IMPORTING LIBRARIES

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

DATA PREPROCESSING

READING DATASET

```
dataset = pd.read_csv('/content/newtrial - Sheet 1 - 01_District_wise_crim 2.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
print(y)
```

```
[['A & N ISLANDS' 2001 13 ... 0 0 323]
 ['A & N ISLANDS' 2002 17 ... 0 0 328]
 ['A & N ISLANDS' 2003 21 ... 0 0 318]
 ...
 ['WEST BENGAL' 2010 2398 ... 8 2847 49096]
 ['WEST BENGAL' 2011 2109 ... 0 3249 56614]
 ['WEST BENGAL' 2012 2252 ... 12 4385 64482]]
[   658    608    644    748    682    676    807    882    941    980
    793    683 130089 143610 156951 158756 157123 173909 175087 179275
 180441 181438 189780 192522   2342   2228   2061   2256   2304   2294
   2286   2374   2362   2439   2286   2420  36877  36346  38195  40675
  42006  43673  45282  53333  55313  61668  66714  77682  88432  94040
  92263 108060  97850 100665 109420 122669 122931 127453 135896 146614
   3397   3806   2806   2889   3133   3126   3643   3931   3555   3373
   3542   3606  38460  37950  38449  41927  43633  45177  45845  51442
  51370  54958  57218  54598    350    349    338    409    434    435
    425    401    442    378    372    318    239    261    269    198
    243    288    260    248    276    203    224    239  54384  49137
  47404  53623  56065  57963  56065  49350  50251  51292  53353  54287
   2341   2440   2244   2127   2119   2204   2479   2742   3005   3293
   3449   3608 103419 106675 103709 105469 113414 120972 123195 123808
 115183 116439 123371 130121  38759  40152  38612  39096  42664  50509
  51597  55344  56229  59120  60741  62480  11499  12243  12011  12326
  12345  13093  14222  13976  13315  13049  14312  12557  19505  19967
  21233  21191  20115  20787  21443  20604  21975  23223  24504  24608
  25447  31439  32203  31439  35175  36364  38489  38686  37436  38889
  35838  40946 109098 113699 112405 114440 117580 117710 120606 127540
 134042 142322 137600 134021 103847 104200  98824 104025 104350 105255
 108530 110620 118369 148313 172137 158989     36     53     31     70
     42     80     56     95    134     42     44     60 181741 191799
 191078 196867 189172 194711 202386 206556 207762 214269 217094 220335
 171233 165462 164306 176302 187027 191788 195707 206243 199598 208168
 204902 202700   2489   2584   2537   2535   2913   2884   3259   3349
   2852   2715   3218   3737   1687   1664   1669   1752   1880   1935
   2079   2318   2448   2505   2755   2557   2246   2820   3456   1515
   2156   2073   2083   1989   2047   2174   1821   1766   1234   1114
    976    984   1049   1103   1180   1202   1059   1059   1083   1090
  46661  47728  47281  48739  51685  52792  54872  56755  55740  56459
  61277  67957   4068   4437   4517   4620   4575   4687   5054   4989
   4591   3935   4362   4281  27774  28794  28756  25630  27136  32068
  35793  35314  35545  36648  34883  35790 155185 151248 145579 154859
 140917 141992 148870 151174 166565 162957 165622 170948    444    485
    443    631    552    703    667    730    669    552    596    528
 154801 166942 157186 166606 162360 148972 172754 176833 174691 185678
 192879 200474   2801   3075   3514   3081   3356   3940   4273   5336
   5486   5805   5803   6264 178129 146037  95073 130181 122108 127001
 150258 168996 172884 174179 195135 198093   8073   7976   7923   8634
   8033   8412   9599   8856   8802   9240   8774   8882  61563  58962
  61174  69350  66406  68052  81102 105419 113036 129616 143197 161427]
```

MANAGING MISSING DATA - REPLACING MISSING ENTRIES WITH MEAN

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 2:31])
X[:, 2:31] = imputer.transform(X[:, 2:31])

print(X)
```

```
[['A & N ISLANDS' 2001 13.0 ... 0.0 0.0 323.0]
 ['A & N ISLANDS' 2002 17.0 ... 0.0 0.0 328.0]
 ['A & N ISLANDS' 2003 21.0 ... 0.0 0.0 318.0]
 ...
 ['WEST BENGAL' 2010 2398.0 ... 8.0 2847.0 49096.0]
 ['WEST BENGAL' 2011 2109.0 ... 0.0 3249.0 56614.0]
 ['WEST BENGAL' 2012 2252.0 ... 12.0 4385.0 64482.0]]
```

ONE HOT ENCODING ON STATES/UT

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from seaborn import load_dataset
import pandas as pd

df = X

transformer = make_column_transformer(
    (OneHotEncoder(), [0]),
    remainder='passthrough')

transformed = transformer.fit_transform(df)
X = transformed
print(X)
```

```
[[1.0 0.0 0.0 ... 0.0 0.0 323.0]
 [1.0 0.0 0.0 ... 0.0 0.0 328.0]
 [1.0 0.0 0.0 ... 0.0 0.0 318.0]
 ...
 [0.0 0.0 0.0 ... 8.0 2847.0 49096.0]
 [0.0 0.0 0.0 ... 0.0 3249.0 56614.0]
 [0.0 0.0 0.0 ... 12.0 4385.0 64482.0]]
```

```
print(X)
print(X.shape)
```

```
[[1.0 0.0 0.0 ... 0.0 0.0 323.0]
 [1.0 0.0 0.0 ... 0.0 0.0 328.0]
 [1.0 0.0 0.0 ... 0.0 0.0 318.0]
 ...
 [0.0 0.0 0.0 ... 8.0 2847.0 49096.0]
 [0.0 0.0 0.0 ... 0.0 3249.0 56614.0]
 [0.0 0.0 0.0 ... 12.0 4385.0 64482.0]]
(420, 65)
```

```
np.shape(X)
```

```
(420, 65)
```

SAVING DATASET TO A FILE AND DOWNLOADING IT TO VERIFY ONE HOT ENCODING

```
DF = pd.DataFrame(X)
DF.to_csv("ds (1).csv")
print(X[0][0])
```

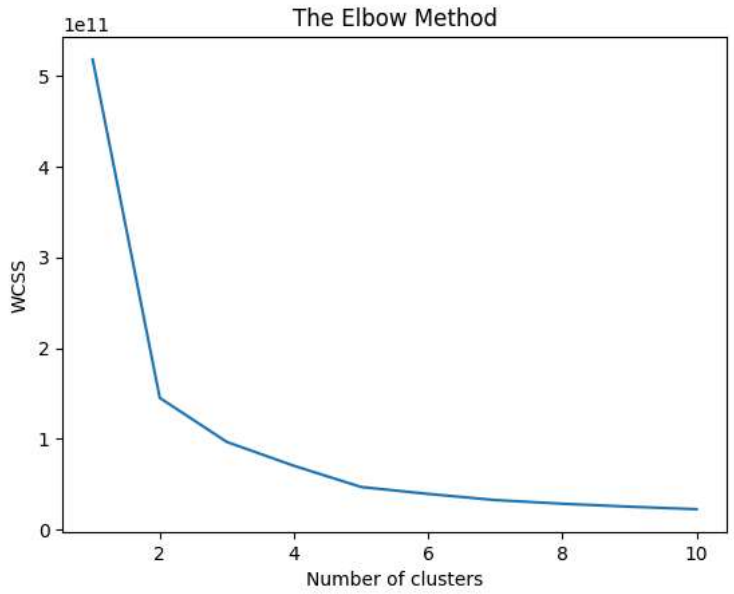
```
1.0
```

CLUSTERING TO FIND OPTIMAL NUMBER OF CLUSTERS

```
y = np.array(y)
Z = y
U = (X[:, 37:])
A = X[:, 36:]
print(X[:, 36:])
```

```
[[13.0 0.0 0.0 ... 0.0 0.0 323.0]
 [17.0 3.0 1.0 ... 0.0 0.0 328.0]
 [21.0 4.0 1.0 ... 0.0 0.0 318.0]
 ...
 [2398.0 2111.0 630.0 ... 8.0 2847.0 49096.0]
 [2109.0 2242.0 486.0 ... 0.0 3249.0 56614.0]
 [2252.0 2854.0 522.0 ... 12.0 4385.0 64482.0]]
```

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42, n_init=10)
    kmeans.fit(A)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```
# Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 2, init = 'k-means++', random_state = 42, n_init = 10)
y_kmeans = kmeans.fit_predict(A)
print(y_kmeans)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1]
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y_kmeans)
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1]
```

SPLITTING DATA

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
print(X_train)
print(X_train.shape)
```

```
[[0.0 0.0 0.0 ... 0.0 57.0 177.0]
 [0.0 0.0 0.0 ... 1.0 10933.0 49834.0]
 [0.0 0.0 0.0 ... 0.0 42.0 54106.0]
 ...
 [0.0 0.0 0.0 ... 0.0 441.0 2227.0]
 [0.0 0.0 0.0 ... 0.0 6008.0 100513.0]
 [0.0 0.0 0.0 ... 0.0 1957.0 8575.0]]
(336, 65)
```

```
print(X_test)
```

```
[[0.0 0.0 0.0 ... 0.0 203.0 1241.0]
 [0.0 0.0 0.0 ... 0.0 610.0 7020.0]
 [0.0 1.0 0.0 ... 0.0 11489.0 43722.0]
 ...
 [0.0 0.0 0.0 ... 0.0 12.0 161.0]
 [0.0 0.0 0.0 ... 0.0 2226.0 9316.0]
 [0.0 0.0 0.0 ... 2.0 2907.0 23091.0]]
```

```
print(y_train)
```

```
[0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0
 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 1
 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0
 1 0 0 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1
 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 0
 0 1 0]
```

```
print(y_test)
```

```
[0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0
 0 0 0 0 1 0 1 0 0 0]
```

```
print(X)
```

```
[[1.0 0.0 0.0 ... 0.0 0.0 323.0]
 [1.0 0.0 0.0 ... 0.0 0.0 328.0]
 [1.0 0.0 0.0 ... 0.0 0.0 318.0]
 ...
 [0.0 0.0 0.0 ... 8.0 2847.0 49096.0]
 [0.0 0.0 0.0 ... 0.0 3249.0 56614.0]
 [0.0 0.0 0.0 ... 12.0 4385.0 64482.0]]
```

