Factors that affect Crime in India and How

Introduction

This is the Data Science Project of USNs 1MS18IS134, 1MS18IS135, 1MS18IS099 only.

The objective of this notebook is to explore the various factors that have affected Crime in India, and if they have how so? Therefore, we worked on various datasets such as the number of Crimes reported in India, the various political parties in power, the literacy rate, the police strength and the population. All this data was divided statewise and for the years 2009-2011. The data was obtained from official government sources such as the National Crime Records Bureau (NCRB) and various other surverys conducted by the Govt. of India.

This notebook contains 5 parts:

1. Introduction
2. Data Preparation
3. Data Cleaning
4. Data Analysis
5. Data Prediciton
6. Data Visualisation
7. Conclusion

In the Introduction part, we simply start importing the libraries and the datasets we need

In [1]:
```
import pandas as pd
from statsmodels.graphics.gofplots import qqplot
import pylab as py
import statsmodels.api as sm
import numpy as np
import datetime as dt
from tqdm import tqdm
from scipy.stats import shapiro
import matplotlib.pyplot as plt
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import normaltest
from scipy.stats import anderson
from scipy import stats
from scipy.stats import chi2
from scipy.stats import chi2_contingency

df_crime = pd.read_csv("Age-group-wise and sex-wise break-up of persons arı
df = pd.read_csv("CM.csv")

print(df_crime.shape, df.shape)
```

```
(1317, 15) (520, 23)
```

In [2]:
```
df_crime.head()
```

Out[2]:

	SL	STATE/UT	Below 18 Years	Unnamed: 3	Between 18-30 Years	Unnamed: 5	Between 30-45 Years	Unnamed: 7	Betwee 45-6 Year
0	NaN	NaN	Male	Female	Male	Female	Male	Female	Mal
1	NaN	NaN	B18M	B18F	B30M	B30F	B45M	B45F	B60M
2	1	ANDHRA PRADESH	28	0	1667	136	1712	155	78
3	2	ARUNACHAL PRADESH	0	0	43	0	35	0	
4	3	ASSAM	13	1	721	14	856	17	19

In [3]:
```
df.head()
```

Out[3]:

	State_Name	Assembly_No	Sequence_No	pid_CM	No_Terms_as_CM	Name	Start_
0	Andaman and Nicobar Islands	NaN	NaN	NaN	NaN	Bishnu Pada Roy	01/0
1	Chandigarh	NaN	NaN	NaN	NaN	Pawan Kumar Bansal	22/0
2	Daman and Diu	NaN	NaN	NaN	NaN	Lalubhai Patel	31/0
3	Delhi	NaN	NaN	NaN	NaN	Manmohan Singh	22/0
4	Jammu and Kashmir	NaN	NaN	NaN	NaN	Omar Abdullah	05/0

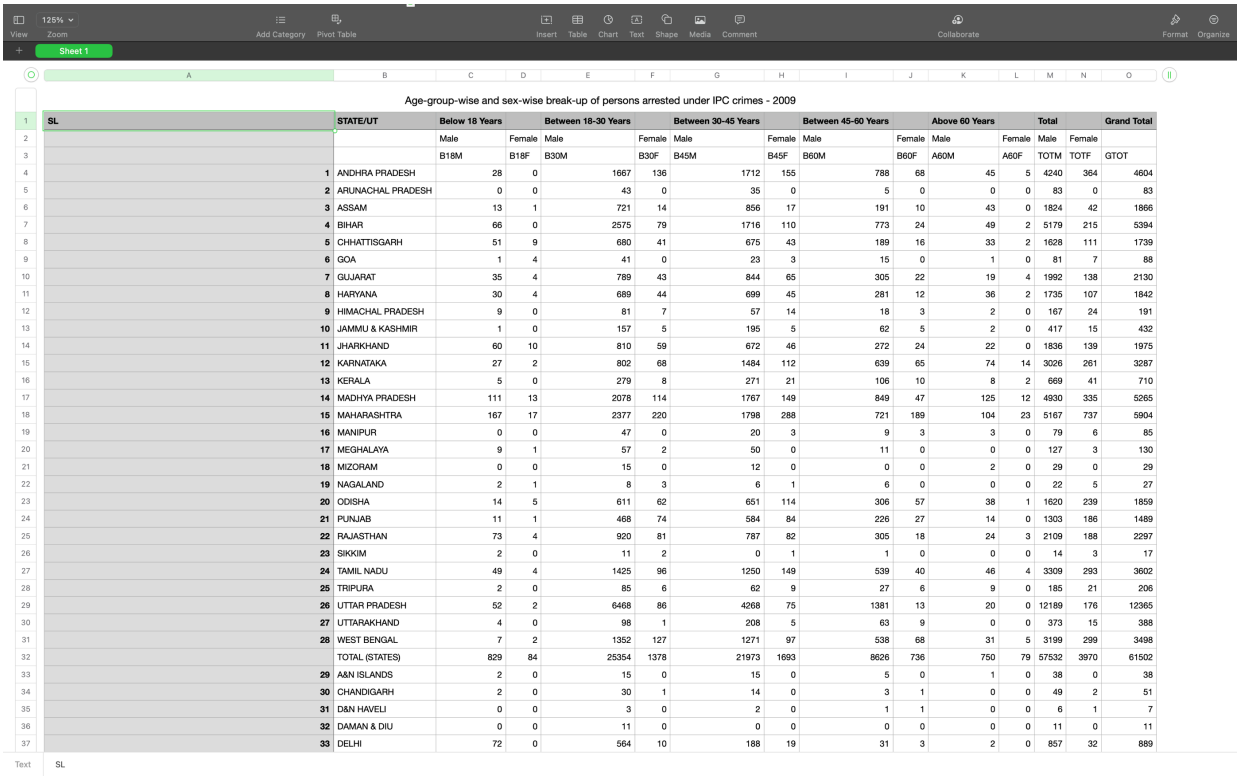
5 rows × 23 columns

As we can see, the data requires extensive preparation and cleaning before any analysis can be performed.

Data Preparation & Data Cleaning

Although, these contents were separated in the Introduction, it was observed that Data Preparation and Data Cleaning go hand-in-hand and is not mutually exclusive. This will be proved as we progress ahead.

In this part, we consolidated the Crime data as shown in the image below to Statewise Crime numbers according to the various categories of crime under IPC. Unfortunately, this part of the process could not be automated and had to be manually prepared.



Age-group-wise and sex-wise break-up of persons arrested under IPC crimes - 2009

SL	STATE/UT	Below 18 Years		Between 18-30 Years		Between 30-45 Years		Between 45-60 Years		Above 60 Years		Total		Grand Total
		Male	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male	Female	
		B18M	B18F	B30M	B30F	B45M	B45F	B60M	B60F	A60M	A60F	TOTM	TOTF	GTOT
1	ANDHRA PRADESH	28	0	1667	136	1712	155	788	68	45	5	4240	364	4604
2	ARUNACHAL PRADESH	0	0	43	0	35	0	5	0	0	0	83	0	83
3	ASSAM	13	1	721	14	856	17	191	10	43	0	1824	42	1866
4	BIHAR	66	0	2575	79	1716	110	773	24	49	2	5179	215	5394
5	CHHATTISGARH	51	9	680	41	675	43	189	16	33	2	1628	111	1739
6	GOA	1	4	41	0	23	3	15	0	1	0	81	7	88
7	GUJARAT	35	4	789	43	844	65	305	22	19	4	1992	138	2130
8	HARYANA	30	4	689	44	699	45	281	12	36	2	1735	107	1842
9	HIMACHAL PRADESH	9	0	81	7	57	14	18	3	2	0	167	24	191
10	JAMMU & KASHMIR	1	0	157	5	195	5	62	5	2	0	417	15	432
11	JHARKHAND	60	10	810	59	672	46	272	24	22	0	1836	139	1975
12	KARNATAKA	27	2	802	68	1484	112	639	65	74	14	3026	261	3287
13	KERALA	5	0	279	8	271	21	106	10	8	2	669	41	710
14	MADHYA PRADESH	111	13	2078	114	1767	149	849	47	125	12	4930	335	5265
15	MAHARASHTRA	167	17	2377	220	1798	288	721	189	104	23	5167	737	5904
16	MANIPUR	0	0	47	0	20	3	9	3	3	0	79	6	85
17	MEGHALAYA	9	1	57	2	50	0	11	0	0	0	127	3	130
18	MIZORAM	0	0	15	0	12	0	0	0	2	0	29	0	29
19	NAGALAND	2	1	8	3	6	1	6	0	0	0	22	5	27
20	ODISHA	14	5	611	62	651	114	306	57	38	1	1620	239	1859
21	PUNJAB	11	1	468	74	584	84	226	27	14	0	1303	186	1489
22	RAJASTHAN	73	4	920	81	787	82	305	18	24	3	2109	188	2297
23	SIKKIM	2	0	11	2	0	1	1	0	0	0	14	3	17
24	TAMIL NADU	49	4	1425	96	1250	149	539	40	46	4	3309	293	3602
25	TRIPURA	2	0	85	6	62	9	27	6	9	0	185	21	206
26	UTTAR PRADESH	52	2	6468	86	4268	75	1381	13	20	0	12189	176	12365
27	UTTARAKHAND	4	0	98	1	208	5	63	9	0	0	373	15	388
28	WEST BENGAL	7	2	1352	127	1271	97	538	68	31	5	3199	299	3498
	TOTAL (STATES)	829	84	25354	1378	21973	1693	8626	736	750	79	57532	3970	61502
29	A&N ISLANDS	2	0	15	0	15	0	5	0	1	0	38	0	38
30	CHANDIGARH	2	0	30	1	14	0	3	1	0	0	49	2	51
31	D&N HAVELI	0	0	3	0	2	0	1	1	0	0	6	1	7
32	DAMAN & DIU	0	0	11	0	0	0	0	0	0	0	11	0	11
33	DELHI	72	0	564	10	188	19	31	3	2	0	857	32	889

As shown, the data is spread sex-wise and age-group-wise for various categories of crime for each state. We required the total number for each state and for each category. After extensive cleaning up and consoldating of the data manually, we were able to make it statewise.

