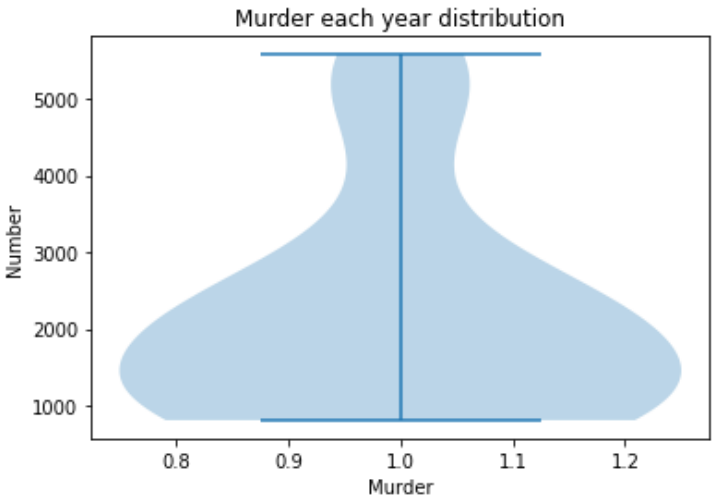
x: array or sequence

Read more on Matplotlib
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.violinplot.html#matplotlib.axes.Axes.violinplot

In [14]:

```
plt.violinplot(df_group['Murder'])
```

```
plt.xlabel('Murder')
plt.ylabel('Number')
plt.title('Murder each year distribution')
plt.show()
```



The violin chart is a combination of the box plot and the density distribution. Similar to box plots, expect that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator. The probability density of the data is expressed by the width of the violin. From the figure, the violin plot reflects the dispersion density more clearly than the box plot. We can see the distribution of Murder number in each interval, this figure shows that most of the Murder number is distributed in the interval of 1000-2000 each year, which means that the probability of data distribution in this interval is the greatest.

1. **Pie chart**

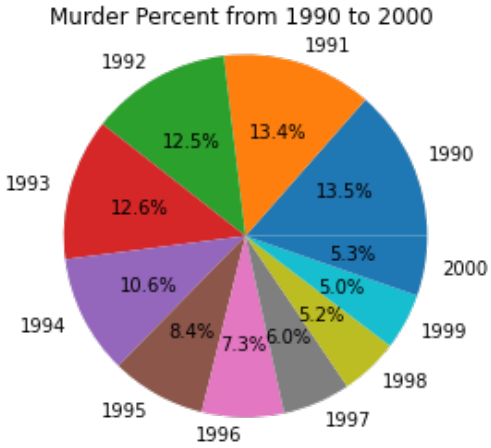
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, rotatelabels=False, *, normalize=None, data=None)

x: an array or sequence

Read more on Matplotlib https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html

In [15]:

```
df_pie = df_group.head(11)
plt.pie(df_pie['Murder'],labels = df_pie['Year'],autopct='%1.1f%%')
plt.axis('equal')
plt.title('Murder Percent from 1990 to 2000 ')
plt.show()
```



Pie chart reflects the related parts of different data groups or categories. It shows users what a fraction looks like in a good visual representation of different data sets. It should be used for showing the relationship between partial and overall elements, the pie chart is used here to show the distribution of Murder percent from 1990 to 2000, this allows users to clearly show the percentage of Murder number in each year, allowing users to compare the annual data more easily. The advantage of using pie chart is that it can display relative proportions

compare the annual data more easily. The advantage of using pie chart is that it can display relative proportions of multiple classes of data, summarize a large data set in visual form and the size of the circle can be made proportional to the total quantity it represents.

1. **Area plot**

DataFrame.plot.area(x=None, y=None, **kwargs)

