VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

COURSE NAME : BIG DATA ANALYTICS

COURSE CODE : SWE2011

FACULTY : Dr. SALEENA B

A PROJECT REPORT ON:

CRIME DATA ANALYSIS

TEAM MEMBERS:

1. YERRU VENKAT AKHIL - 18MIS1066
2. THARUNRAJ S B - 18MIS1044
3. SAI KALYAN B - 18MIS1113

CONTENTS

DECLARATION

I hereby declare that the project entitled Crime Data Analysis submitted by us to the School of Computer Science and Engineering, Vellore Institute of Technology - Chennai Campus, 600127 in partial fulfilment of the requirements of the award of the course of SWE2011 - Big Data Analytics is a Bonafide record of the work carried out by me under the supervision of Guide. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or any other institute or University.

Place: Chennai

Signature of Candidates

Date: 12-12-2021

Yerru Venkat Akhil - 18MIS1066

Sai Kalyan B - 18MIS1113

Tharunraj S B - 18MIS1044

CERTIFICATE

This is to certify that the report entitled Crime Data Analysis is prepared and submitted by Tharunraj S B -18MIS1044, Yerru Venkat Akhil - 18MIS1066, Sai Kalyan B - 18MIS1113 to Vellore Institute of Technology - Chennai Campus, in partial fulfilment of the requirement for the award of the course of SWE2011 - Big Data Analytics is a Bonafide record carried out under my guidance. The project fulfils the requirements as per the regulations of this University and in my opinion, meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Guide/Supervisor	Head of the Department
Name: Dr. Saleena B	Name: Dr. Asnath Phamila Y
Date: 12-12-2021	Date: 12-12-2021

ACKNOWLEDGEMENT

We are profoundly grateful to Dr. Saleena B for his expert guidance and continuous encouragement throughout to see that this project rights its target from its commencement to its completion. We would like to express our deepest appreciation towards Vellore Institute of Technology Chennai and Dr. Asnath Phamila Y, Head of the Department of Software Engineering whose invaluable guidance supported us in completing this project. At last, we must express our sincere heartfelt gratitude to our friends and seniors who helped me directly or indirectly during this course of work.

YERRU VENKAT AKHIL - 18MIS1066

THARUNRAJ S B - 18MIS1044

SAI KALYAN B - 18MIS1113

ABSTRACT:

As the year progresses in India, the number of crime factors increases, and the number of crimes per district rises exponentially. It is quite tough to analyse these crime statistics in India's various states year by year. We used Big Data Tools and Technologies like Hadoop and Hive Queries to analyse these criminal factors. The process is made easier by using Hadoop's Map Reduce Query. We can simply extract data for any crime factor in any district using Hive Queries. The dataset we picked comprises detailed information on many aspects of crimes that occurred in India between 2001 to 2012. In this project, we'll look at some of the different aspects of crime and depict them.

1. INTRODUCTION:

The crime datasets were gathered using Kaggle. Users can use it (Kaggle) to locate and publish datasets, study and develop models using various data mining and machine learning algorithms, and do big data analytics. In this project, we will train the algorithms and dataset in order to execute the test for crime analysis, and we will compare the various sets of algorithms that we have created. After that, we'll run Map Reduce and Hive queries to improve our analysis. Then the many parameters of crime, such as murder, robbery, theft, auto theft, burglary, dowry deaths, and rape, will be visualized and the project's future improvements will be incorporated in this final report.