In [4]:
```
df_2009 = pd.read_csv("CrimeRate_2009_2.csv")
df_2009.head()
```

Out[4]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C
0	1	ANDHRA PRADESH	4604.0	3065.0	240.0	1487.0	0.0	1487.0	2521.0	1889.0	...	123(
1	2	ARUNACHAL PRADESH	83.0	48.0	4.0	60.0	0.0	60.0	61.0	38.0	...	25
2	3	ASSAM	1866.0	559.0	56.0	1644.0	0.0	1644.0	2875.0	2392.0	...	87
3	4	BIHAR	5394.0	6169.0	385.0	1086.0	0.0	1086.0	4727.0	2397.0	...	112(
4	5	CHHATTISGARH	1739.0	1350.0	31.0	1128.0	0.0	1128.0	373.0	267.0	...	32

5 rows × 32 columns

This was prepared for the other two years as well.

In [5]:

```
df_2010 = pd.read_csv("CrimeRate_2010.csv")
df_2010.head()
```

Out[5]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C21
0	1	A&N ISLANDS	7.0	39.0	5.0	39.0	0.0	39.0	18.0	11.0	...	10.0
1	2	ANDHRA PRADESH	4239.0	3173.0	266.0	1761.0	0.0	1761.0	2543.0	1722.0	...	972.0
2	3	ARUNACHAL PRADESH	103.0	58.0	3.0	49.0	0.0	49.0	81.0	48.0	...	30.0
3	4	ASSAM	1537.0	471.0	43.0	1629.0	5.0	1624.0	3190.0	2687.0	...	826.0
4	5	BIHAR	5207.0	5418.0	547.0	892.0	0.0	892.0	4518.0	2503.0	...	940.0

5 rows × 32 columns

In [6]:

```
df_2011 = pd.read_csv("CrimeRate_2011.csv")
df_2011.head()
```

Out[6]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C21
0	1	A&N ISLANDS	18.0	19.0	4.0	28.0	0.0	28.0	16.0	14.0	...	4.0
1	2	ANDHRA PRADESH	5584.0	4239.0	286.0	1758.0	0.0	1758.0	2461.0	1698.0	...	1186.0
2	3	ARUNACHAL PRADESH	62.0	27.0	3.0	47.0	0.0	47.0	100.0	67.0	...	31.0
3	4	ASSAM	1666.0	554.0	48.0	1470.0	0.0	1470.0	3279.0	2838.0	...	831.0
4	5	BIHAR	8898.0	8192.0	562.0	1185.0	1.0	1184.0	5721.0	3565.0	...	1206.0

5 rows × 32 columns

As we can see, some of the data is not in the same order and we require the total count of the crimes commited, hence grouping was done and the felony/misdemeanor ratio was found. Felony - a major crime - Punished by a fine, by imprisonment, or both. This includes :-

1. Murder
2. Robbery
3. Rape
4. Kidnapping

Midemeanor - a less serious crime - Punished by a fine, jail time, or both. This includes :-

1. Driving without license
2. Theft
3. Simple Assault
4. Disorderly Conduct
5. Trespass

We consolidated the entire dataset over the three years and later split it year-wise again.

In [7]:
```
df_2009.dropna(inplace=True)
df_2010.dropna(inplace=True)
df_2011.dropna(inplace=True)
df_2009.sort_values(by=['STATE/UT'], inplace=True, ignore_index=True)
df_2009.loc[0, ["STATE/UT"]] = df_2010.iloc[0]["STATE/UT"]
df_all = pd.concat([df_2009, df_2010, df_2011])
df_all.dropna(inplace=True, axis=1)
df_all.drop(['C30'], axis=1, inplace=True)
df_all['Felony'] = df_all.iloc[:, 2:14].sum(axis=1)
df_all['TotalCrimes'] = df_all.iloc[:, 14:].sum(axis=1)
df_all['Misdemeanor'] = df_all['TotalCrimes']-df_all['Felony']
df_all['FelonyRatio'] = df_all['Felony']/df_all['TotalCrimes']
df_all['MisdemeanorRatio'] = df_all['Misdemeanor']/df_all['TotalCrimes']
df_2009, df_2010, df_2011 = np.array_split(df_all, 3)
```

In [8]:
```
df_2009.head()
```

Out[8]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C25
0	29	A&N ISLANDS	38.0	5.0	7.0	36.0	0.0	36.0	16.0	5.0	...	8.0
1	1	ANDHRA PRADESH	4604.0	3065.0	240.0	1487.0	0.0	1487.0	2521.0	1889.0	...	4178.0
2	2	ARUNACHAL PRADESH	83.0	48.0	4.0	60.0	0.0	60.0	61.0	38.0	...	2.0
3	3	ASSAM	1866.0	559.0	56.0	1644.0	0.0	1644.0	2875.0	2392.0	...	15.0
4	4	BIHAR	5394.0	6169.0	385.0	1086.0	0.0	1086.0	4727.0	2397.0	...	16.0

5 rows × 36 columns

In [9]:
```
df_2010.head()
```

Out[9]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C25
0	1	A&N ISLANDS	7.0	39.0	5.0	39.0	0.0	39.0	18.0	11.0	...	14.0
1	2	ANDHRA PRADESH	4239.0	3173.0	266.0	1761.0	0.0	1761.0	2543.0	1722.0	...	3820.0
2	3	ARUNACHAL PRADESH	103.0	58.0	3.0	49.0	0.0	49.0	81.0	48.0	...	1.0
3	4	ASSAM	1537.0	471.0	43.0	1629.0	5.0	1624.0	3190.0	2687.0	...	35.0
4	5	BIHAR	5207.0	5418.0	547.0	892.0	0.0	892.0	4518.0	2503.0	...	22.0

5 rows × 36 columns

In [10]:
```
df_2011.head()
```

Out[10]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	C25
0	1	A&N ISLANDS	18.0	19.0	4.0	28.0	0.0	28.0	16.0	14.0	...	7.0
1	2	ANDHRA PRADESH	5584.0	4239.0	286.0	1758.0	0.0	1758.0	2461.0	1698.0	...	4122.0
2	3	ARUNACHAL PRADESH	62.0	27.0	3.0	47.0	0.0	47.0	100.0	67.0	...	0.0
3	4	ASSAM	1666.0	554.0	48.0	1470.0	0.0	1470.0	3279.0	2838.0	...	26.0
4	5	BIHAR	8898.0	8192.0	562.0	1185.0	1.0	1184.0	5721.0	3565.0	...	9.0

5 rows × 36 columns

Now, coming to the Chief Ministers of India dataset (to represent the political party in power), we were to acquire a consolidated dataset of all CMs since 1950. But we required the year-wise split of the political party in power, but the dataset we had had the start-date and the end-date. Using code we were able to solve this problem to split it year-wise.

In [11]:
```
df.head()
```

Out[11]:

	State_Name	Assembly_No	Sequence_No	pid_CM	No_Terms_as_CM	Name	Start_
0	Andaman and Nicobar Islands	NaN	NaN	NaN	NaN	Bishnu Pada Roy	01/(
1	Chandigarh	NaN	NaN	NaN	NaN	Pawan Kumar Bansal	22/0
2	Daman and Diu	NaN	NaN	NaN	NaN	Lalubhai Patel	31/0
3	Delhi	NaN	NaN	NaN	NaN	Manmohan Singh	22/0
4	Jammu and Kashmir	NaN	NaN	NaN	NaN	Omar Abdullah	05/(

5 rows × 23 columns

In [12]:

```
df['Start_Date'] = pd.to_datetime(df['Start_Date'])
df['End_Date'] = pd.to_datetime(df['End_Date'], errors="coerce")
df[(df['Start_Date'] < dt.datetime(2012, 1, 1)) & (df['End_Date'] > dt.date
df1 = pd.DataFrame(columns=['Year'])
df = df.join(df1, how="outer")
def year_wise(df):
    updated_df = pd.DataFrame(columns=["Year", "State_Name", "Name", "Start
    for i in tqdm (range(0, len(df))):
        start_date = df.iloc[i]["Start_Date"]
        start_year = df.iloc[i]["Start_Date"].year
        end_year = df.iloc[i]["End_Date"].year
        end_date = df.iloc[i]["End_Date"]
        state_name = df.iloc[i]["State_Name"]
        name = df.iloc[i]["Name"]
        party = df.iloc[i]["Party"]
        while(start_year<=end_year):
            temp_df = {"Year": start_year, "State_Name": state_name, "Name"
            updated_df = updated_df.append(temp_df, ignore_index=True)
            start_year += 1
    return updated_df

new_df = year_wise(df)
new_df.head()
```

```
100%|██████████| 520/520 [00:08<00:00, 64.05it/s]
```

Out[12]:

	Year	State_Name	Name	Start_Date	End_Date	Party
0	2005	Andaman and Nicobar Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	2006	Andaman and Nicobar Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
2	2007	Andaman and Nicobar Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
3	2008	Andaman and Nicobar Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
4	2009	Andaman and Nicobar Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP

With this, we were able to split the dataset year-wise. Next using excel, we filtered the data for the years we required and created separate datasets for the year 2009, 2010 and 2011.