Parameters:

x:floats or array-like, x coordinates of plot

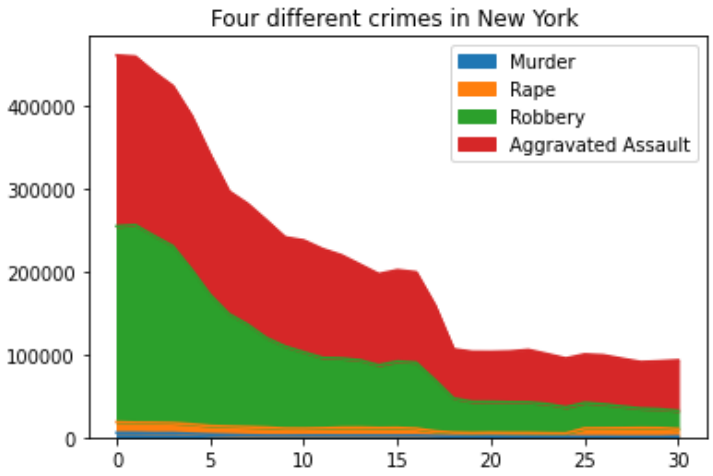
y:floats or array-like, y coordinates of plot

Read more on pandas https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.area.html

For this time, we need to compare three different crimes in New York.

In [16]:

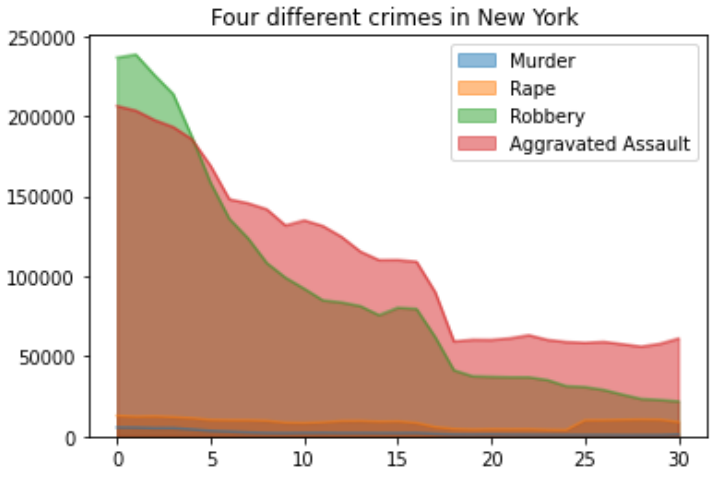
```
df_group1=df_group[["Murder","Rape","Robbery","Aggravated Assault"]]
df_group1.plot.area()
plt.title('Four different crimes in New York ')
plt.show()
```



If you would like to do an overlapping, you can specify (stacked=False)

In [17]:

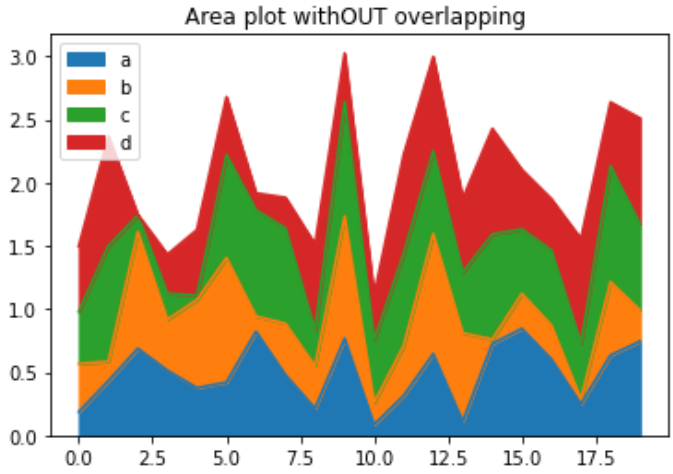
```
df_group1.plot.area(stacked=False);
plt.title('Four different crimes in New York ')
plt.show()
```



Even though they looks little similar, to ditinguish them, we create a new random datafram to watch the difference.

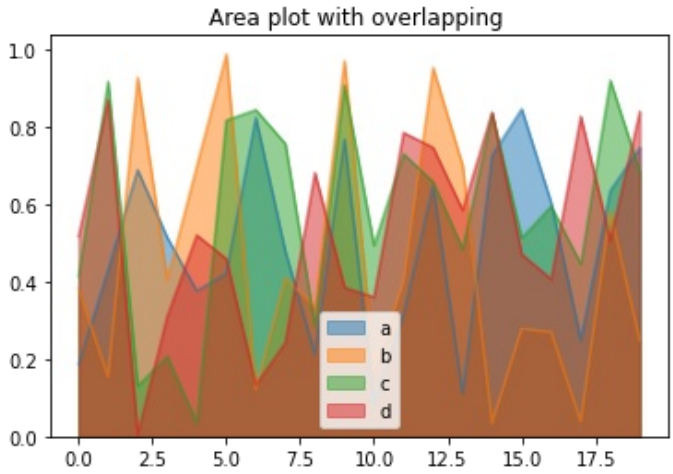
In [18]:

```
df_new= pd.DataFrame(np.random.rand(20, 4), columns=["a", "b", "c","d"])
df_new.plot.area()
plt.title('Area plot withOUT overlapping ')
plt.show()
```