2. DATASET USED:

Crime in India

State-wise data from 2001 is classified according to 40+factors.
https://www.kaggle.com/rajanand/crime-in-india

3. TOOLS AND TECHNOLOGIES USED:

Framework used:

Apache Hadoop

Database used:

Apache Hive

Visualization Tool:

TABLEAU

For Algorithms:

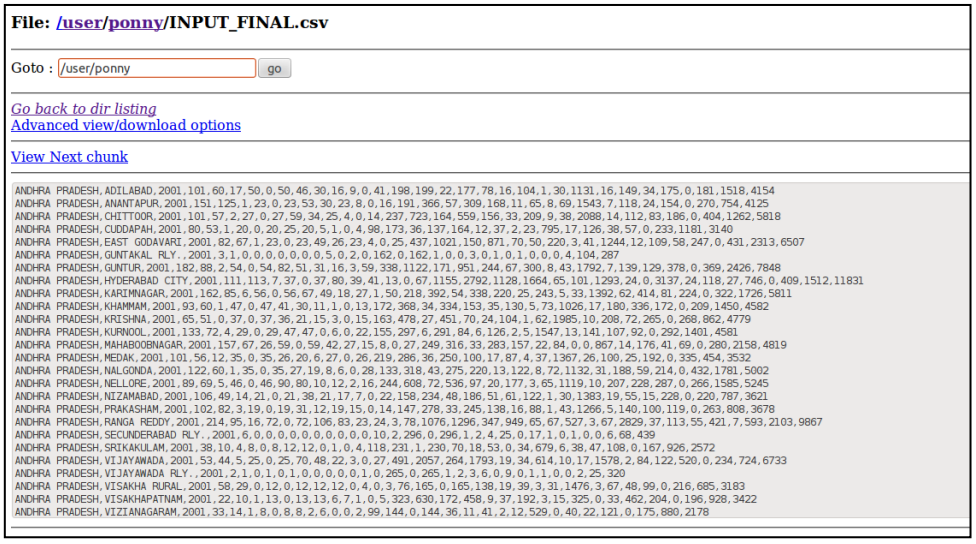
Google Colaboratory

4. PROPOSED WORK:

4.1 Analysis using Hadoop:

In this part we are going to see the analysis of the CRIME data set. Initially to perform analysis, we need load our dataset into HDFS to work in the Hadoop by using Map Reduce technique. We have performed three Map Reduce Queries.

1) First we loaded the dataset into HDFS named 'INPUT FINAL.csv'. Then we ran our first Map reduce query to find the overall number of murders that happened between 2001 and 2012, listed in state wise. The output file after executing the query is map_reduce_total_output.csv