In [13]:

```
df_cm_2009 = pd.read_csv("Updated_CM_2009.csv")
df_cm_2010 = pd.read_csv("Updated_CM_2010.csv")
df_cm_2011 = pd.read_csv("Updated_CM_2011.csv")
```

In [14]:
```
df_cm_2009.head()
```

Out[14]:

	Year	State_Name	Name	Start_Date	End_Date	Party
0	2009	A&N Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	2009	Chandigarh	Pawan Kumar Bansal	2009-05-22	2014-05-18	INC
2	2009	Delhi	Manmohan Singh	2004-05-22	2014-05-25	INC
3	2009	Jammu and Kashmir	Omar Abdullah	2009-05-01	2015-08-01	JKNC
4	2009	Lakshadweep	Muhammed Hamdulla Sayeed	2009-05-18	2014-05-18	INC

In [15]:
```
df_cm_2010.head()
```

Out[15]:

	Unnamed: 0	Year	State_Name	Name	Start_Date	End_Date	Party
0	5	2010	A&N Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	11	2010	Chandigarh	Pawan Kumar Bansal	2009-05-22	2014-05-18	INC
2	22	2010	Delhi	Manmohan Singh	2004-05-22	2014-05-25	INC
3	28	2010	Jammu & Kashmir	Omar Abdullah	2009-05-01	2015-08-01	JKNC
4	35	2010	Lakshadweep	Muhammed Hamdulla Sayeed	2009-05-18	2014-05-18	INC

In [16]:
```
df_cm_2011.head()
```

Out[16]:

	Unnamed: 0	Year	State_Name	Name	Start_Date	End_Date	Party
0	6	2011	A&N Islands	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	12	2011	Chandigarh	Pawan Kumar Bansal	2009-05-22	2014-05-18	INC
2	23	2011	Delhi	Manmohan Singh	2004-05-22	2014-05-25	INC
3	29	2011	Jammu & Kashmir	Omar Abdullah	2009-05-01	2015-08-01	JKNC
4	36	2011	Lakshadweep	Muhammed Hamdulla Sayeed	2009-05-18	2014-05-18	INC

Other datasets, such as the population strength and literacy rate were manually entered and compiled into a dataset.

In [17]:

```
cdf = pd.read_csv("Compiled Datasets.csv", thousands=',')
cdf["POPULATION 2011"] = pd.to_numeric(cdf["POPULATION 2011"])
cdf = cdf.rename({"State" : "STATE/UT"}, axis = 1)
cdf["STATE/UT"] = cdf["STATE/UT"].str.upper()
cdf.head()
```

Out[17]:

	STATE/UT	POPULATION 2011	Unnamed: 2	2009 % of LR	LR 2009	2010 % of LR	LR 2010	2011 % of LR	L
0	A&N ISLANDS	380581.0	NaN	82.3	314033.75	83.4	318231.04	0.86	329
1	ANDHRA PRADESH	49577103.0	NaN	60.5	5126467.50	76.1	6448333.50	0.15	12
2	ARUNACHAL PRADESH	1383727.0	NaN	54.3	67386.30	78.3	97170.30	0.14	
3	ASSAM	31205576.0	NaN	63.3	1934954.40	85.3	2607450.40	0.11	3
4	BIHAR	104099452.0	NaN	47.0	4592840.00	71.9	7026068.00	0.20	19

Data cleaning is one of the most importants part of data science. As with most datasets, these dataset required data cleaning. Much of the cleaning for the Crime dataset was performed during the data preparation, such as the dropping of columns and rows with NAN values etc. Now, coming to the CM dataset, we need the states to have the same name as the Crime datasets to enable joining/merging them. And the column name should also be the same.

Since we transformed a range set to a discrete set, there could be some duplicates in the CM dataset (if there was a change in the CM that year). Hence, we dropped the duplicates and kept the last entry for simplicity.

In [18]:
```
df_cm_2009.drop_duplicates(subset ="State_Name", keep = "last", inplace = T
df_cm_2009.sort_values(by=["State_Name"], inplace = True, ignore_index = Tr
df_cm_2009["State_Name"] = df_cm_2009["State_Name"].str.upper()
df_cm_2009 = df_cm_2009.rename({"State_Name" : "STATE/UT"}, axis = 1)

df_cm_2010.drop_duplicates(subset ="State_Name", keep = "last", inplace = T
df_cm_2010.sort_values(by=["State_Name"], inplace = True, ignore_index = Tr
df_cm_2010["State_Name"] = df_cm_2010["State_Name"].str.upper()
df_cm_2010 = df_cm_2010.rename({"State_Name" : "STATE/UT"}, axis = 1)

df_cm_2011.drop_duplicates(subset ="State_Name", keep = "last", inplace = T
df_cm_2011.sort_values(by=["State_Name"], inplace = True, ignore_index = Tr
df_cm_2011["State_Name"] = df_cm_2011["State_Name"].str.upper()
df_cm_2011 = df_cm_2011.rename({"State_Name" : "STATE/UT"}, axis = 1)
```

In [19]:
```
df_cm_2009.head()
```

Out[19]:

	Year	STATE/UT	Name	Start_Date	End_Date	Party
0	2009	A&N ISLANDS	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	2009	ANDHRA PRADESH	Sri. K. Rosaiah	2009-03-09	2010-11-24	INC
2	2009	ARUNACHAL PRADESH	Dorjee Khandu	2009-10-23	2011-04-05	INC
3	2009	ASSAM	Shri Tarun Gogoi	2006-12-05	2011-05-16	INC
4	2009	BIHAR	Nitish Kumar	2005-11-24	2014-05-20	JD(U)

In [20]:
```
df_cm_2010.head()
```

Out[20]:

	Unnamed: 0	Year	STATE/UT	Name	Start_Date	End_Date	Party
0	5	2010	A&N ISLANDS	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	93	2010	ANDHRA PRADESH	Sri. N.Kiran Kumar Reddy	2010-11-25	2014-02-28	INC
2	153	2010	ARUNACHAL PRADESH	Dorjee Khandu	2009-10-23	2011-04-05	INC
3	233	2010	ASSAM	Shri Tarun Gogoi	2006-12-05	2011-05-16	INC
4	252	2010	BIHAR	Nitish Kumar	2005-11-24	2014-05-20	JD(U)

In [21]:
```
df_cm_2011.head()
```

Out[21]:

	Unnamed: 0	Year	STATE/UT	Name	Start_Date	End_Date	Party
0	6	2011	A&N ISLANDS	Bishnu Pada Roy	2005-01-07	2014-10-04	BJP
1	94	2011	ANDHRA PRADESH	Sri. N.Kiran Kumar Reddy	2010-11-25	2014-02-28	INC
2	156	2011	ARUNACHAL PRADESH	Nabam Tuki	2011-01-11	2014-05-17	INC
3	235	2011	ASSAM	Shri Tarun Gogoi	2011-05-17	2016-05-21	INC
4	253	2011	BIHAR	Nitish Kumar	2005-11-24	2014-05-20	JD(U)

After the data has been cleaned and prepared they are merged with the crime dataset and the compiled dataset with all the police strength and the population and the literacy rates, please note that for population, we used the census of 2011, no other data was available from official sources.

In [22]:
```
result_2009 = pd.merge(df_2009, df_cm_2009, how='inner', on='STATE/UT')
result_2009 = pd.merge(result_2009, cdf, how='inner', on='STATE/UT')
result_2009["STATE/UT Party"] = result_2009["STATE/UT"] + "_" + result_2009
result_2009.sort_values(by=["FelonyRatio"], inplace=True)
result_2009.head()
```

Out[22]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	2011 % of LR
14	13	KERALA	710.0	1176.0	157.0	694.0	0.0	694.0	436.0	202.0	...	0.93
15	34	LAKSHADWEEP	0.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.92
22	35	PUDUCHERRY	167.0	87.0	28.0	1.0	0.0	1.0	24.0	11.0	...	0.80
10	9	HIMACHAL PRADESH	191.0	146.0	26.0	250.0	0.0	250.0	148.0	102.0	...	0.83
1	1	ANDHRA PRADESH	4604.0	3065.0	240.0	1487.0	0.0	1487.0	2521.0	1889.0	...	0.15

5 rows × 57 columns

In [23]:

```
result_2010 = pd.merge(df_2010, df_cm_2010, how='inner', on='STATE/UT')
result_2010 = pd.merge(result_2010, cdf, how='inner', on='STATE/UT')
result_2010["STATE/UT Party"] = result_2010["STATE/UT"] + "_" + result_2010
result_2010.sort_values(by=["FelonyRatio"], inplace=True)
result_2010.head()
```

Out[23]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	2011 % of LR
15	19	LAKSHADWEEP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.92
14	18	KERALA	680.0	941.0	123.0	659.0	0.0	659.0	340.0	221.0	...	0.93
10	14	HIMACHAL PRADESH	193.0	198.0	17.0	197.0	0.0	197.0	161.0	101.0	...	0.83
22	27	PUDUCHERRY	163.0	57.0	24.0	5.0	0.0	5.0	31.0	22.0	...	0.80
1	2	ANDHRA PRADESH	4239.0	3173.0	266.0	1761.0	0.0	1761.0	2543.0	1722.0	...	0.15

5 rows × 58 columns

In [24]:

```
result_2011 = pd.merge(df_2011, df_cm_2011, how='inner', on='STATE/UT')
result_2011 = pd.merge(result_2011, cdf, how='inner', on='STATE/UT')
result_2011["STATE/UT Party"] = result_2011["STATE/UT"] + "_" + result_2011
result_2011.sort_values(by=["FelonyRatio"], inplace=True)
result_2011.head()
```

Out[24]:

	SL	STATE/UT	C1	C2	C3	C4	C5	C6	C7	C8	...	2011 % of LR
14	18	KERALA	733.0	1337.0	134.0	1226.0	0.0	1226.0	349.0	230.0	...	0.93
15	19	LAKSHADWEEP	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.92
10	14	HIMACHAL PRADESH	186.0	115.0	14.0	187.0	0.0	187.0	145.0	134.0	...	0.83
8	12	GUJARAT	2408.0	1788.0	76.0	621.0	0.0	621.0	2235.0	1888.0	...	0.16
1	2	ANDHRA PRADESH	5584.0	4239.0	286.0	1758.0	0.0	1758.0	2461.0	1698.0	...	0.15