FEATURE SCALING

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 37:] = sc.fit_transform(X_train[:, 37:])
X_test[:, 37:] = sc.transform(X_test[:, 37:])
```

```
print(X_train)
```

```
[[0.0 0.0 0.0 ... -0.24230949037789695 -0.6355423218652001
  -0.8087924772174302]
 [0.0 0.0 0.0 ... -0.13885842460029418 2.4298134415889105
  0.9351913193329178]
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.6397700100273748
  1.0852265364999454]
 ...
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.5273135049135287
  -0.7367952405150878]
 [0.0 0.0 0.0 ... -0.24230949037789695 1.0417224950082289
  2.715068372424629]
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.10003515465640915
  -0.5138496509607127]]
```

```
print(X_test)
```

```
[[0.0 0.0 0.0 ... -0.24230949037789695 -0.5943928237533667
  -0.7714241553387511]
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.4796815516196941
  -0.5684621890446845]
 [0.0 1.0 0.0 ... -0.24230949037789695 2.5865197494668513
  0.7205341921501294]
 ...
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.6482253863517241
  -0.8093544068697411]
 [0.0 0.0 0.0 ... -0.24230949037789695 -0.02421861361474348
  -0.4878252839380611]
 [0.0 0.0 0.0 ... -0.035407358822691415 0.16771842894798633
  -0.0040389739015900565]]
```

ANALYSIS OF CRIME RATE

```
dataset.describe()
```

	YEAR	MURDER	ATTEMPT TO MURDER	CULPABLE HOMICIDE NOT AMOUNTING TO MURDER	RAPE	CUSTODIAL RAPE	OTHER RA
count	420.000000	420.000000	420.000000	420.00000	420.00000	420.000000	420.000(
mean	2006.500000	958.435714	838.040476	106.22381	569.37381	0.061905	569.311
std	3.456169	1213.687365	1149.623253	253.86659	694.97765	0.392016	694.930
min	2001.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.000(
25%	2003.750000	46.000000	32.000000	4.00000	29.00000	0.000000	29.000(
50%	2006.500000	463.500000	434.000000	29.00000	298.50000	0.000000	298.500(
75%	2009.250000	1461.000000	1302.250000	92.25000	936.00000	0.000000	935.250(
max	2012.000000	7601.000000	7964.000000	1616.00000	3425.00000	5.000000	3425.000(

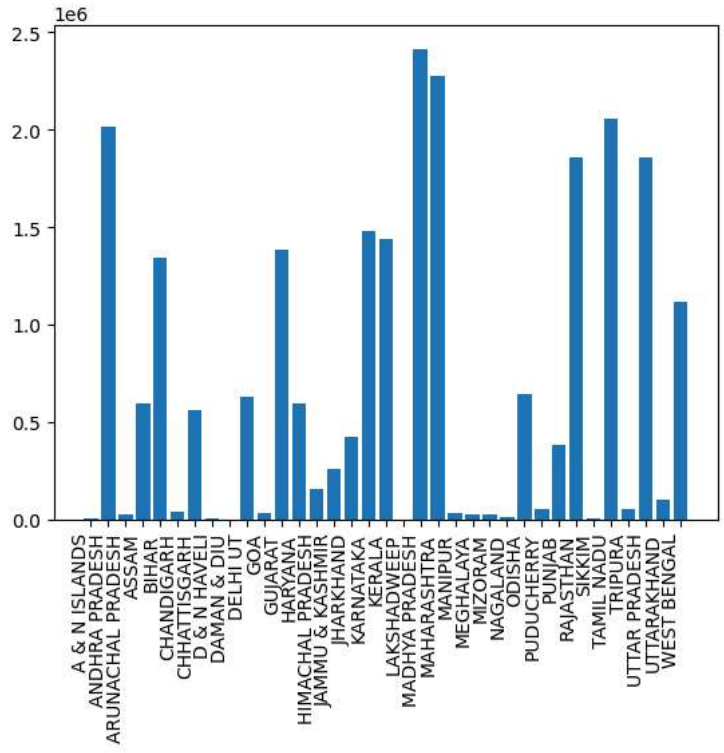
8 rows × 31 columns

```
Start coding or generate with AI.
```

SATE VS TOTAL CRIME OVER 10 YEARS

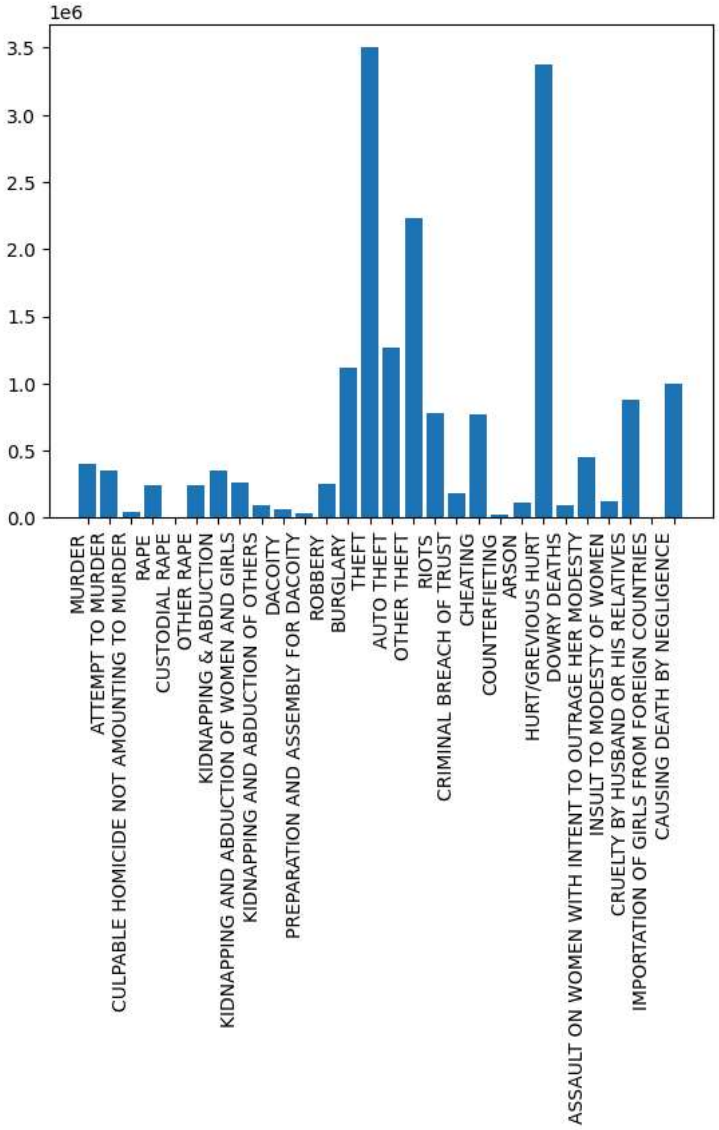
```
import matplotlib.pyplot as plt
import seaborn as sns

df_sum_by_state = dataset.groupby('STATE/UT')['TOTAL IPC CRIMES'].sum().reset_index()
states = df_sum_by_state['STATE/UT']
sum = df_sum_by_state['TOTAL IPC CRIMES']
print(df_sum_by_state)
fig, ax = plt.subplots()
plt.xticks(rotation=90, ha='right')
ax.bar(states, sum)
plt.show()
```