In [19]:

```
df_new.plot.area(stacked=False)
plt.title('Area plot with overlapping ')
plt.show()
```



Area plot represents the distribution of the quantitative data, While the example above only plots a single line with shaded area, an area chart is typically used with multiple lines to make a comparison between groups (aka series) or to show how a whole is divided into component parts. There are two different types of area chart: the stacked area chart and the overlapping area chart. The stacked area chart is a part of the area graph where it demonstrates the behavior of multiple groups in a single chart. In the overlapping area chart, each line was shaded from its vertical value to a common baseline. In the stacked area chart, lines are plotted one at a time, with the height of the most recently plotted group serving as a moving baseline. Here, crimes in New York are classified into three categories: Robbery, Rape, Murder and Aggravated Assault, which are represented by green, yellow, blue, and red respectively. In the case that we want to compare the values between groups, we end up with an overlapping area chart. We can see most of the space occupied by the green and red areas, and they represent Robbery and Aggravated Assault respectively. This represents that most of the crimes that occurred in New York were classified as Robbery and Aggravated Assault. The blue area is the smallest area in the plot, which means that murder type crimes occur less frequently than other types of crimes.

1. **Hexagonal bin**

DataFrame.plot.hexbin(x, y, C=None, reduce_C_function=None, gridsize=None, **kwargs)

Parameters:

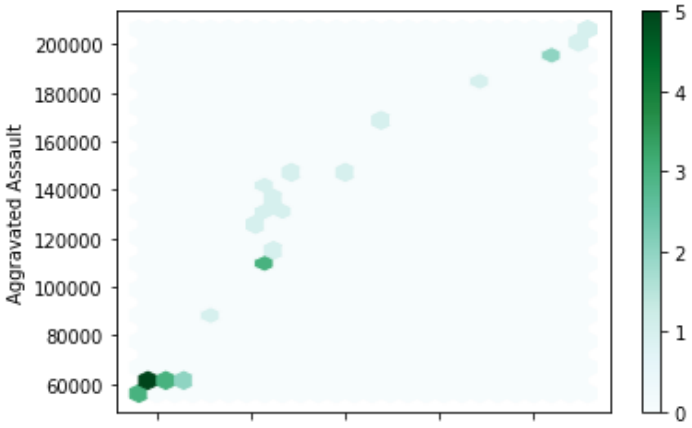
x:floats or array-like, x coordinates of plot

y:floats or array-like, y coordinates of plot

Read more on pandas https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.hexbin.html

In [20]:

```
df_group2=df_group[["Murder","Aggravated Assault"]]
df_group2.plot.hexbin(x="Murder", y="Aggravated Assault", gridsize=25);
plt.show()
```



1. **Parallel Coordinates**

pandas.plotting.parallel_coordinates(frame, class_column, cols=None, ax=None, color=None, use_columns=False, xticks=None, colormap=None, axvlines=True, axvlines_kwds=None, sort_labels=False, **kwargs)

Parameters:

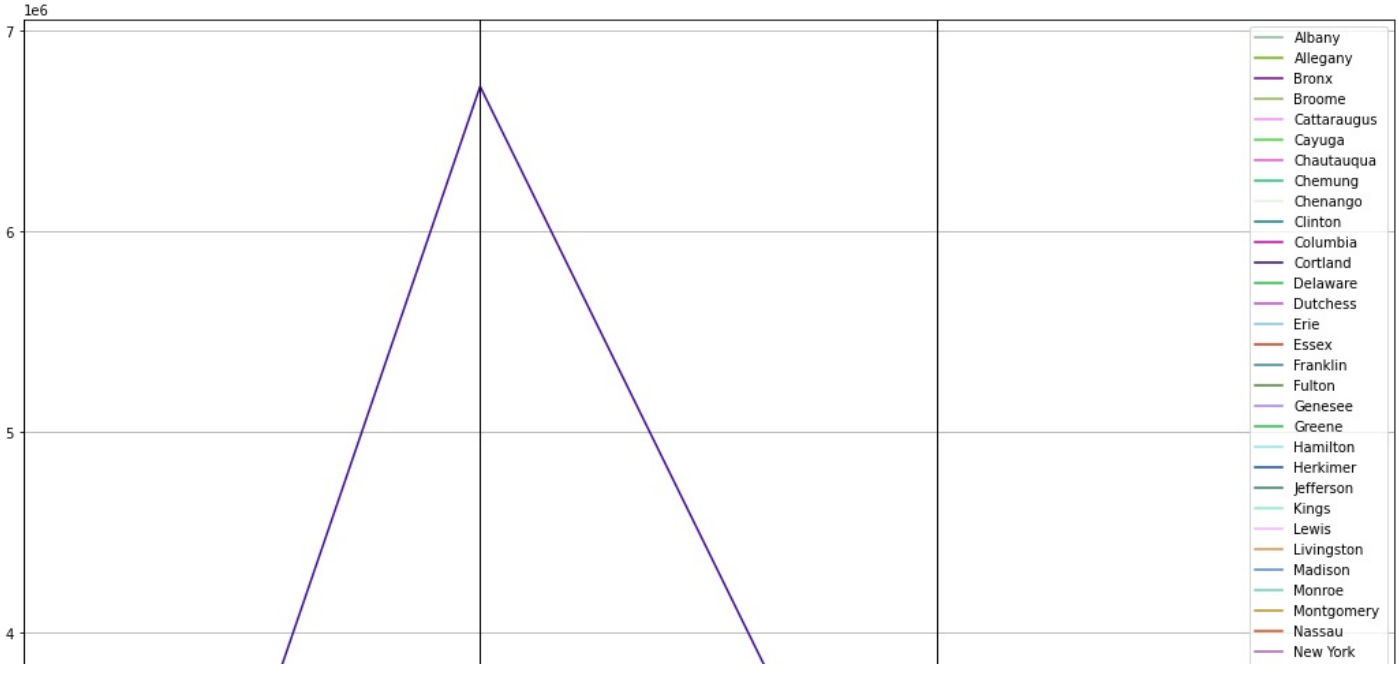
frame: Data Frame for plotting class_column: Column name, one line for each class