5 rows × 58 columns

Data Analysis

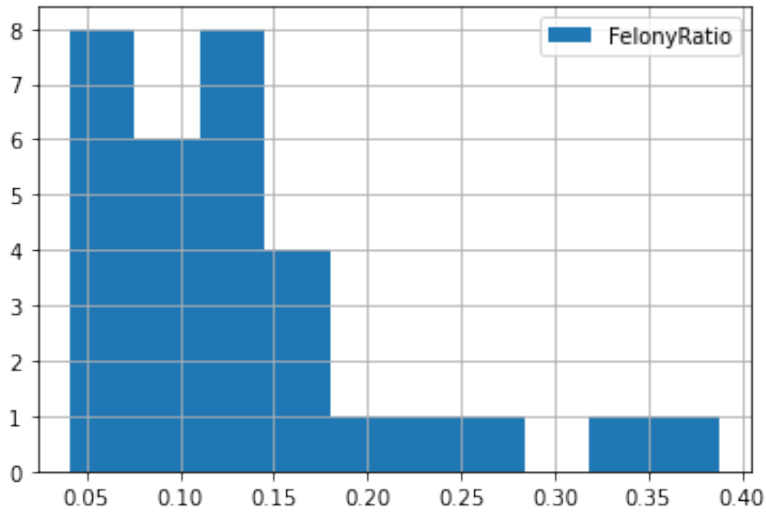
Hypothesis Testing

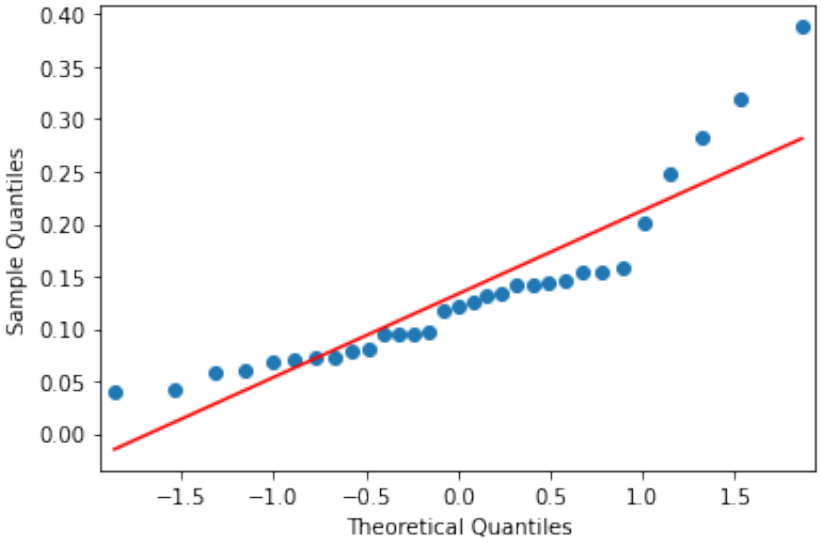
In this part, we plotted all questions for visual pieces of information. Our first course of action is to check if the available data is normal or not.

Let us plot the histogram plots and the qq plots first and then go ahead with various other tests.

In [25]:

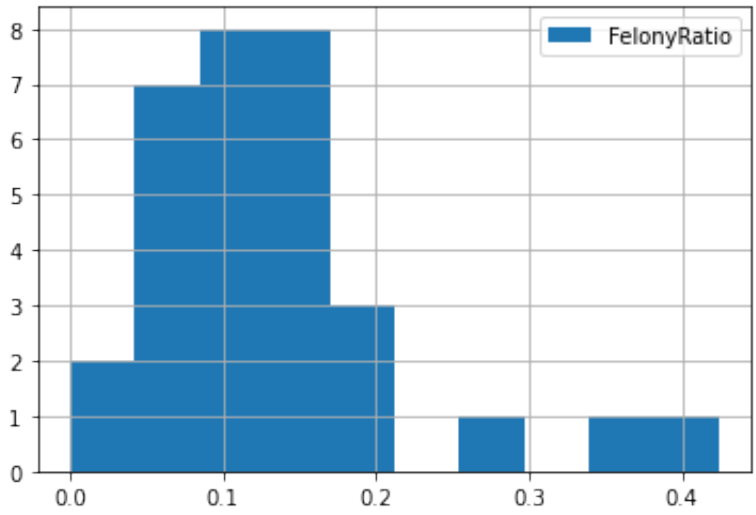
```
result_2009["FelonyRatio"].hist(legend=True)
sm.qqplot(result_2009.loc[:, "FelonyRatio"], line ='s')
py.show()
```

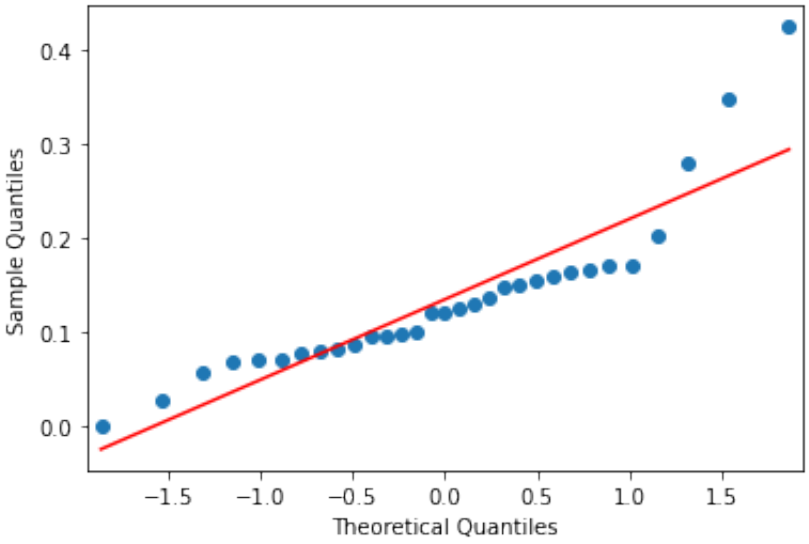




In [26]:

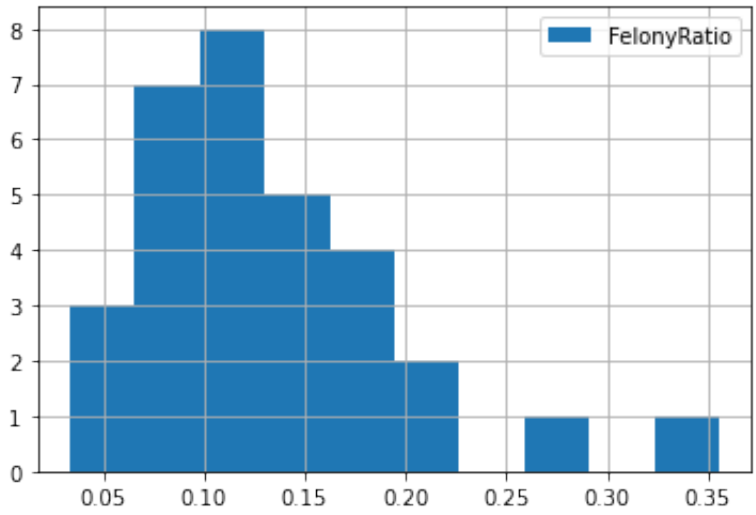
```
result_2010["FelonyRatio"].hist(legend=True)
sm.qqplot(result_2010.loc[:, "FelonyRatio"], line ='s')
py.show()
```

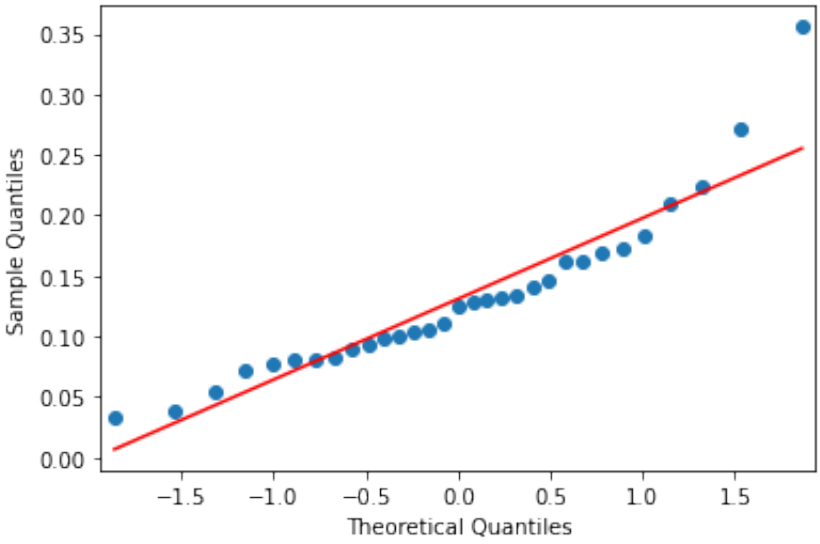




In [27]:

```
result_2011["FelonyRatio"].hist(legend=True)
sm.qqplot(result_2011.loc[:, "FelonyRatio"], line ='s')
py.show()
```



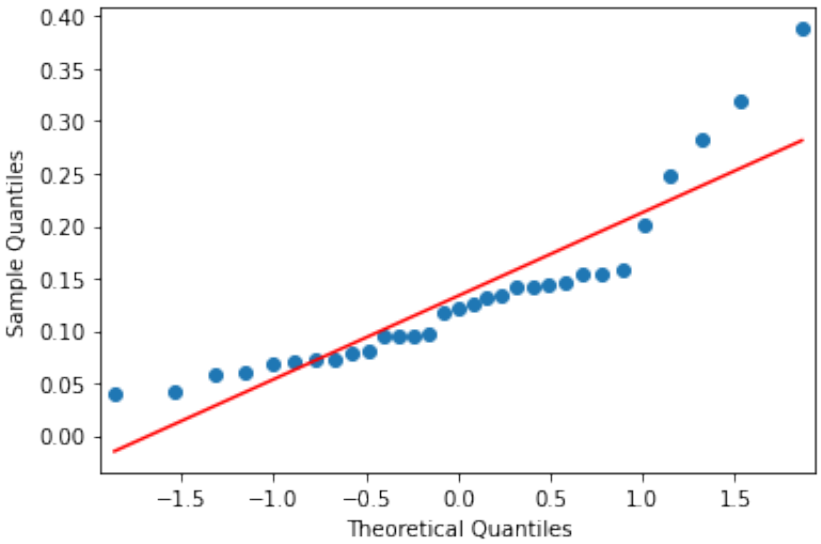


As we can see, the data doesn't look normal in these graphs, let us conduct some normality tests to verify.