```
              STATE/UT  TOTAL IPC CRIMES
0       A & N ISLANDS              9102
1      ANDHRA PRADESH           2018981
2   ARUNACHAL PRADESH             27652
3               ASSAM            597764
4               BIHAR           1346293
5          CHANDIGARH             40807
6        CHHATTISGARH            561027
7        D & N HAVELI              4651
8         DAMAN & DIU              2948
9           DELHI UT            633174
10               GOA             32051
11           GUJARAT           1385775
12           HARYANA            595303
13   HIMACHAL PRADESH            154948
14    JAMMU & KASHMIR            259155
15          JHARKHAND            422351
16          KARNATAKA           1481063
17             KERALA           1437459
18        LAKSHADWEEP               743
19     MADHYA PRADESH           2413770
20        MAHARASHTRA           2273436
21            MANIPUR             35072
22          MEGHALAYA             25249
23            MIZORAM             26146
24           NAGALAND             13133
25             ODISHA            647946
26         PUDUCHERRY             54116
27             PUNJAB            384131
28          RAJASTHAN           1855916
29             SIKKIM              7000
30         TAMIL NADU           2060176
31            TRIPURA             52734
32      UTTAR PRADESH           1858074
33        UTTARAKHAND            103204
34        WEST BENGAL           1119304
```



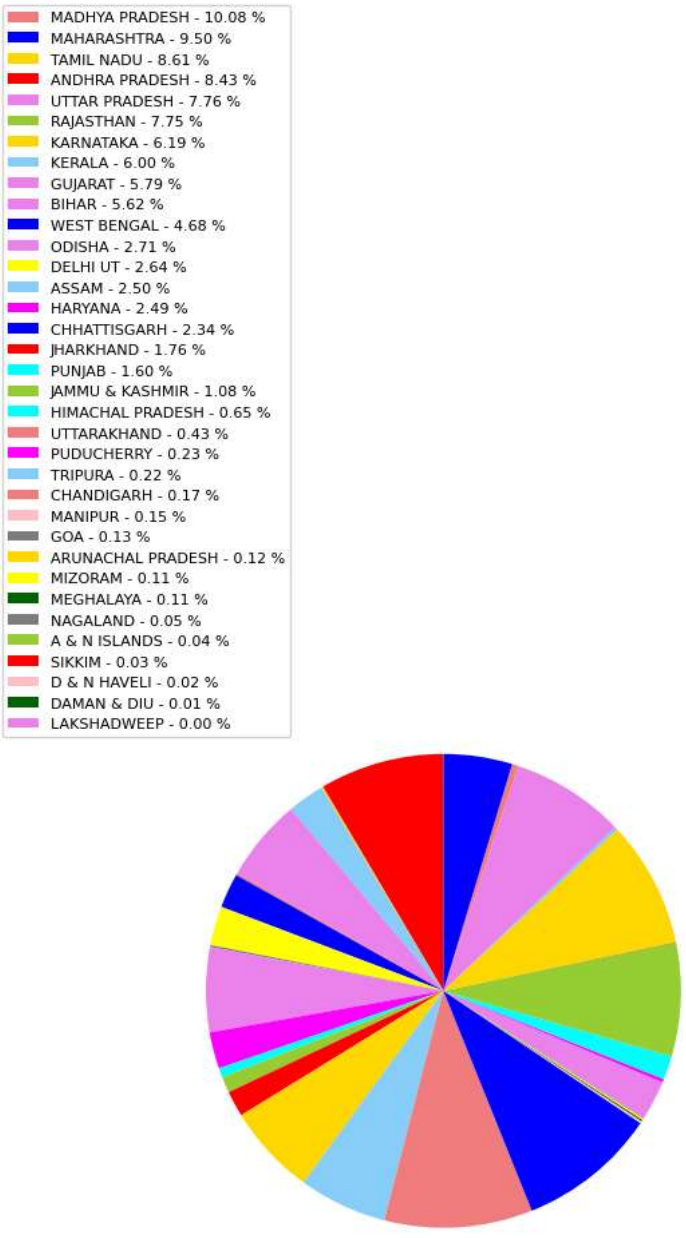
TYPE OF CRIME V/S RATE OF THAT CRIME

```
import matplotlib.pyplot as plt
import seaborn as sns


sum_column = dataset.sum(axis=0)
sum_col = sum_column
f = np.array(sum_col[2:30])
crimes = dataset.columns.values[2:30]

fig, ax = plt.subplots()
plt.xticks(rotation=90, ha='right')
ax.bar(crimes, f)

plt.show()
```



PIE CHART OF CRIME RATE PER STATE

```
import matplotlib.pyplot as plt
import seaborn as sns

df_sum_by_state = dataset.groupby('STATE/UT')['TOTAL IPC CRIMES'].sum().reset_index()
states = df_sum_by_state['STATE/UT']
sum = df_sum_by_state['TOTAL IPC CRIMES']

x = states
y = sum
colors = ['yellowgreen','red','gold','lightskyblue','violet','lightcoral','blue','pink', 'darkgreen','yellow','grey','violet','magenta','cya
porcent = 100.*y/y.sum()

patches, texts = plt.pie(y, colors=colors, startangle=90, radius=1.2)
labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(x, porcent)]

sort_legend = True
if sort_legend:
    patches, labels, dummy =  zip(*sorted(zip(patches, labels, y),
                                          key=lambda x: x[2],
                                          reverse=True))

plt.legend(patches, labels, loc='lower center', bbox_to_anchor=(-0.1, 1.),
           fontsize=8)

plt.savefig('piechart.png', bbox_inches='tight')
```



APPLYING MODELS STARTS :-

Building the ANN

Initializing the ANN

```
import numpy as np
import pandas as pd
import tensorflow as tf
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the second hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the ANN

Compiling the ANN

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_train = np.asarray(X_train).astype('float32')
y_train = np.asarray(y_train).astype('float32')
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
11/11 [==============================] - 1s 2ms/step - loss: 79.0217 - accuracy: 0.2976
Epoch 2/100
11/11 [==============================] - 0s 2ms/step - loss: 63.2351 - accuracy: 0.2976
Epoch 3/100
11/11 [==============================] - 0s 2ms/step - loss: 48.5180 - accuracy: 0.2976
Epoch 4/100
11/11 [==============================] - 0s 2ms/step - loss: 36.2006 - accuracy: 0.2976
Epoch 5/100
11/11 [==============================] - 0s 2ms/step - loss: 25.1489 - accuracy: 0.2976
Epoch 6/100
11/11 [==============================] - 0s 2ms/step - loss: 15.0859 - accuracy: 0.2976
Epoch 7/100
11/11 [==============================] - 0s 2ms/step - loss: 7.4394 - accuracy: 0.6845
Epoch 8/100
11/11 [==============================] - 0s 2ms/step - loss: 4.5786 - accuracy: 0.7589
Epoch 9/100
11/11 [==============================] - 0s 2ms/step - loss: 3.0048 - accuracy: 0.8065
Epoch 10/100
11/11 [==============================] - 0s 3ms/step - loss: 2.0314 - accuracy: 0.8482
Epoch 11/100
```

```
11/11 [==============================] - 0s 3ms/step - loss: 1.4151 - accuracy: 0.8631
Epoch 12/100
11/11 [==============================] - 0s 3ms/step - loss: 1.2370 - accuracy: 0.8631
Epoch 13/100
11/11 [==============================] - 0s 2ms/step - loss: 1.1547 - accuracy: 0.8720
Epoch 14/100
11/11 [==============================] - 0s 2ms/step - loss: 1.0809 - accuracy: 0.8720
Epoch 15/100
11/11 [==============================] - 0s 3ms/step - loss: 1.0116 - accuracy: 0.8661
Epoch 16/100
11/11 [==============================] - 0s 2ms/step - loss: 0.9577 - accuracy: 0.8661
Epoch 17/100
11/11 [==============================] - 0s 2ms/step - loss: 0.9126 - accuracy: 0.8661
Epoch 18/100
11/11 [==============================] - 0s 2ms/step - loss: 0.8776 - accuracy: 0.8750
Epoch 19/100
11/11 [==============================] - 0s 2ms/step - loss: 0.8444 - accuracy: 0.8720
Epoch 20/100
11/11 [==============================] - 0s 2ms/step - loss: 0.8210 - accuracy: 0.8661
Epoch 21/100
11/11 [==============================] - 0s 2ms/step - loss: 0.8004 - accuracy: 0.8661
Epoch 22/100
11/11 [==============================] - 0s 2ms/step - loss: 0.7828 - accuracy: 0.8720
Epoch 23/100
11/11 [==============================] - 0s 2ms/step - loss: 0.7673 - accuracy: 0.8810
Epoch 24/100
11/11 [==============================] - 0s 3ms/step - loss: 0.7530 - accuracy: 0.8780
Epoch 25/100
11/11 [==============================] - 0s 2ms/step - loss: 0.7449 - accuracy: 0.8958
Epoch 26/100
11/11 [==============================] - 0s 3ms/step - loss: 0.7286 - accuracy: 0.8899
Epoch 27/100
11/11 [==============================] - 0s 4ms/step - loss: 0.7154 - accuracy: 0.8899
Epoch 28/100
11/11 [==============================] - 0s 3ms/step - loss: 0.7143 - accuracy: 0.8899
Epoch 29/100
```

Double-click (or enter) to edit

```
X_test = np.asarray(X_test).astype('float32')
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
3/3 [==============================] - 0s 3ms/step
[[0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
```

```
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[1 1]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[59  2]
 [ 2 21]]
0.9523809523809523
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
print(X_train)
print(y_train)
```

```
[[ 0.          0.          0.         ... -0.2423095  -0.63554233
  -0.8087925 ]
 [ 0.          0.          0.         ... -0.13885842  2.4298134
   0.93519133]
 [ 0.          0.          0.         ... -0.2423095  -0.63977003
   1.0852265 ]
 ...
 [ 0.          0.          0.         ... -0.2423095  -0.52731353
  -0.73679525]
 [ 0.          0.          0.         ... -0.2423095   1.0417225
   2.7150683 ]
 [ 0.          0.          0.         ... -0.2423095  -0.10003515
  -0.5138497 ]]
[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1.
 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0.
 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0.
 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1.
 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0.]
```

Kernel SVM model

```
# Training the Kernel SVM model on the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

```
SVC
SVC(random_state=0)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[58  3]
 [ 5 18]]
0.9047619047619048
```

SVM model

```
# Training the SVM model on the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
```

K-NN model

```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)


# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
```