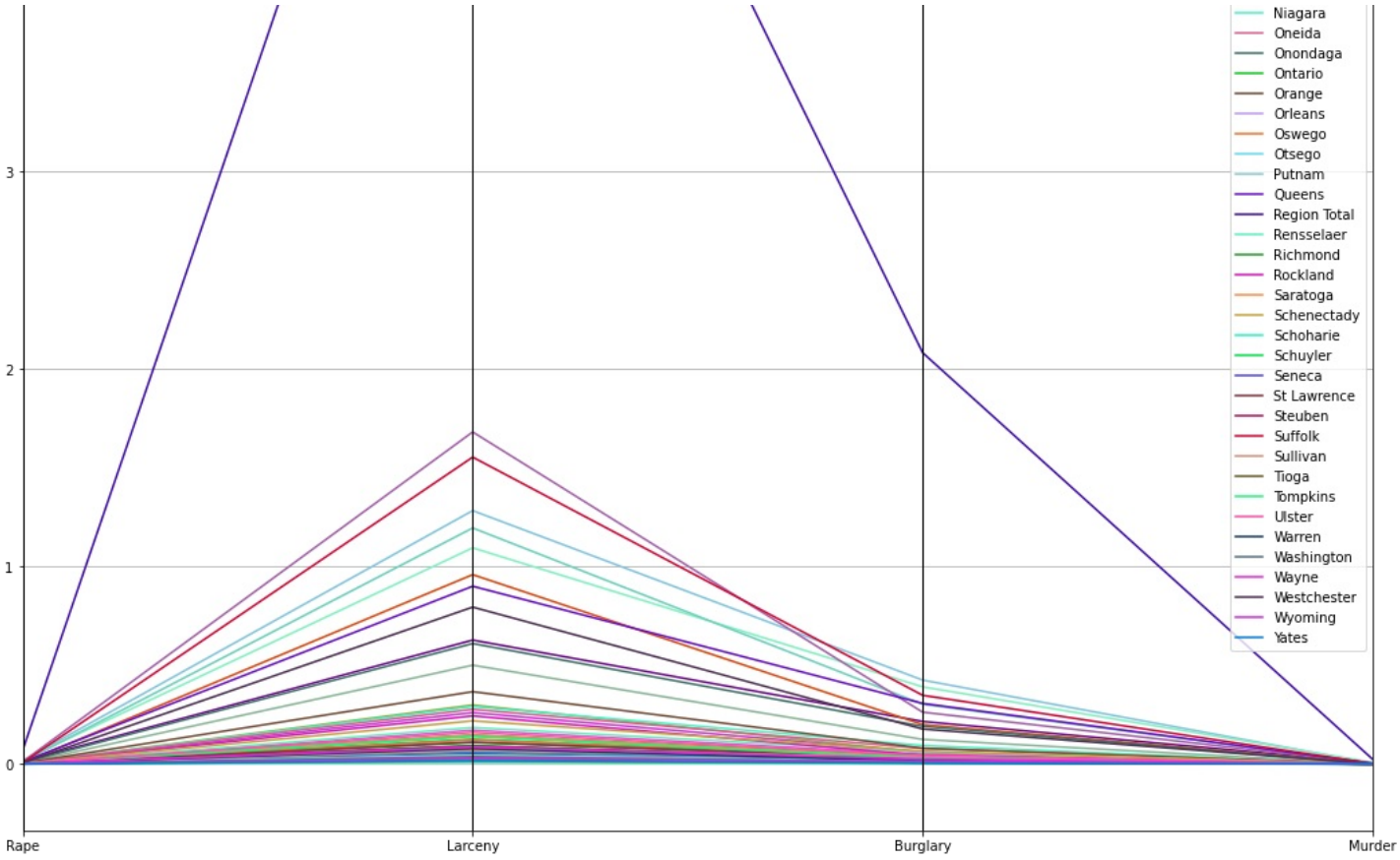
Read more on pandas https://pandas.pydata.org/docs/reference/api/pandas.plotting.parallel_coordinates.html

Parallel Coordinates plot is used to observe trend among the classes plotted. It is especially useful for observing relationship and outliers.

In [21]:

```
df2= df2.groupby(by=['County']).sum() # group by county sum the rest
df2.reset_index(level=0, inplace=True)
plt.figure(figsize=(18,20))
pd.plotting.parallel_coordinates(df2, 'County')
plt.show()
```





1. **Scatter Matrix**

pandas.plotting.scatter_matrix(frame, alpha=0.5, figsize=None, ax=None, grid=False, diagonal='hist', marker='.', density_kwds=None, hist_kwds=None, range_padding=0.05, **kwargs)

Parameters:

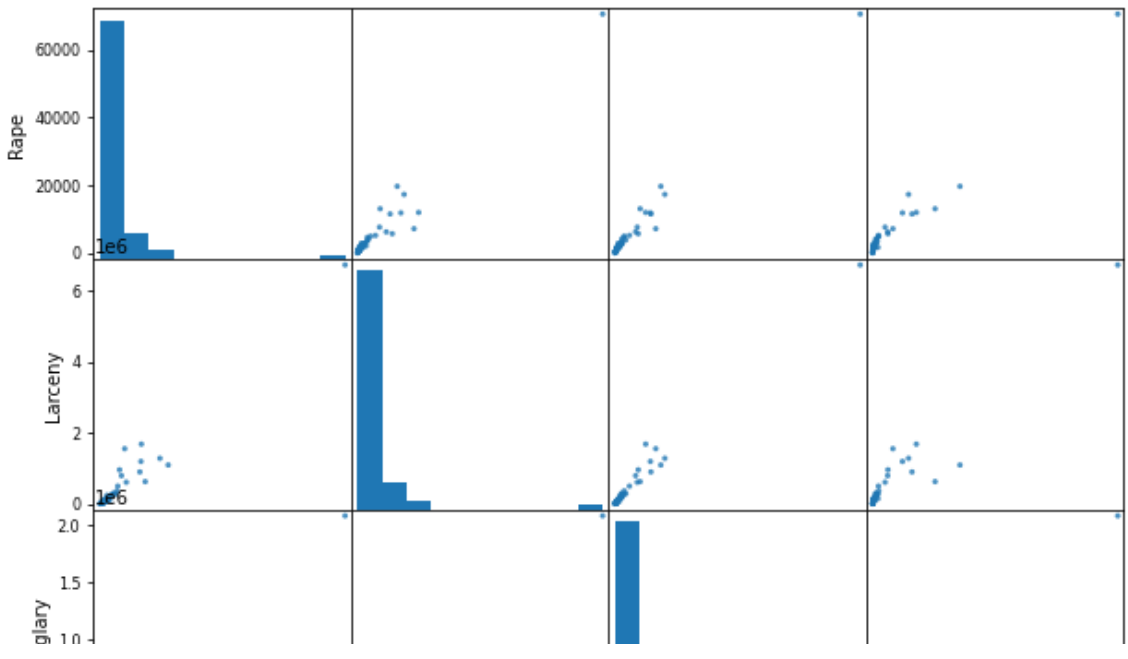
frame: data frame

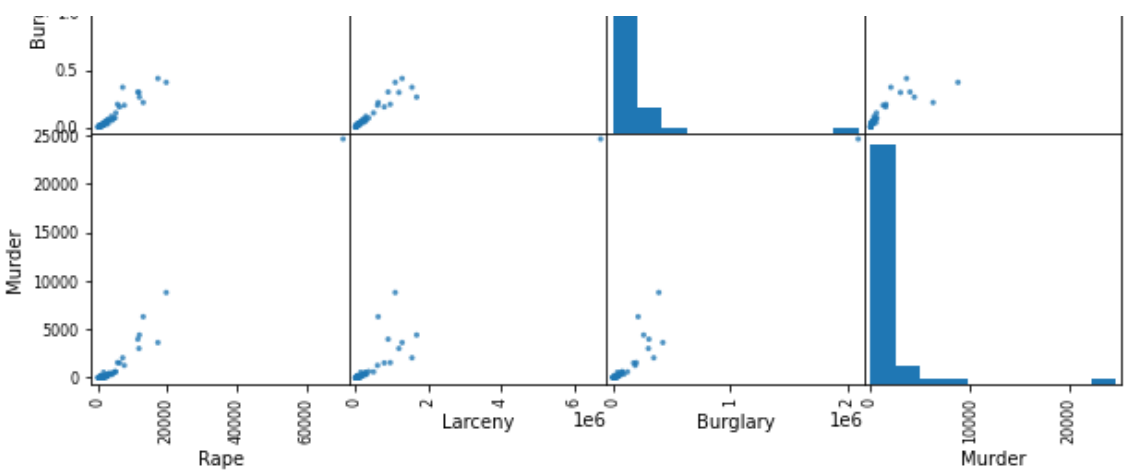
Read more on pandas https://pandas.pydata.org/docs/reference/api/pandas.plotting.scatter_matrix.html

Scatter Matrix is a plot that is good at comparing each pair of column features in the dataframe. This plots show a pairwise relationship in a matrix.

In [22]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df2,alpha=0.8,figsize=(10,10))
plt.show()
```





1. **Mosaic Plot**

statsmodels.graphics.mosaicplot.mosaic(data, index=None, ax=None, horizontal=True, gap=0.005, properties=<function >, labelizer=None, title='', statistic=False, axes_label=True, label_rotation=0.0)

Parameters:

frame: Data Frame

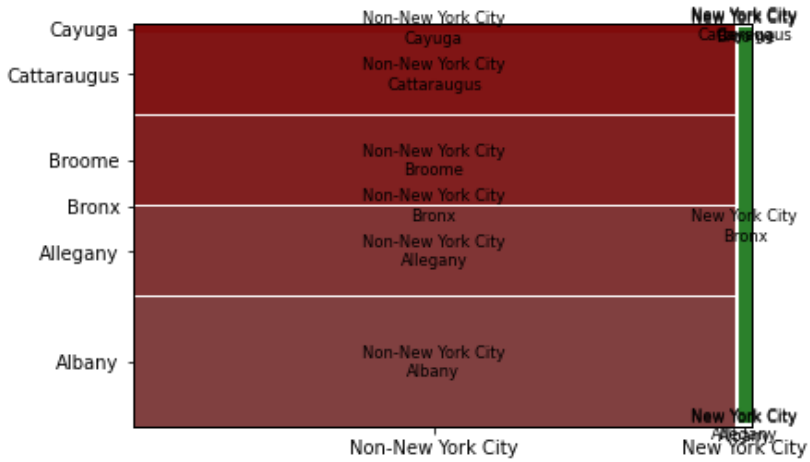
Columns: an array, columns for splitting

Read more on https://www.statsmodels.org/dev/generated/statsmodels.graphics.mosaicplot.mosaic.html

Mosaic plot is a tool to visualize whether there's a relationship among the target variables.

In [23]:

```
from statsmodels.graphics.mosaicplot import mosaic
dfm=df[df['Year']==2000] #select year 2000 data
dfmo = dfm.head(50) # choose first 50 rows
mosaic(dfmo,['Region','County'])
plt.show()
```



Multiple plots in one figure

Sometimes multiple plots side by side may provide a better visualization

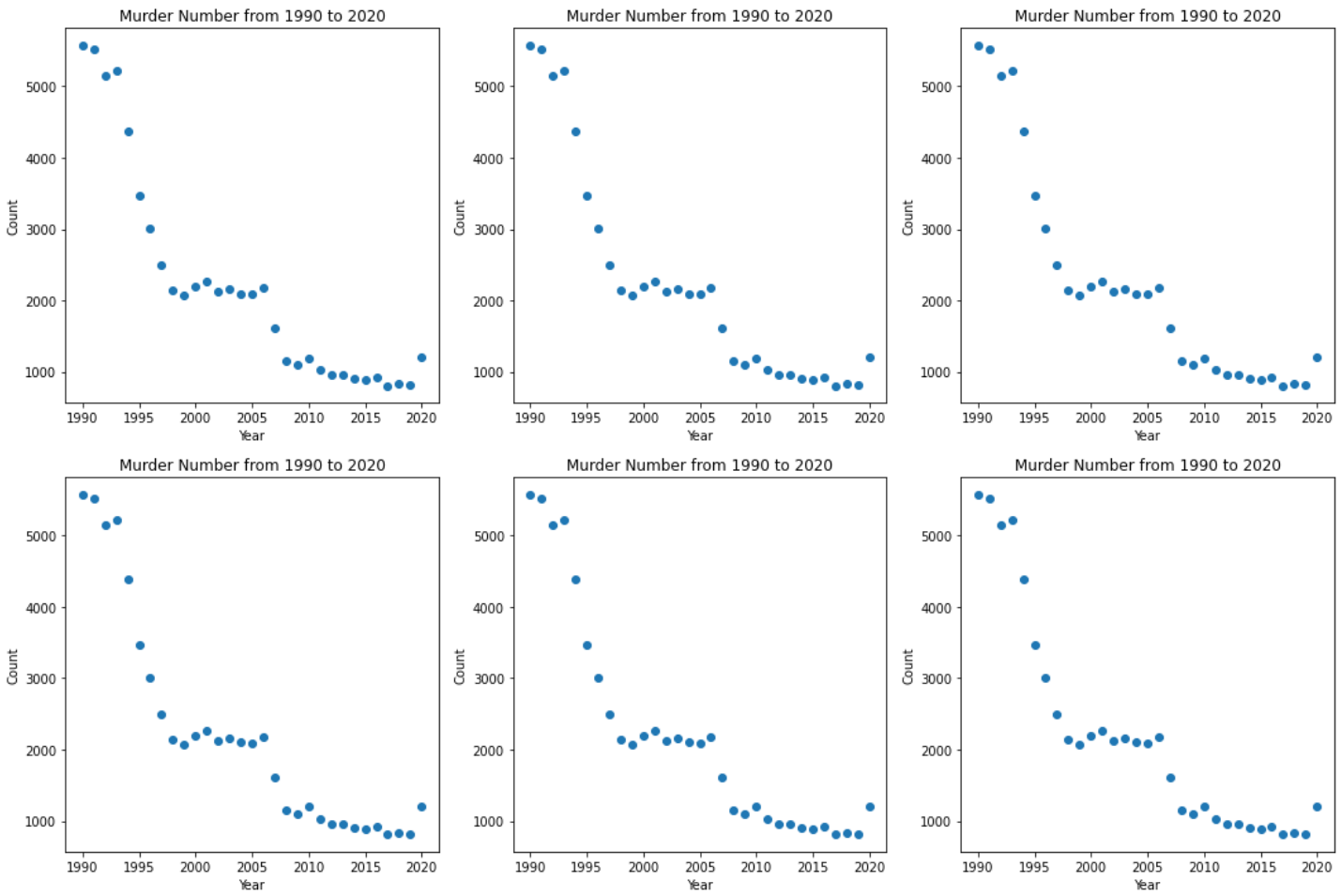
1. **Using Subplot**

Subplot can be used for plots arranged in a matrix

In [24]:

```
plt.figure(figsize=(18,12))
for i in range(6):
    plt.subplot(2,3,i+1)
```

```
    plt.scatter(df_group['Year'],df_group['Murder'])
    plt.title("Murder Number from 1990 to 2020")
    plt.xlabel('Year')
    plt.ylabel('Count')
```

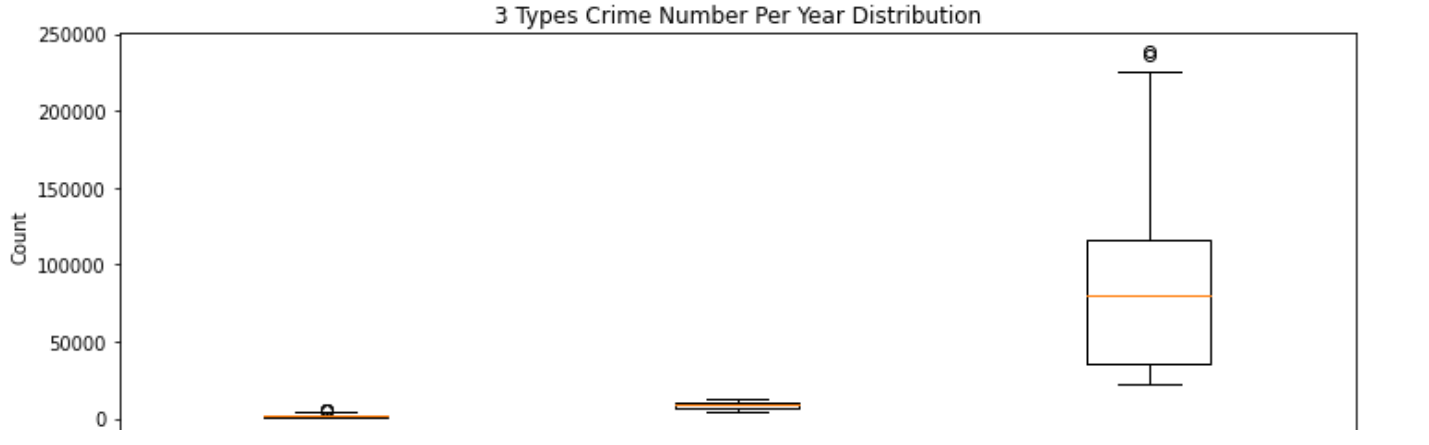


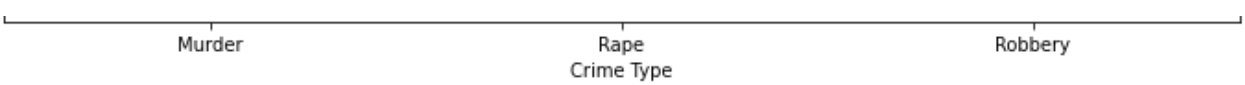
1. **Multiple plots storing in a dictionary**

Same kind of plot using a common x and y axis plot in one figure. This type of plot can be done by storing data in a dictionary. The plot below is a multiple boxplots in one figure, other types of plots can also be plotted in this way, for example violin plot.

In [25]:

```
box = {}
box['Murder'] = df_group['Murder']
box['Rape'] = df_group['Rape']
box['Robbery'] = df_group['Robbery']
fig, ax = plt.subplots(figsize=(12,4))
ax.boxplot(box.values())
ax.set_xticklabels(box.keys())
ax.set_xlabel('Crime Type')
ax.set_ylabel('Count')
ax.set_title('3 Types Crime Number Per Year Distribution')
plt.show()
```




Murder
Rape
Robbery
Crime Type