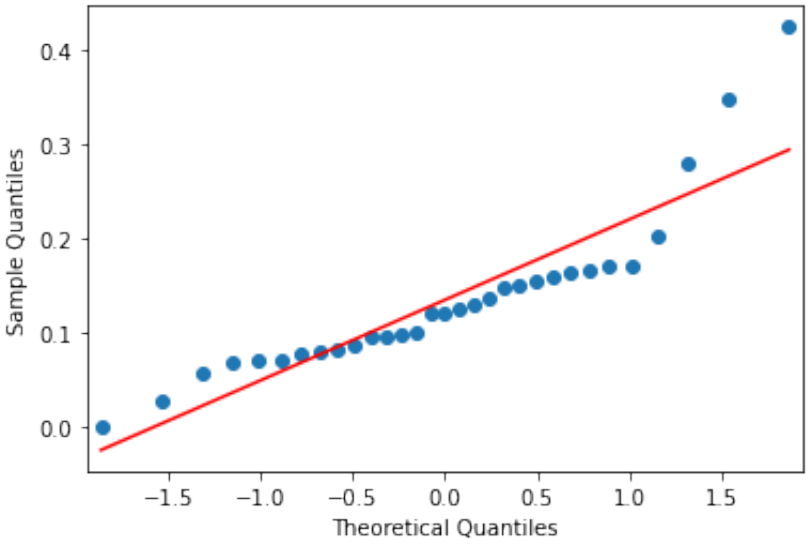
In [28]:

```
df_list = [result_2009, result_2010, result_2011]
for i in range(3):
#     try:
    print("Test for Year: ", (i+2009))
    print('\nQ-Q Plot')
    qqplot(df_list[i]['FelonyRatio'], line='s')
    plt.show()
    print('\nShapiro Wilk Test')
    stat, p = shapiro(df_list[i]['FelonyRatio'])
    print('Statistics=%.3f, p=%.3f ' % (stat, p))
    # interpret
    alpha = 0.05
    if p > alpha:
        print('Sample looks Gaussian (fail to reject H0)\n\n')
    else:
        print('Sample does not look Gaussian (reject H0)\n\n')
    print('Anderson-Darling Test\n')
    result = anderson(df_list[i]['FelonyRatio'])
    print('Statistic: %.3f' % result.statistic)
    p = 0
    for j in range(len(result.critical_values)):
        sl, cv = result.significance_level[j], result.critical_values[j]
        if result.statistic < result.critical_values[j]:
            print('%.3f: %.3f, data looks normal (fail to reject H0)\n\n' %
        else:
            print('%.3f: %.3f, data does not look normal (reject H0)\n\n' %
```

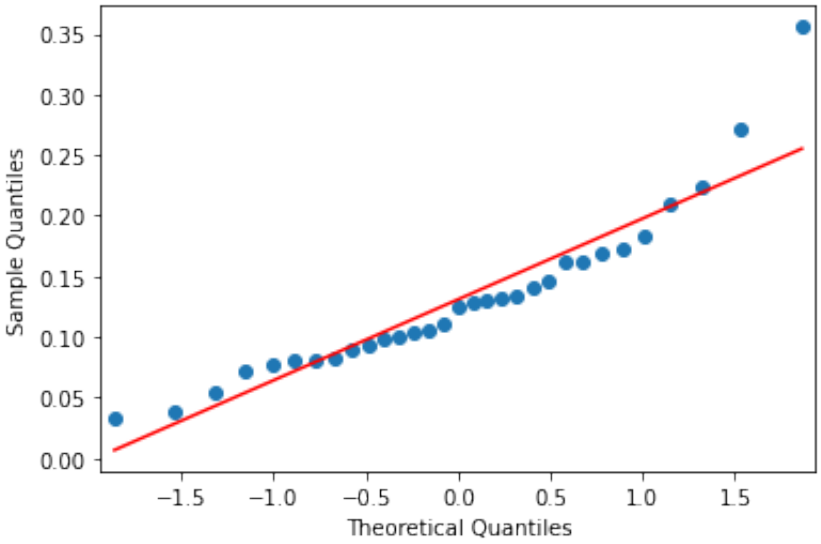
```
Test for Year:  2009

Q-Q Plot
```



```
Shapiro Wilk Test
Statistics=0.839, p=0.000
Sample does not look Gaussian (reject H0)


Anderson-Darling Test

Statistic: 1.623
15.000: 0.522, data does not look normal (reject H0)


10.000: 0.595, data does not look normal (reject H0)


5.000: 0.713, data does not look normal (reject H0)


2.500: 0.832, data does not look normal (reject H0)


1.000: 0.990, data does not look normal (reject H0)


Test for Year:  2010

Q-Q Plot
```



```
Shapiro Wilk Test
Statistics=0.850, p=0.000
Sample does not look Gaussian (reject H0)


Anderson-Darling Test

Statistic: 1.475
15.000: 0.522, data does not look normal (reject H0)


10.000: 0.595, data does not look normal (reject H0)


5.000: 0.713, data does not look normal (reject H0)


2.500: 0.832, data does not look normal (reject H0)


1.000: 0.990, data does not look normal (reject H0)


Test for Year:  2011

Q-Q Plot
```



```
Shapiro Wilk Test
Statistics=0.901, p=0.007
Sample does not look Gaussian (reject H0)


Anderson-Darling Test

Statistic: 0.826
15.000: 0.522, data does not look normal (reject H0)


10.000: 0.595, data does not look normal (reject H0)


5.000: 0.713, data does not look normal (reject H0)


2.500: 0.832, data looks normal (fail to reject H0)


1.000: 0.990, data looks normal (fail to reject H0)
```

Hence, we can conclude that the data is not normal. We will proceed with using the Chi-Square test.

A chi-squared test (also chi-square or χ2 test) is a statistical hypothesis test that is valid to perform when the test statistic is chi-squared distributed under the null hypothesis, specifically Pearson's chi-squared test and variants thereof. Pearson's chi-squared test is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table.

Let us first check if the Felony Crime Ratio is related to Police Strength.

In [29]:
```
result_2009["PolVPop"] = result_2009["2,009.00"]/result_2009["POPULATION 2(
```

In [30]:
```
result_2010["PolVPop"] = result_2010["2,009.00"]/result_2010["POPULATION 2(
```

In [31]:
```
result_2011["PolVPop"] = result_2011["2,009.00"]/result_2011["POPULATION 2(
```

In [32]:
```
def hypothesistesting(df, column1, column2):
    contigency_pct = pd.crosstab(df[column1], df[column2], normalize='index
#     print(contigency_pct)
    c, p, dof, expected = chi2_contingency(contigency_pct)
    print("p-value: ", p)
    if(p>=0.05):
        print("Do not reject Null Hypothesis, the columns are not independe
    else:
        print("Reject Null Hypothesis, the columns are independent")
```

In [33]:
```
hypothesistesting(result_2009, "FelonyRatio", "PolVPop")
```

```
p-value:  0.23719675904068735
Do not reject Null Hypothesis, the columns are not independent
```

In [34]:
```
hypothesistesting(result_2010, "FelonyRatio", "PolVPop")
```

```
p-value:  0.2371967590406865
Do not reject Null Hypothesis, the columns are not independent
```

In [35]:
```
hypothesistesting(result_2011, "FelonyRatio", "PolVPop")
```

```
p-value:  0.23719675904068735
Do not reject Null Hypothesis, the columns are not independent
```

Data Prediction

Since we are working with minimal data, we will implement statistical methods for predicting the consequent crime numbers in all the states. The method we will be employing is the Moving Average (MA) technique.

MA is a popular method to smooth out random movements in time-series data. Similar to a sliding window, an MA is an average that moves along the time scale/periods; older data points get dropped as newer data points are added.

Two types of MA are most preferred: **Simple MA** and **Exponential MA**.

In [36]:
```
crime_rate_df = pd.DataFrame()
crime_rate_df["STATE/UT"] = result_2009["STATE/UT"]
crime_rate_df["Crime_2009"] = result_2009["TotalCrimes"]
crime_rate_df["Crime_2010"] = result_2010["TotalCrimes"]
crime_rate_df["Crime_2011"] = result_2011["TotalCrimes"]
crime_rate_df.head(40)
```

Out[36]:

	STATE/UT	Crime_2009	Crime_2010	Crime_2011
14	KERALA	167585.0	202020.0	216821.0
15	LAKSHADWEEP	234.0	15.0	80.0
22	PUDUCHERRY	6821.0	5756.0	5877.0
10	HIMACHAL PRADESH	20002.0	19806.0	18684.0
1	ANDHRA PRADESH	257362.0	253717.0	265326.0
8	GUJARAT	167400.0	174372.0	179616.0
24	RAJASTHAN	201612.0	188582.0	193386.0
26	TAMIL NADU	225840.0	227070.0	237823.0
20	MIZORAM	2926.0	3072.0	2361.0
16	MADHYA PRADESH	351067.0	360209.0	354117.0
13	KARNATAKA	158680.0	171234.0	176511.0
17	MAHARASHTRA	329491.0	341679.0	348351.0
11	JAMMU & KASHMIR	37595.0	34966.0	48668.0
25	SIKKIM	977.0	1108.0	835.0
5	CHANDIGARH	2908.0	3170.0	3412.0
7	GOA	3650.0	4046.0	3861.0
6	DELHI	45363.0	42013.0	50928.0
4	BIHAR	219501.0	211439.0	267835.0
9	HARYANA	74764.0	72760.0	71886.0
29	UTTARAKHAND	13312.0	14761.0	12397.0
0	A&N ISLANDS	1208.0	1174.0	933.0
27	TRIPURA	9885.0	7723.0	10951.0
2	ARUNACHAL PRADESH	3271.0	3331.0	2779.0
23	PUNJAB	52065.0	55572.0	51381.0
12	JHARKHAND	57021.0	56615.0	56854.0
3	ASSAM	86134.0	83788.0	80243.0
28	UTTAR PRADESH	316681.0	323551.0	427881.0
21	NAGALAND	1463.0	1440.0	1396.0
30	WEST BENGAL	156128.0	166739.0	163499.0
19	MEGHALAYA	2171.0	2325.0	2729.0
18	MANIPUR	1657.0	1553.0	1782.0

Simple MA

SMA, short for Simple Moving Average, calculates the average over a specific number of periods in that range. The formula for SMA is:

$$SMA = \frac{P1 + P2 + ... + Pn}{N}$$

, where Pn = the crime rate at time point n, N = the number of time points.