Logistic Regression

```
# Training the Logistic Regression model on the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)


# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
```

Random Forest Regression

```
# Training the Random Forest Regression model on the whole dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))


from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
```

Decision Tree Regression

```
# Training the Decision Tree Regression model on the whole dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 1.]
 [1. 1.]
 [0. 0.]
```

SVR model

```
# Training the SVR model on the whole dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
```

```
SVR()
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 0.]
 [1. 1.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
```

```
[0. 0.]
[0. 1.]
[1. 1.]
[0. 0.]
```

Double-click (or enter) to edit

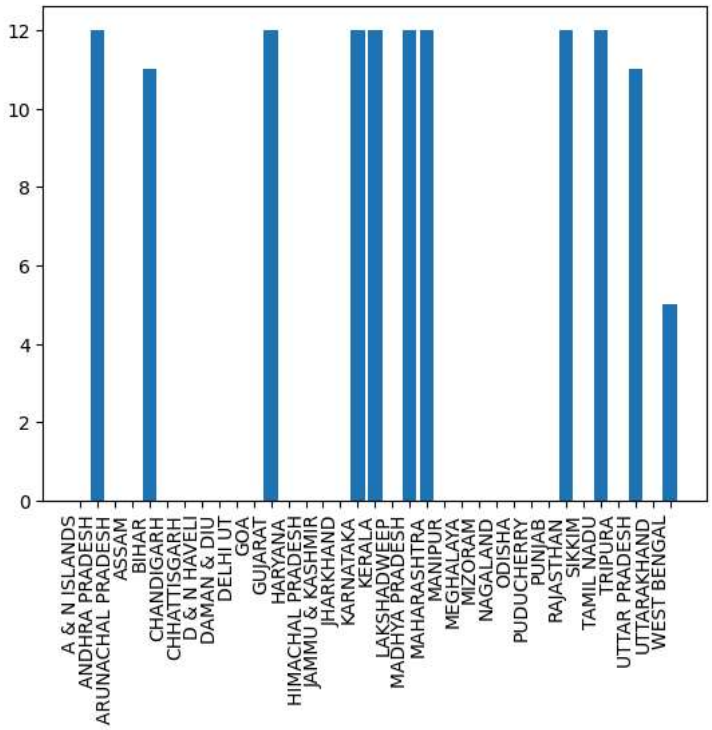
RESULTS

CLASSIFYING STATES AS SAFE/UNSAFE

STATE WITH HIGHER BARS ARE UNSAFE

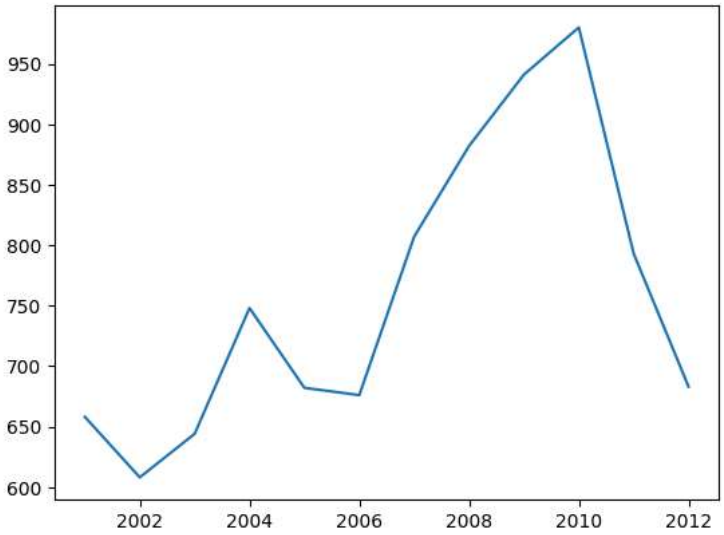
```
print(y_kmeans)
dataset['category'] = y_kmeans
df_sum_by_state = dataset.groupby('STATE/UT')['category'].sum().reset_index()
states = df_sum_by_state['STATE/UT']
sum = df_sum_by_state['category']
#print(df_sum_by_state)
fig, ax = plt.subplots()
plt.xticks(rotation=90, ha='right')
ax.bar(states, sum)
# Display the resulting DataFrame
plt.show()
#print(dataset.head(10))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1]
```

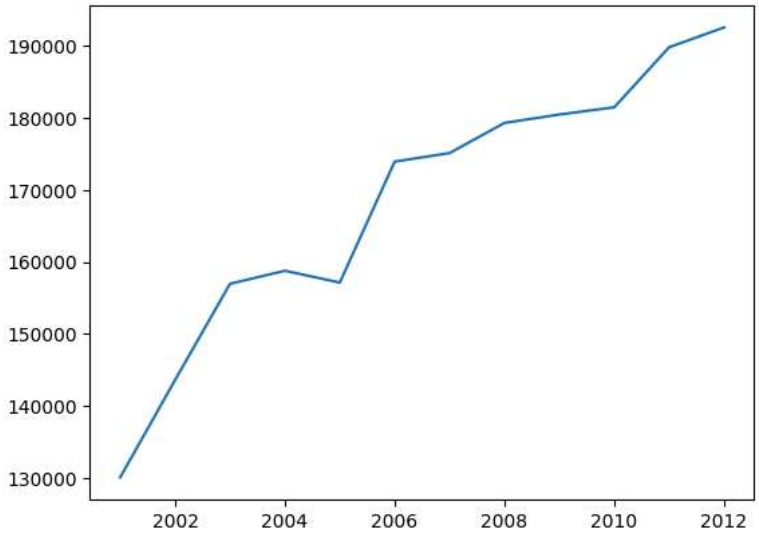


```
import seaborn as sns
import numpy as np
import pandas as pd

states = (dataset.iloc[:, 0].unique())
for state in states:
  print( state, " : \n")
  data = dataset[dataset['STATE/UT'] == state]
  grouped = data.groupby('YEAR').agg('TOTAL IPC CRIMES').sum()
  arr = np.array(grouped)
  year = (dataset.iloc[:, 1].unique())
  plt.figure()
  plt.plot(year, arr)
  plt.show()
  print("\n")
```

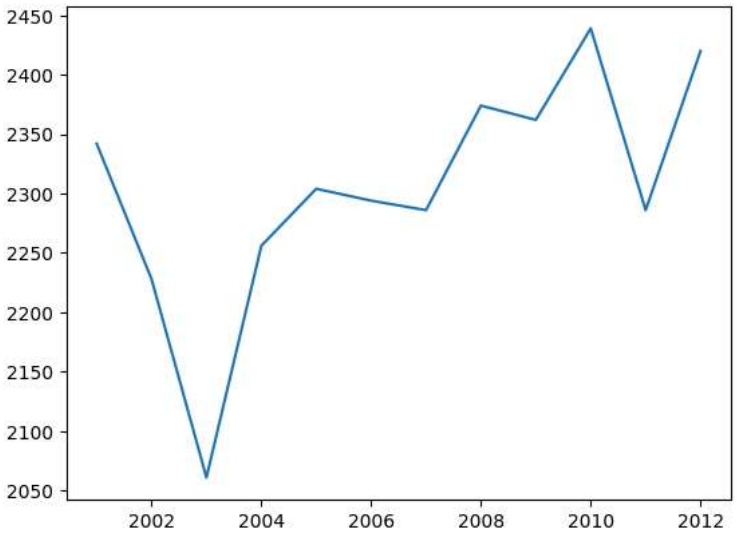
A & N ISLANDS :



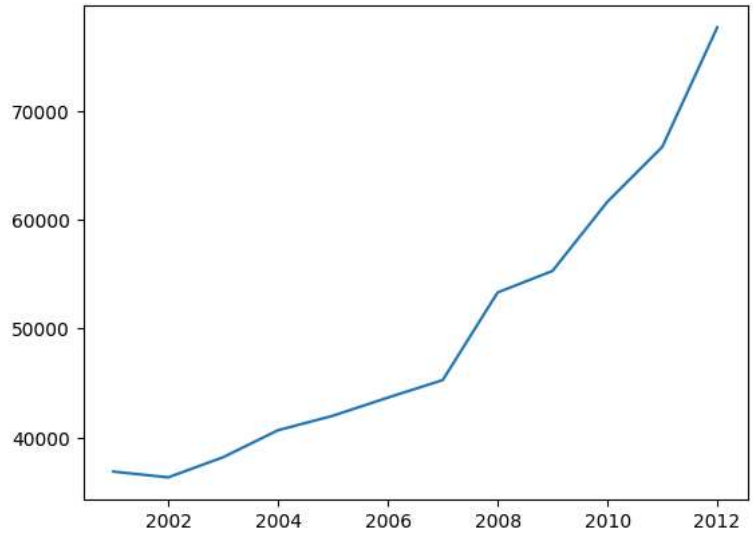
ANDHRA PRADESH :



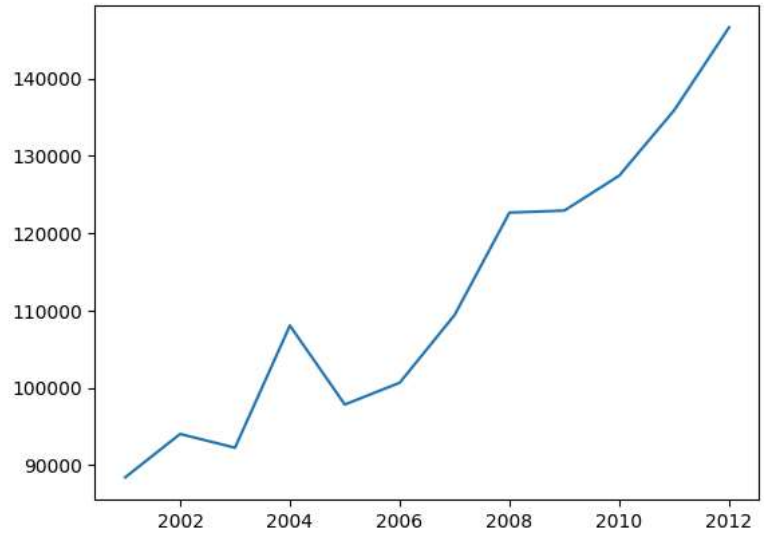
ARUNACHAL PRADESH :



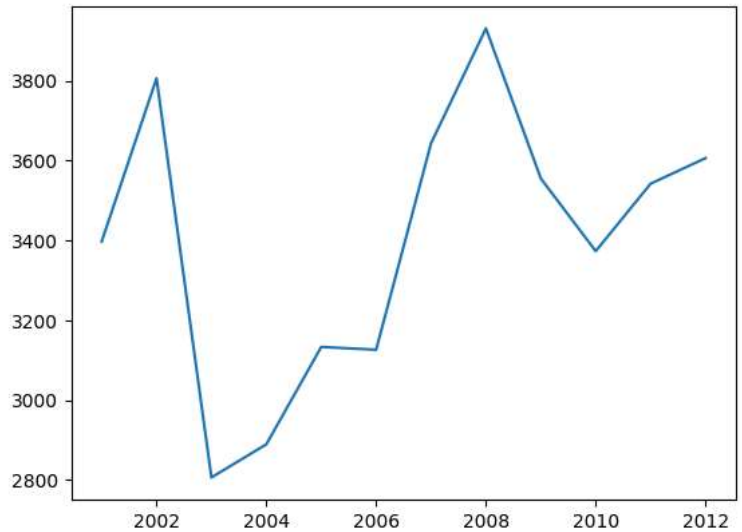
ASSAM :



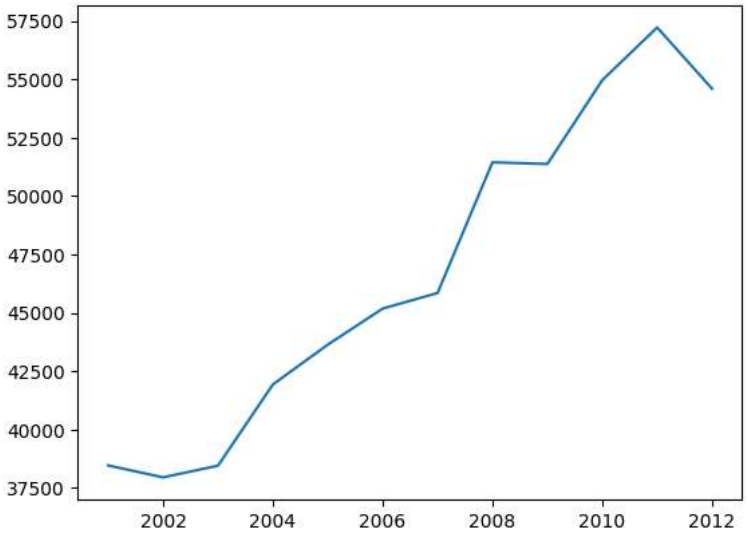
BIHAR :



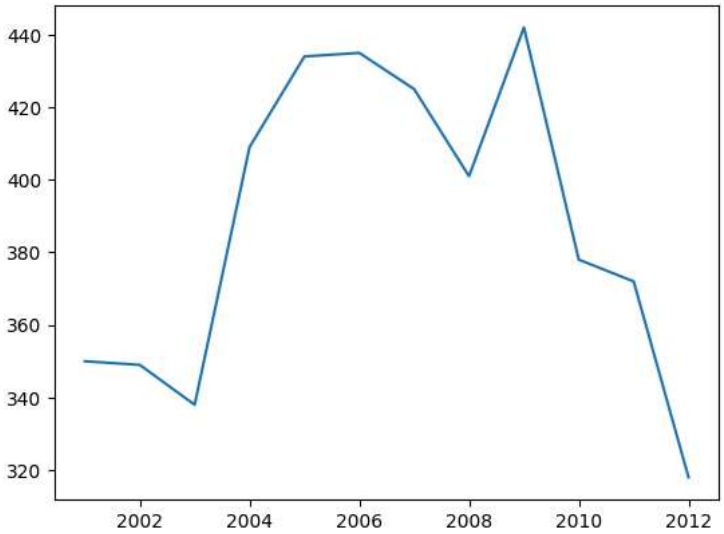
CHANDIGARH :



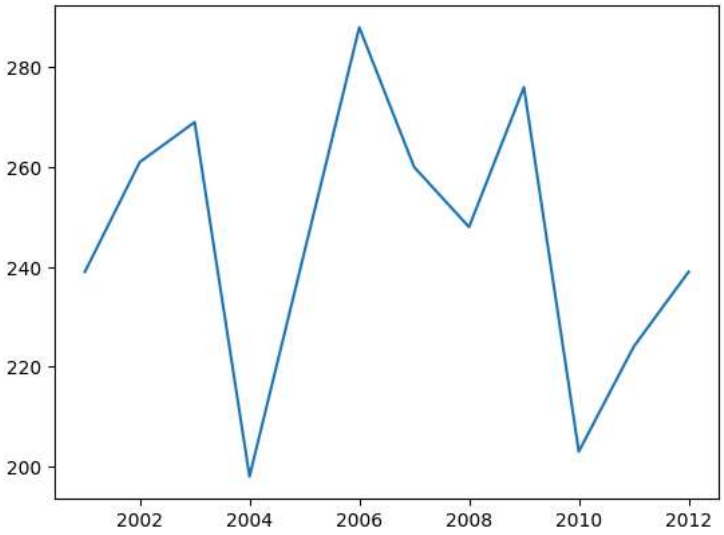
CHHATTISGARH :



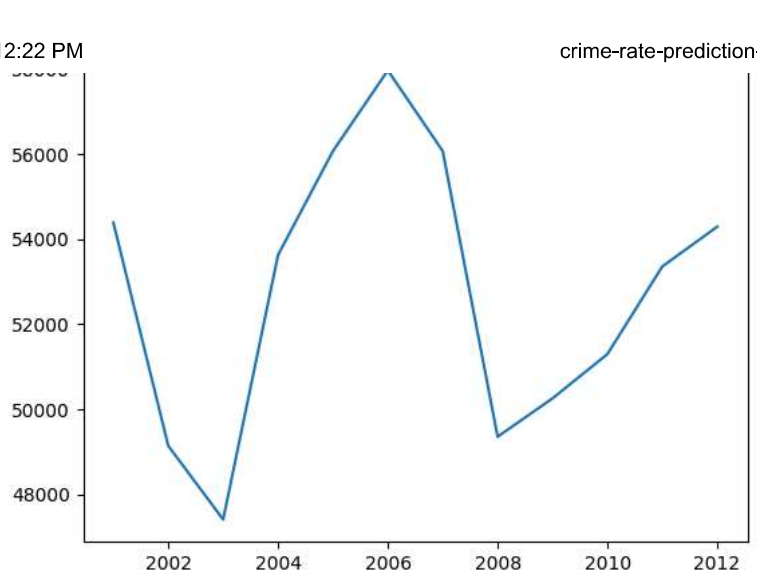
D & N HAVELI :



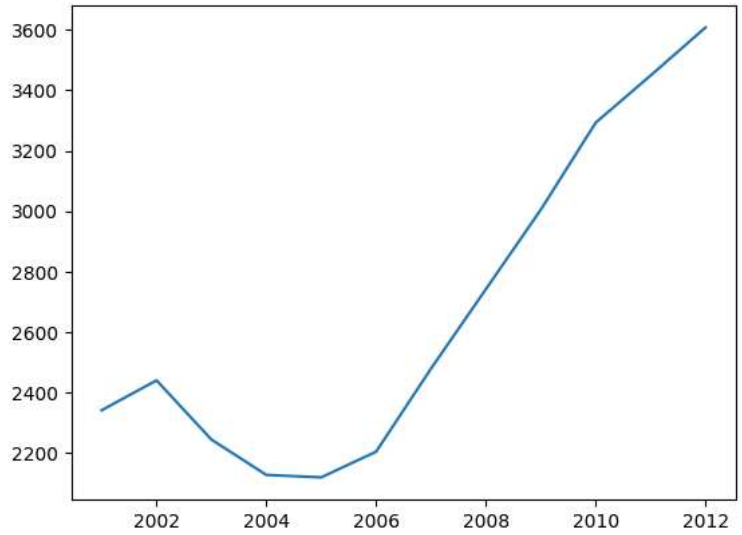
DAMAN & DIU :