For this exercise of building an SMA model, we'll use the Python code below.

In [37]:
```
crime_rate_df["Predicted_Crime_SMA"] = crime_rate_df.loc[:, ["Crime_2009",
crime_rate_df.head(40)
```

Out[37]:

	STATE/UT	Crime_2009	Crime_2010	Crime_2011	Predicted_Crime_SMA
14	KERALA	167585.0	202020.0	216821.0	195475.0
15	LAKSHADWEEP	234.0	15.0	80.0	110.0
22	PUDUCHERRY	6821.0	5756.0	5877.0	6151.0
10	HIMACHAL PRADESH	20002.0	19806.0	18684.0	19497.0
1	ANDHRA PRADESH	257362.0	253717.0	265326.0	258802.0
8	GUJARAT	167400.0	174372.0	179616.0	173796.0
24	RAJASTHAN	201612.0	188582.0	193386.0	194527.0
26	TAMIL NADU	225840.0	227070.0	237823.0	230244.0
20	MIZORAM	2926.0	3072.0	2361.0	2786.0
16	MADHYA PRADESH	351067.0	360209.0	354117.0	355131.0
13	KARNATAKA	158680.0	171234.0	176511.0	168808.0
17	MAHARASHTRA	329491.0	341679.0	348351.0	339840.0
11	JAMMU & KASHMIR	37595.0	34966.0	48668.0	40410.0
25	SIKKIM	977.0	1108.0	835.0	973.0
5	CHANDIGARH	2908.0	3170.0	3412.0	3163.0
7	GOA	3650.0	4046.0	3861.0	3852.0
6	DELHI	45363.0	42013.0	50928.0	46101.0
4	BIHAR	219501.0	211439.0	267835.0	232925.0
9	HARYANA	74764.0	72760.0	71886.0	73137.0
29	UTTARAKHAND	13312.0	14761.0	12397.0	13490.0
0	A&N ISLANDS	1208.0	1174.0	933.0	1105.0
27	TRIPURA	9885.0	7723.0	10951.0	9520.0
2	ARUNACHAL PRADESH	3271.0	3331.0	2779.0	3127.0
23	PUNJAB	52065.0	55572.0	51381.0	53006.0
12	JHARKHAND	57021.0	56615.0	56854.0	56830.0
3	ASSAM	86134.0	83788.0	80243.0	83388.0
28	UTTAR PRADESH	316681.0	323551.0	427881.0	356038.0
21	NAGALAND	1463.0	1440.0	1396.0	1433.0
30	WEST BENGAL	156128.0	166739.0	163499.0	162122.0
19	MEGHALAYA	2171.0	2325.0	2729.0	2408.0
18	MANIPUR	1657.0	1553.0	1782.0	1664.0

Exponential MA

Different from SMA, which assigns equal weights to all historical data points, EMA, short for Exponential Moving Average, applies higher weights to recent prices, i.e., tail data points of the 50-day MA in our example. The magnitude of the weighting factor depends on the number of time periods. The formula to calculate EMA is:

$$EMA = P_t * k + EMA_{t-1} * (1 - k),$$

where Pt = the price at time point t,

EMAt-1 = EMA at time point t-1,

N = number of time points in EMA,

and weighting factor k = 2/(N+1).

One advantage of the EMA over SMA is that EMA is more responsive to price changes, which makes it useful for short-term trading. Here's a Python implementation of EMA:

In [38]:
```
temp = crime_rate_df.loc[:, ["Crime_2009", "Crime_2010", "Crime_2011"]].ewm
crime_rate_df["Predicted_Crime_EMA"] = temp.mean(axis = 1).round(decimals =
crime_rate_df.head(40)
```

Out[38]:

	STATE/UT	Crime_2009	Crime_2010	Crime_2011	Predicted_Crime_SMA	Predicte
14	KERALA	167585.0	202020.0	216821.0	195475.0	
15	LAKSHADWEEP	234.0	15.0	80.0	110.0	
22	PUDUCHERRY	6821.0	5756.0	5877.0	6151.0	
10	HIMACHAL PRADESH	20002.0	19806.0	18684.0	19497.0	
1	ANDHRA PRADESH	257362.0	253717.0	265326.0	258802.0	
8	GUJARAT	167400.0	174372.0	179616.0	173796.0	
24	RAJASTHAN	201612.0	188582.0	193386.0	194527.0	
26	TAMIL NADU	225840.0	227070.0	237823.0	230244.0	
20	MIZORAM	2926.0	3072.0	2361.0	2786.0	
16	MADHYA PRADESH	351067.0	360209.0	354117.0	355131.0	
13	KARNATAKA	158680.0	171234.0	176511.0	168808.0	
17	MAHARASHTRA	329491.0	341679.0	348351.0	339840.0	
11	JAMMU & KASHMIR	37595.0	34966.0	48668.0	40410.0	

25	SIKKIM	977.0	1108.0	835.0	973.0
5	CHANDIGARH	2908.0	3170.0	3412.0	3163.0
7	GOA	3650.0	4046.0	3861.0	3852.0
6	DELHI	45363.0	42013.0	50928.0	46101.0
4	BIHAR	219501.0	211439.0	267835.0	232925.0
9	HARYANA	74764.0	72760.0	71886.0	73137.0
29	UTTARAKHAND	13312.0	14761.0	12397.0	13490.0
0	A&N ISLANDS	1208.0	1174.0	933.0	1105.0
27	TRIPURA	9885.0	7723.0	10951.0	9520.0
2	ARUNACHAL PRADESH	3271.0	3331.0	2779.0	3127.0
23	PUNJAB	52065.0	55572.0	51381.0	53006.0
12	JHARKHAND	57021.0	56615.0	56854.0	56830.0
3	ASSAM	86134.0	83788.0	80243.0	83388.0
28	UTTAR PRADESH	316681.0	323551.0	427881.0	356038.0
21	NAGALAND	1463.0	1440.0	1396.0	1433.0
30	WEST BENGAL	156128.0	166739.0	163499.0	162122.0
19	MEGHALAYA	2171.0	2325.0	2729.0	2408.0
18	MANIPUR	1657.0	1553.0	1782.0	1664.0

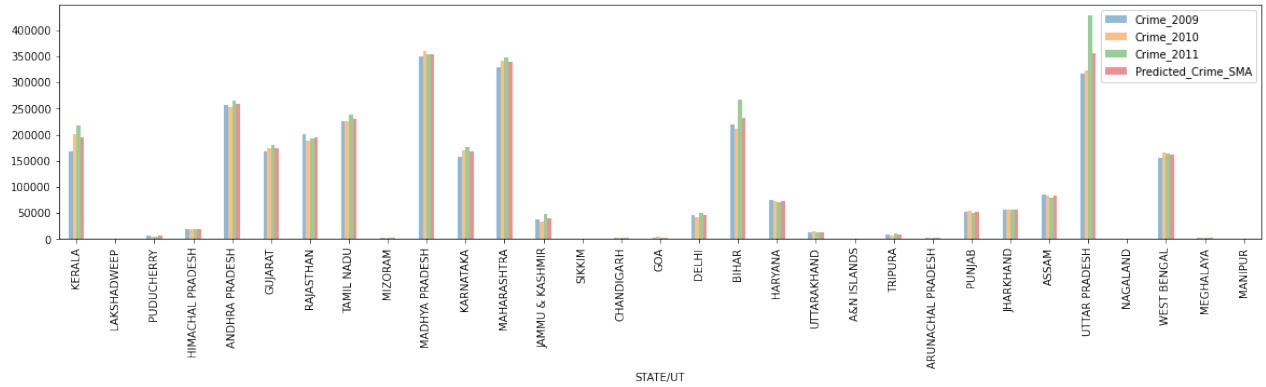
Data Visualisation

Here, we start visualising the data we have consolidated over the weeks and see if we can reach any conclusions based on the graphs plotted

Let us start with the visualisation of the predicted numbers of crime rate.