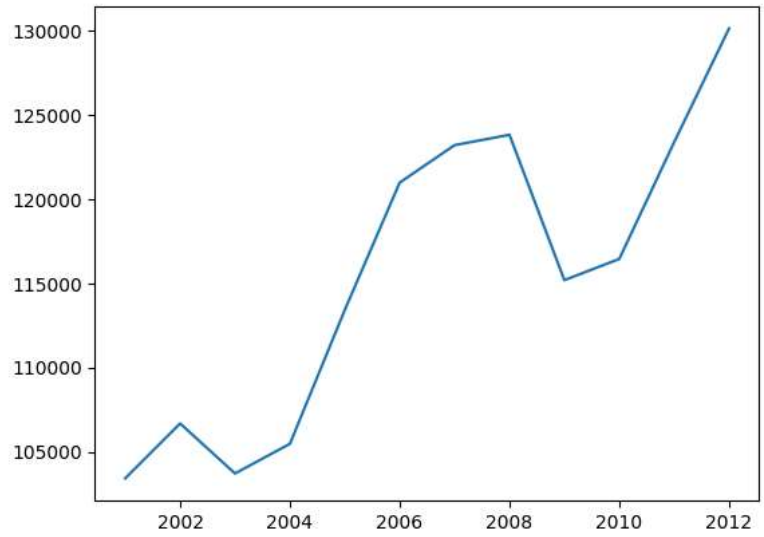
DELHI UT :



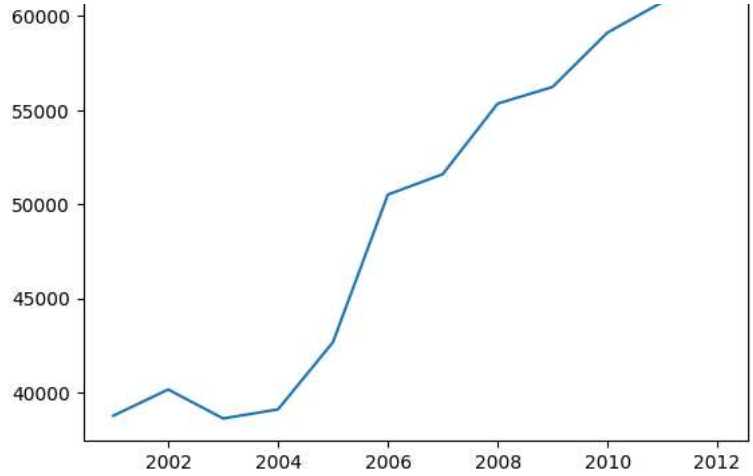
GOA :



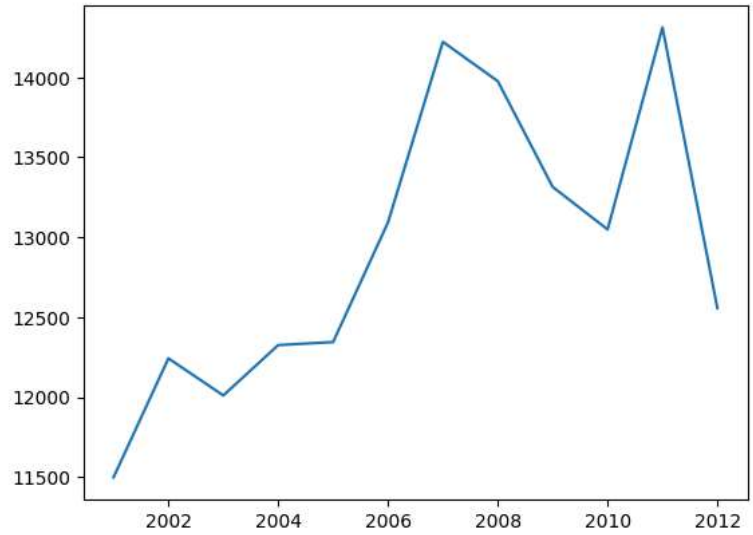
GUJARAT :



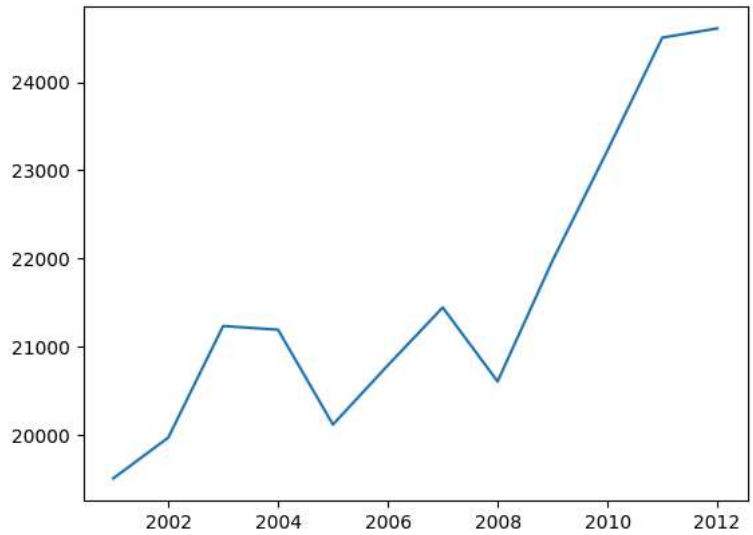
HARYANA :



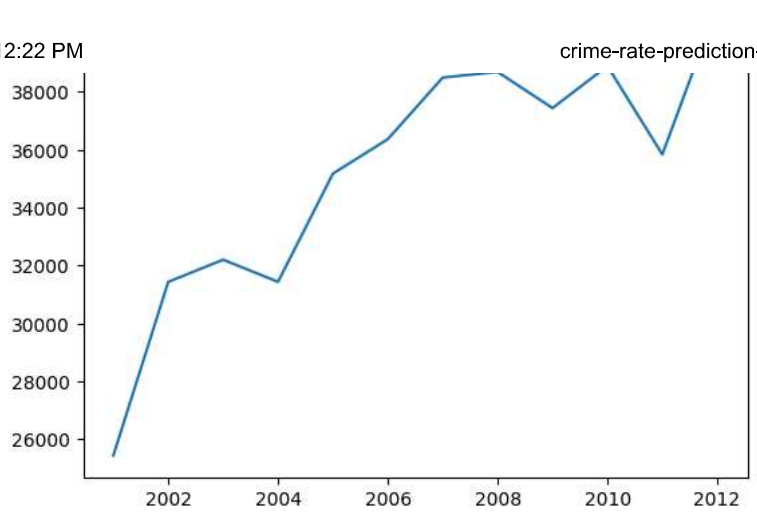
HIMACHAL PRADESH :



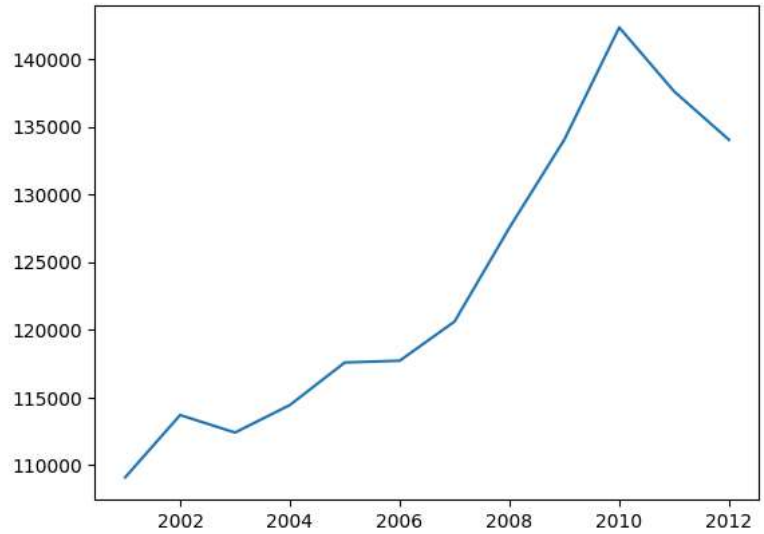
JAMMU & KASHMIR :



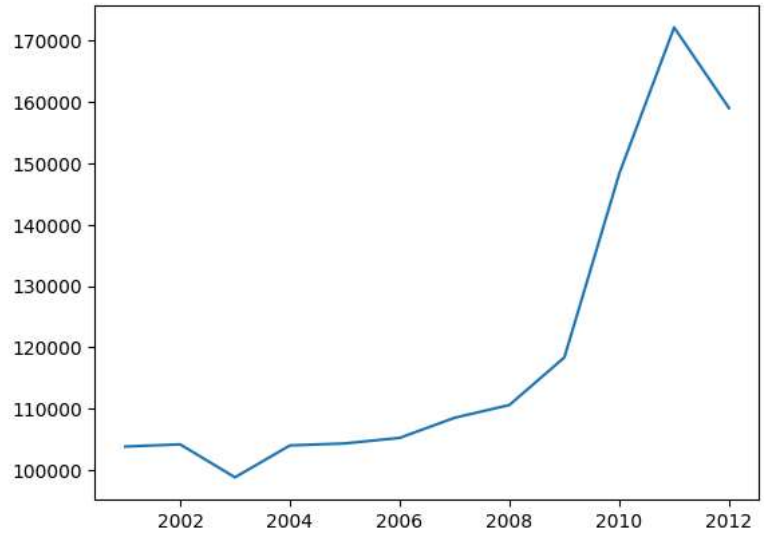
JHARKHAND :



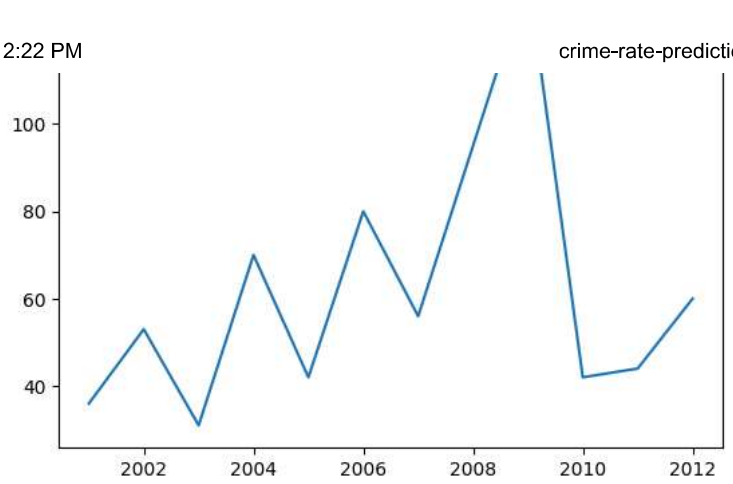
KARNATAKA :



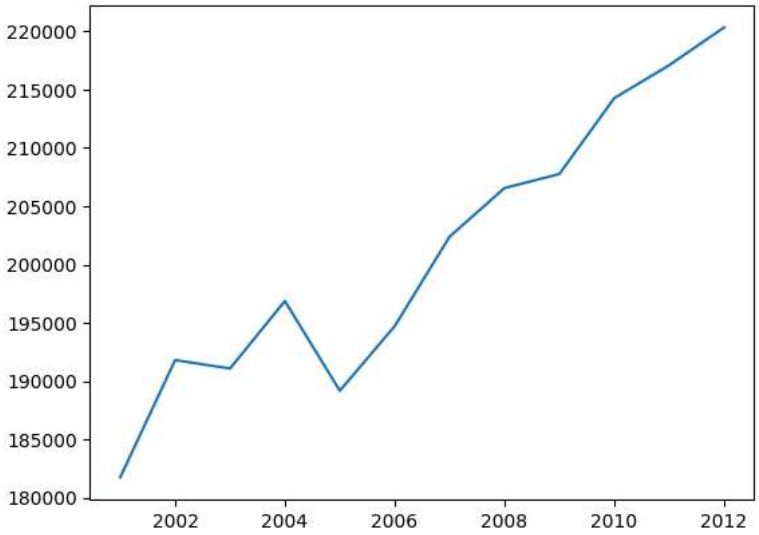
KERALA :



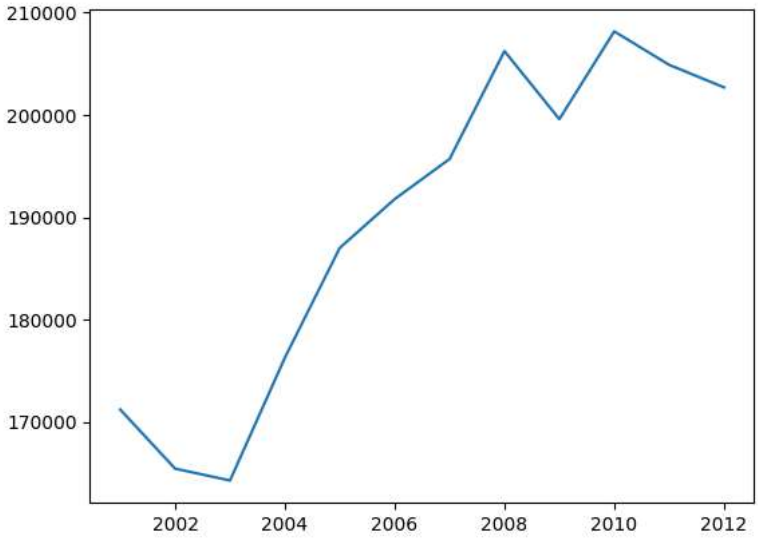
LAKSHADWEEP :



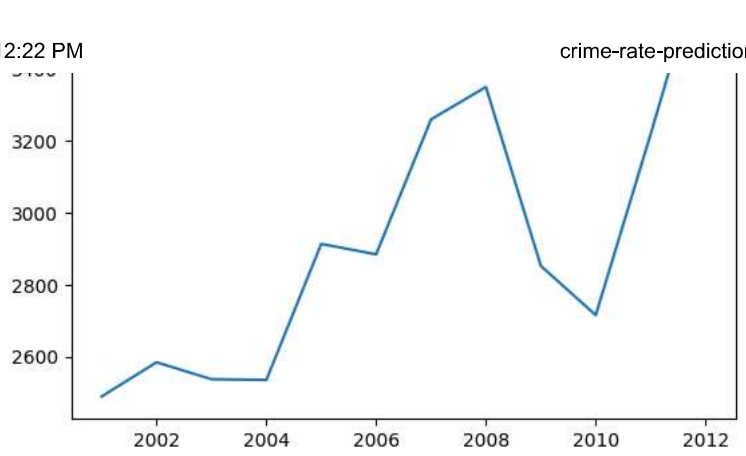
MADHYA PRADESH :



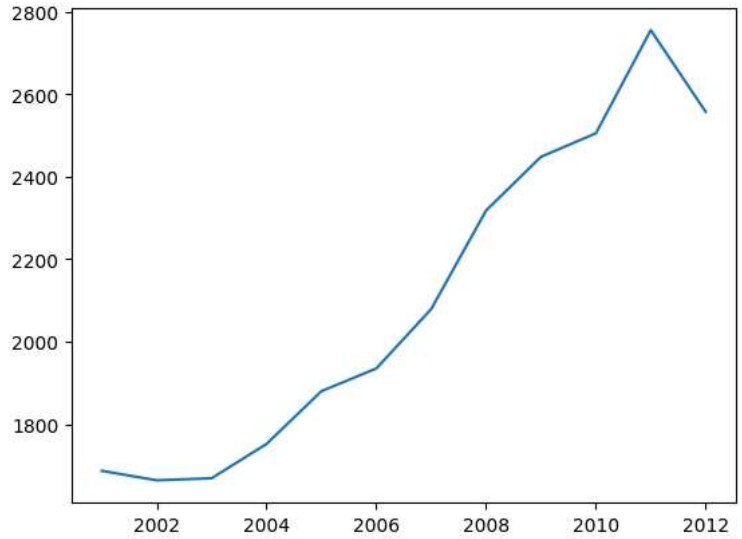
MAHARASHTRA :



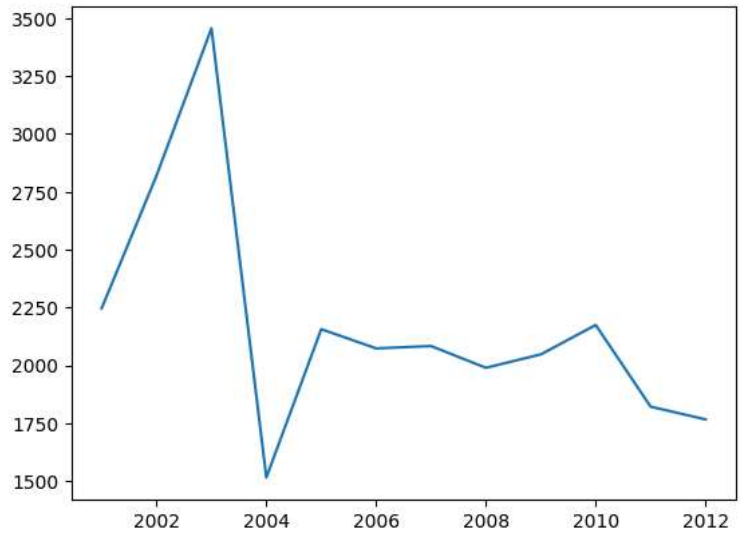
MANIPUR :



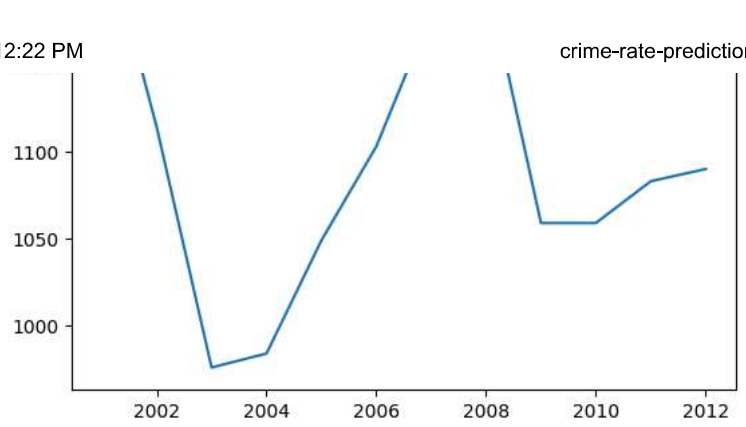
MEGHALAYA :