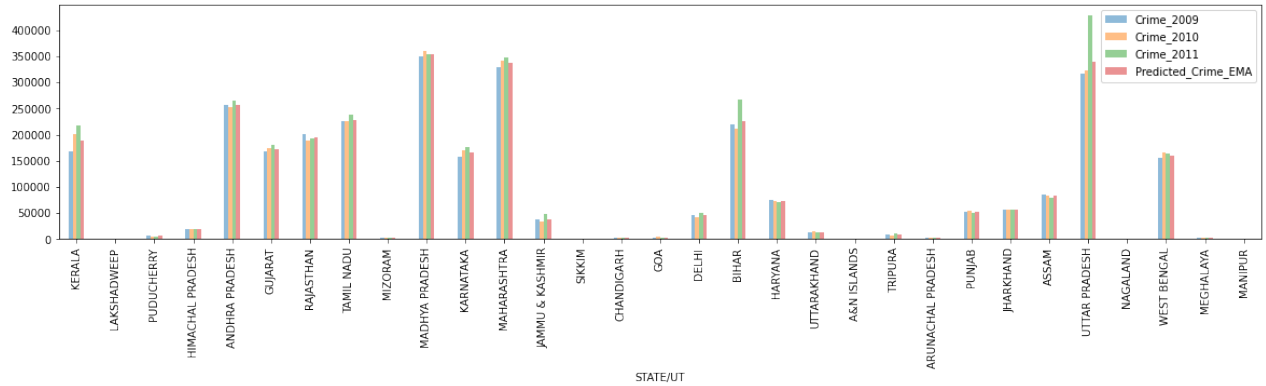
In [39]:
```
crime_rate_df.plot.bar(x='STATE/UT', y=["Crime_2009", "Crime_2010", "Crime_
```

Out[39]: <AxesSubplot:xlabel='STATE/UT'>



In [40]:
```
crime_rate_df.plot.bar(x='STATE/UT', y=["Crime_2009", "Crime_2010", "Crime_
```

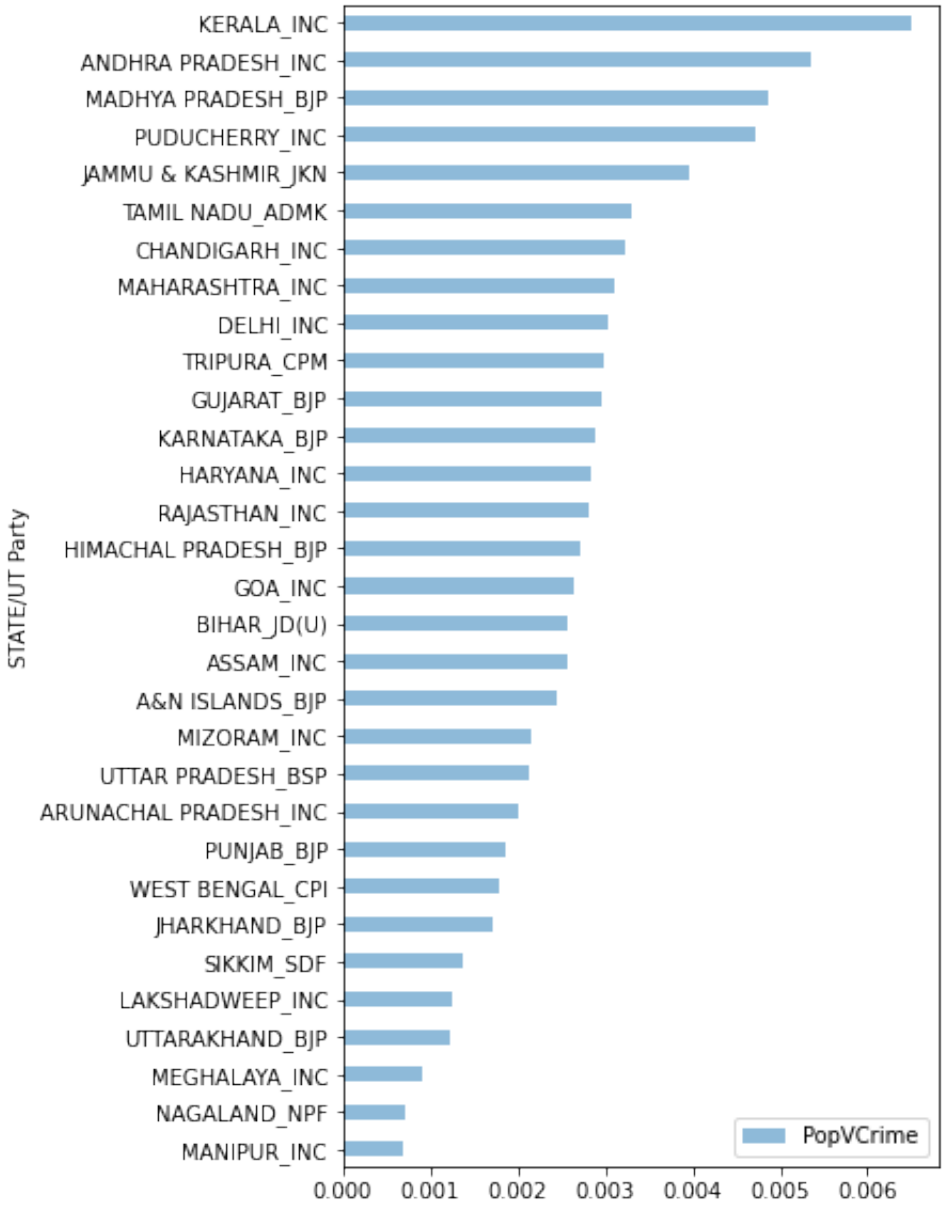
Out[40]: <AxesSubplot:xlabel='STATE/UT'>



As we have to stick to absolute numbers in this case the graph shown is not apt. Hence, moving forward we will use only ratios for visualisation, starting with the Crime per Capita for each state.

In [41]:
```
result_2011["PopVCrime"] = result_2011["TotalCrimes"]/result_2011["POPULATI
result_2011.sort_values(by=["PopVCrime"], inplace=True)
result_2011.plot.barh(x='STATE/UT Party', y="PopVCrime",  width = 0.4,alpha
```

Out[41]: <AxesSubplot:ylabel='STATE/UT Party'>

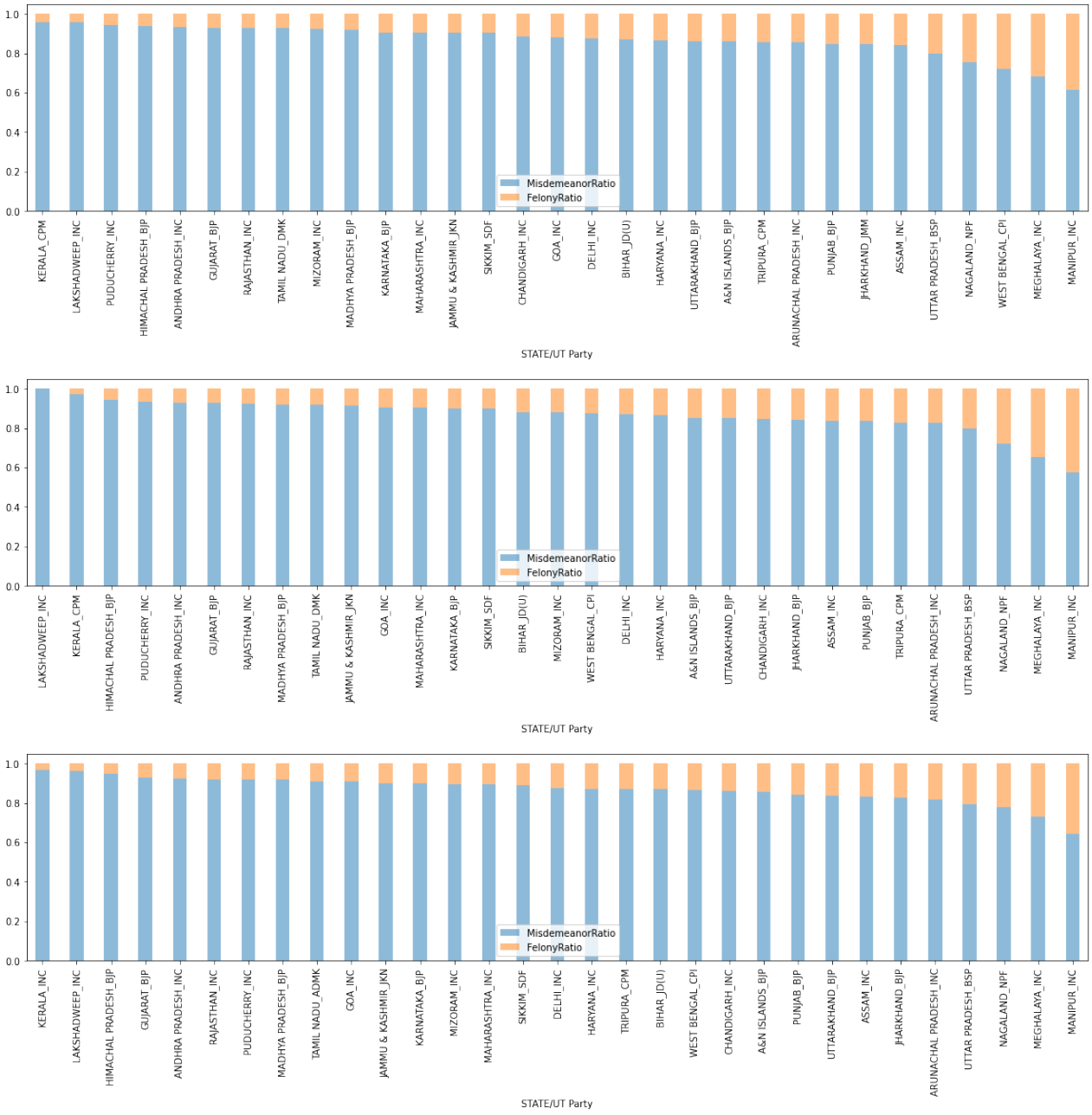


Here, we see that Kerala, led by the Indian National Congress, has the highest Crime committed per capita and Manipur which is also led by the INC has the lowest Crime committed per capita.

Now, lets check the ratio of felony and misdemeanors statewise and over the 3 years.