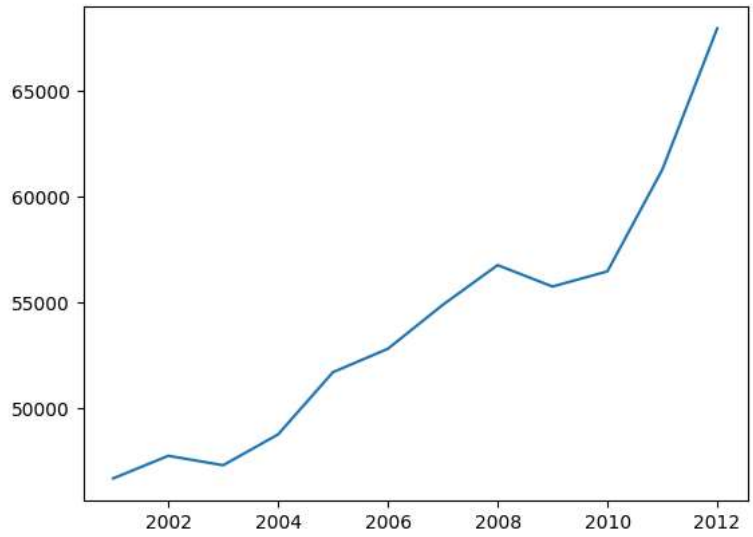
MIZORAM :



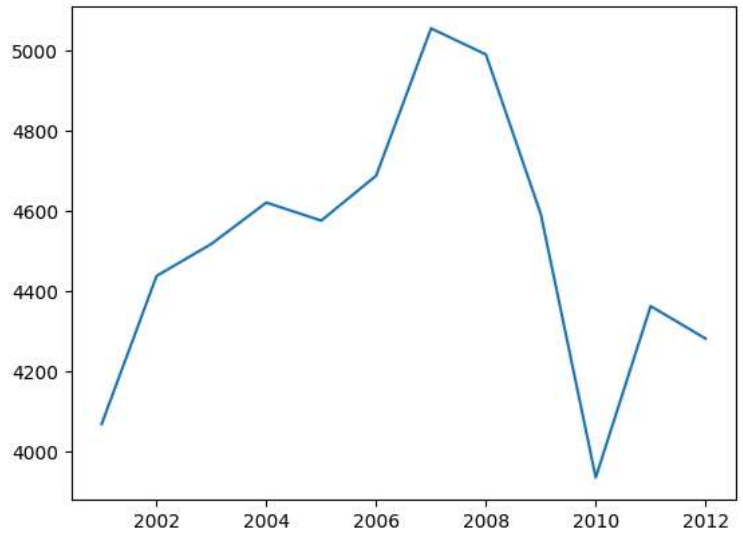
NAGALAND :



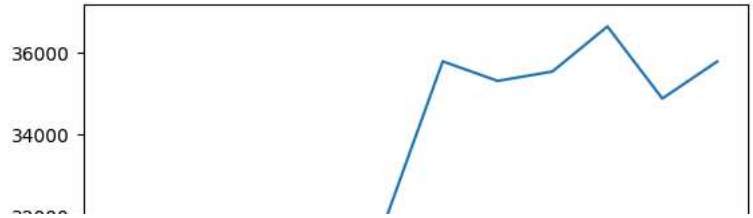
ODISHA :

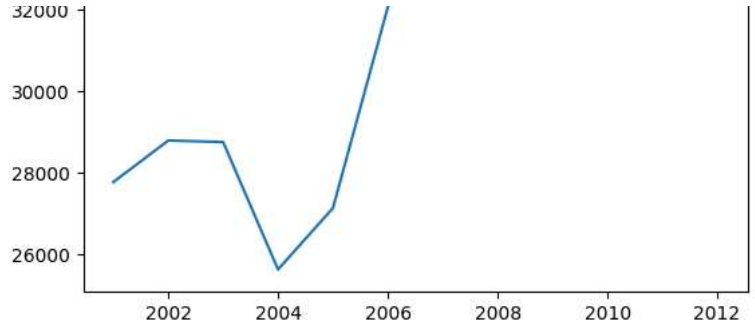


PUDUCHERRY :

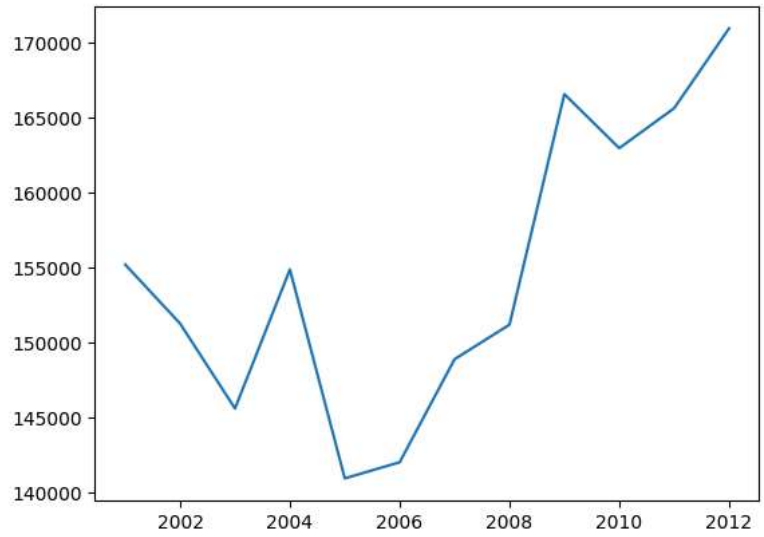


PUNJAB :

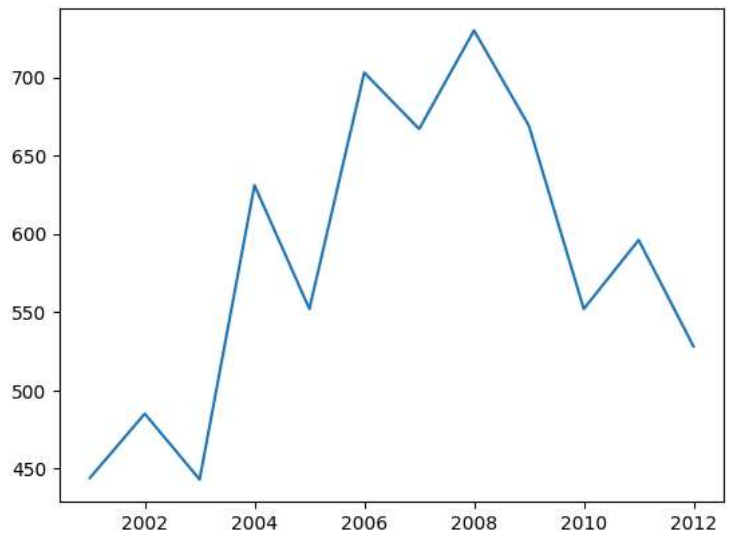




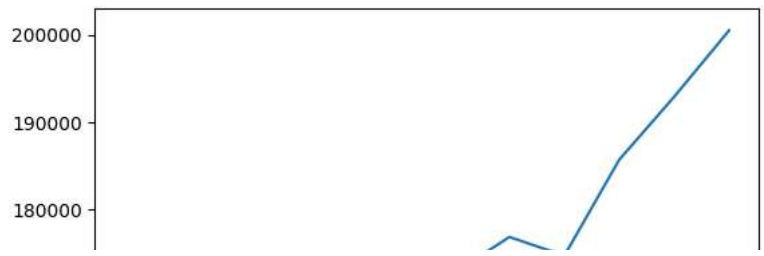
RAJASTHAN :

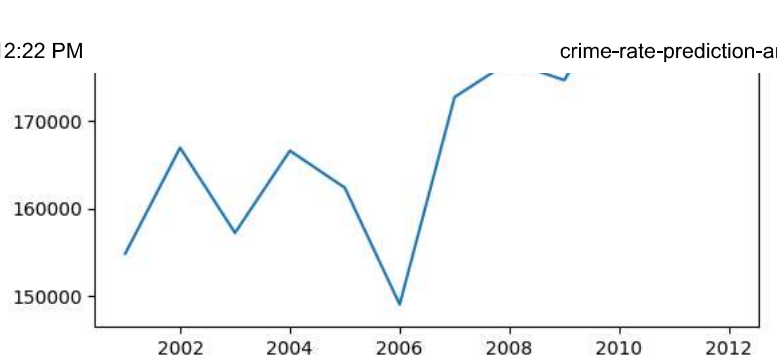


SIKKIM :

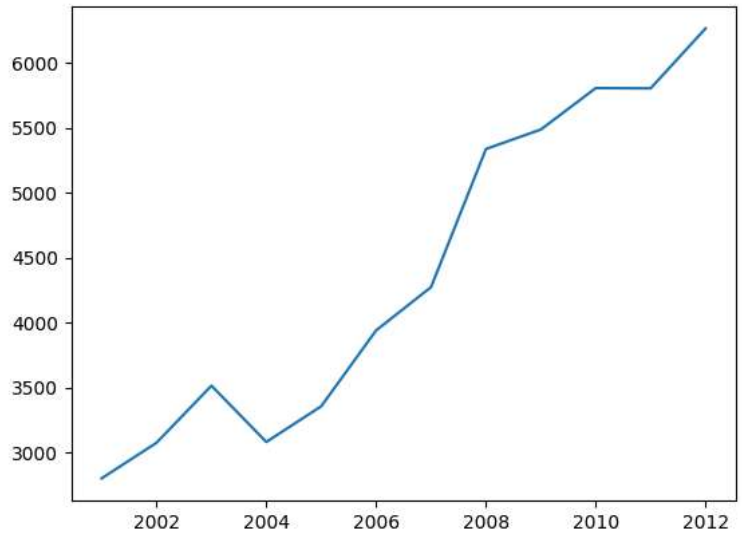


TAMIL NADU :




170000
160000
150000
2002
2004
2006
2008
2010
2012

TRIPURA :


6000
5500
5000
4500
4000
3500
3000
2002
2004
2006
2008
2010
2012