In [42]:
```
result_2011.sort_values(by=["FelonyRatio"], inplace=True)
result_2009.plot.bar(x='STATE/UT Party', y=["MisdemeanorRatio", "FelonyRati
result_2010.plot.bar(x='STATE/UT Party', y=["MisdemeanorRatio", "FelonyRati
result_2011.plot.bar(x='STATE/UT Party', y=["MisdemeanorRatio", "FelonyRati
```

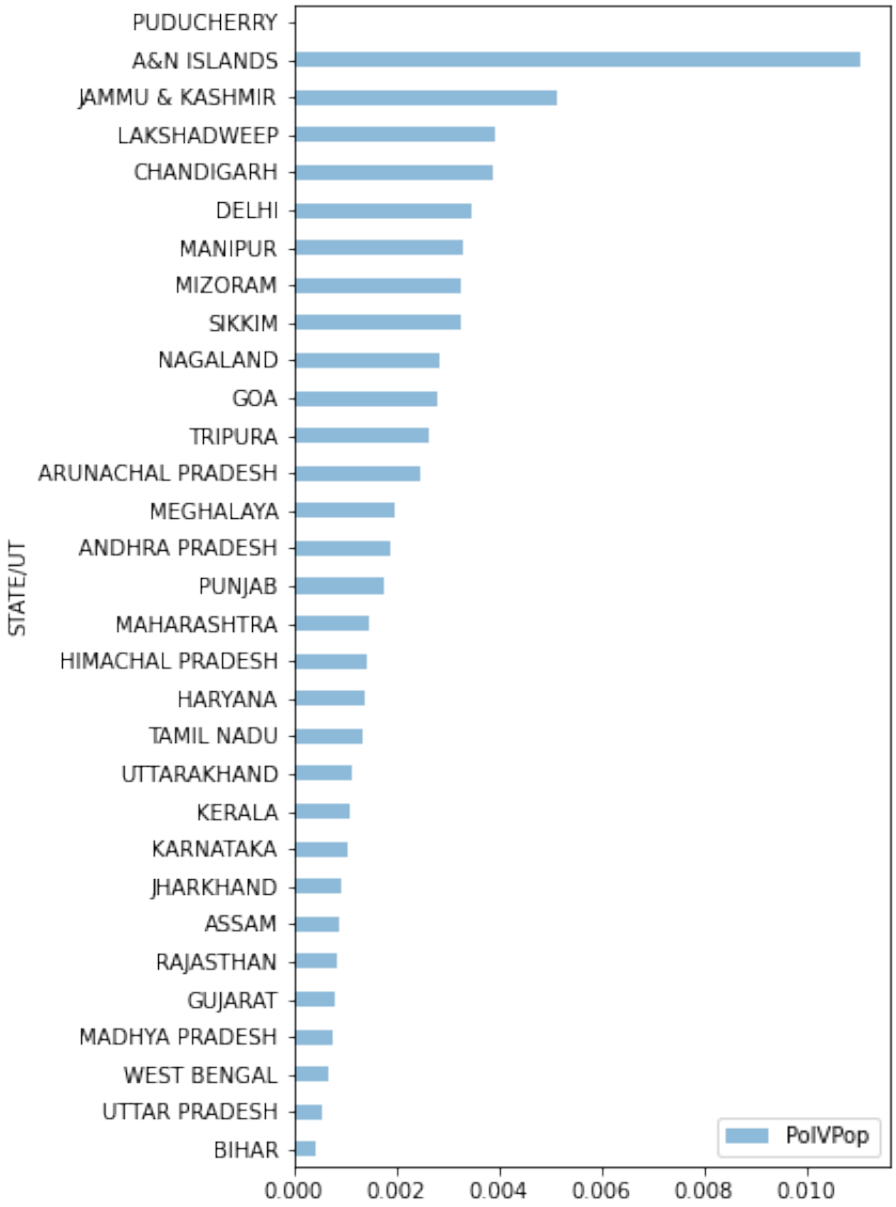
Out[42]: <AxesSubplot:xlabel='STATE/UT Party'>



It is very interesting to note that the state with the lowest Crime committed per capita had a very high ratio of Felonies committed compared to the states with relatively high crime per capita

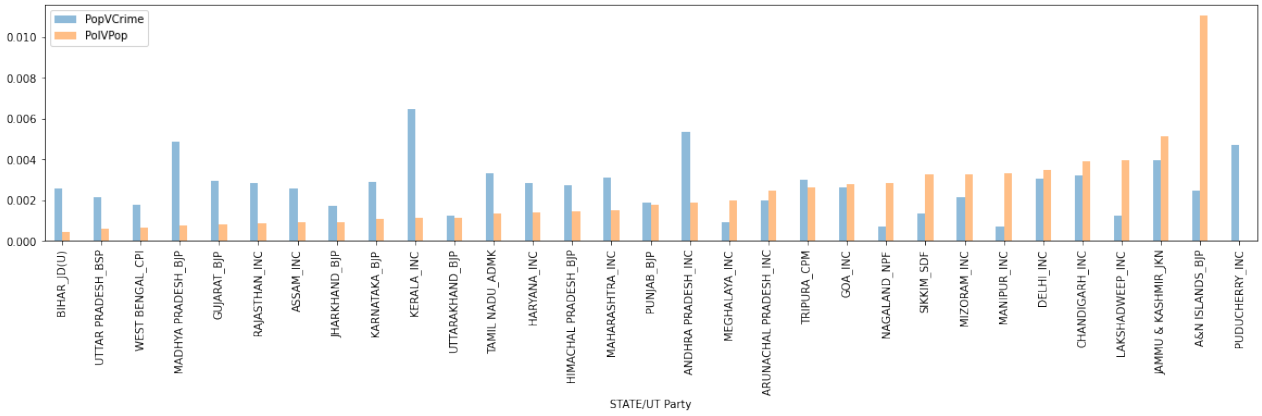
In [43]:
```
result_2011.sort_values(by=["PolVPop"], inplace=True)
result_2011.plot.barh(x='STATE/UT', y="PolVPop",  width = 0.4,alpha=0.5, fi
```

Out[43]: <AxesSubplot:ylabel='STATE/UT'>



In [44]:
```
result_2011.plot.bar(x='STATE/UT Party', y=["PopVCrime", "PolVPop"],  width
```

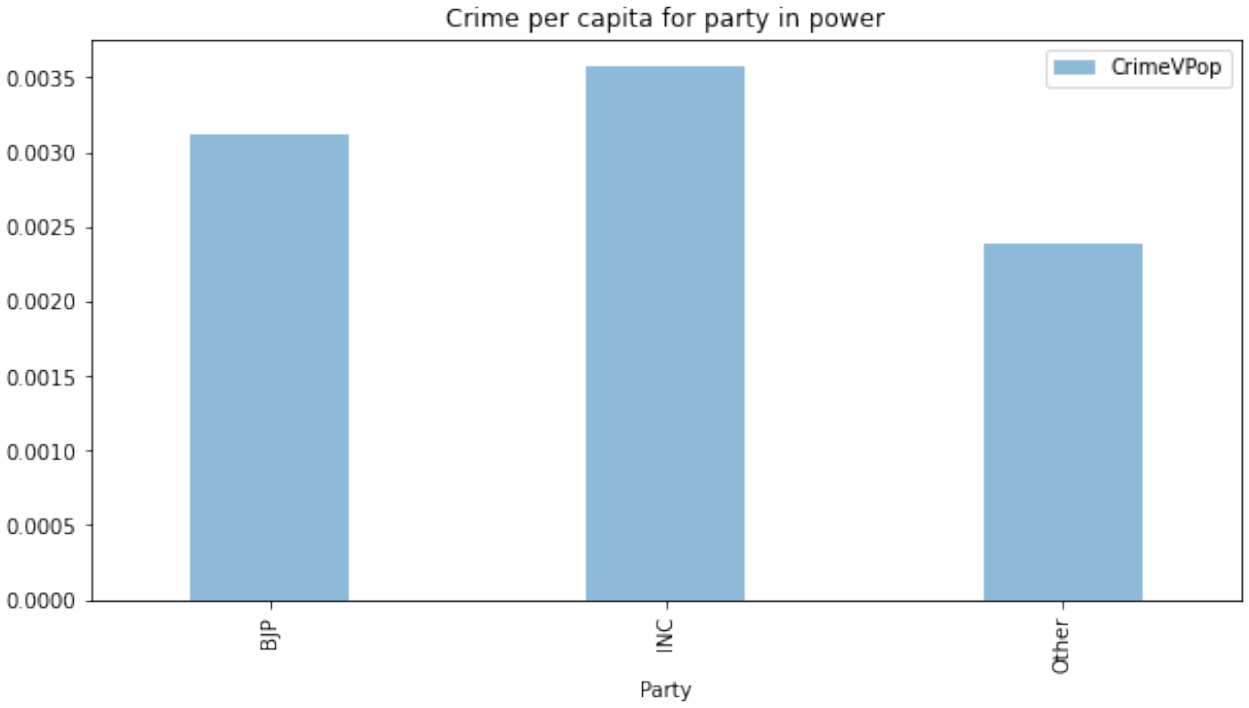
Out[44]: <AxesSubplot:xlabel='STATE/UT Party'>



We can clearly see, as the number of police per capita increases, the Crime numbers also drop.

In [45]:
```
data = {'Party':['BJP', 'INC', 'Other'],
        'Crime':[0,0,0],
       'Population':[0,0,0]}
df = pd.DataFrame(data)
for i in range(len(result_2011)):
    if (result_2011.loc[i, "Party"] == "BJP"):
        df.loc[0, ["Crime", "Population"]] = [df.iloc[0, 1]+result_2011.loc
    elif (result_2011.loc[i, "Party"] == "INC"):
        df.loc[1, ["Crime", "Population"]] = [df.iloc[1, 1]+result_2011.loc
    else:
        df.loc[2, ["Crime", "Population"]] = [df.iloc[2, 1]+result_2011.loc

df["CrimeVPop"] = df["Crime"]/df["Population"]
df.plot.bar(x='Party', y="CrimeVPop",  width = 0.4,alpha=0.5, figsize=[10,
```

Out[45]:
```
<AxesSubplot:title={'center':'Crime per capita for party in power'}, xlabel
='Party'>
```



Conclusion

Here is a list of trends observed:

1. States with High Crime per capita had lower percentage of felonies
2. States with greater police strength per capita had lower Crime
3. States under INC had higher Crime per capita compared to other parties