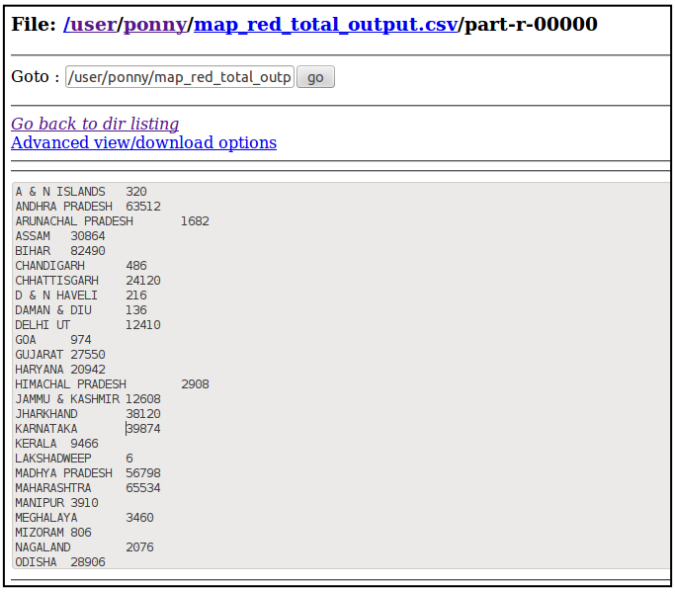
File: /user/ponny/INPUT_FINAL.csv

Goto : /user/ponny go

Go back to dir listing
Advanced view/download options

View Next chunk

```
ANDHRA PRADESH,ADILABAD,2001,101,60,17,50,0,50,46,30,16,9,0,41,198,199,22,177,78,16,104,1,30,1131,16,149,34,175,0,181,1518,4154
ANDHRA PRADESH,ANANTAPUR,2001,151,125,1,23,0,23,53,30,23,8,0,16,191,366,57,309,168,11,65,8,69,1543,7,118,24,154,0,270,754,4125
ANDHRA PRADESH,CHITTOOR,2001,101,57,2,27,0,27,59,34,25,4,0,14,237,723,164,559,156,33,209,9,38,2088,14,112,83,186,0,404,1262,5818
ANDHRA PRADESH,CUDDAPAH,2001,80,53,1,20,0,20,25,20,5,1,0,4,98,173,36,137,164,12,37,2,23,795,17,126,38,57,0,233,1181,3140
ANDHRA PRADESH,EAST GODAVARI,2001,82,67,1,23,0,23,49,26,23,4,0,25,437,1021,150,871,70,50,220,3,41,1244,12,109,58,247,0,431,2313,6507
ANDHRA PRADESH,GUNTAKAL RLY.,2001,3,1,0,0,0,0,0,0,5,0,2,0,162,0,162,1,0,0,3,0,1,0,1,0,0,0,4,104,287
ANDHRA PRADESH,GUNTUR,2001,182,88,2,54,0,54,82,51,31,16,3,59,338,1122,171,951,244,67,300,8,43,1792,7,139,129,378,0,369,2426,7848
ANDHRA PRADESH,HYDERABAD CITY,2001,111,113,7,37,0,37,80,39,41,13,0,67,1155,2792,1128,1664,65,101,1293,24,0,3137,24,118,27,746,0,409,1512,11831
ANDHRA PRADESH,KARIMNAGAR,2001,162,85,6,56,0,56,67,49,18,27,1,50,218,392,54,338,220,25,243,5,33,1392,62,414,81,224,0,322,1726,5811
ANDHRA PRADESH,KHAMMAM,2001,93,60,1,47,0,47,41,30,11,1,0,13,172,368,34,334,153,35,130,5,73,1026,17,180,336,172,0,209,1450,4582
ANDHRA PRADESH,KRISHNA,2001,65,51,0,37,0,37,36,21,15,3,0,15,163,478,27,451,70,24,104,1,62,1985,10,208,72,265,0,268,862,4779
ANDHRA PRADESH,KURNOOL,2001,133,72,4,29,0,29,47,47,0,6,0,22,155,297,6,291,84,6,126,2,5,1547,13,141,107,92,0,292,1401,4581
ANDHRA PRADESH,MAHABOOBNAGAR,2001,157,67,26,59,0,59,42,27,15,8,0,27,249,316,33,283,157,22,84,0,0,867,14,176,41,69,0,280,2158,4819
ANDHRA PRADESH,MEDAK,2001,101,56,12,35,0,35,26,20,6,27,0,26,219,286,36,250,100,17,87,4,37,1367,26,100,25,192,0,335,454,3532
ANDHRA PRADESH,NALGONDA,2001,122,60,1,35,0,35,27,19,8,6,0,28,133,318,43,275,220,13,122,8,72,1132,31,188,59,214,0,432,1781,5002
ANDHRA PRADESH,NELLORE,2001,89,69,5,46,0,46,90,80,10,12,2,16,244,608,72,536,97,20,177,3,65,1119,10,207,228,287,0,266,1585,5245
ANDHRA PRADESH,NIZAMABAD,2001,106,49,14,21,0,21,38,21,17,7,0,22,158,234,48,186,51,61,122,1,30,1383,19,55,15,228,0,220,787,3621
ANDHRA PRADESH,PRAKASHAM,2001,102,82,3,19,0,19,31,12,19,15,0,14,147,278,33,245,138,16,88,1,43,1266,5,140,100,119,0,263,808,3678
ANDHRA PRADESH,RANGA REDDY,2001,214,95,16,72,0,72,106,83,23,24,3,78,1076,1296,347,949,65,67,527,3,67,2829,37,113,55,421,7,593,2103,9867
ANDHRA PRADESH,SECUNDERABAD RLY.,2001,6,0,0,0,0,0,0,0,0,0,0,0,10,2,296,0,296,1,2,4,25,0,17,1,0,1,0,0,6,68,439
ANDHRA PRADESH,SRIKAKULAM,2001,38,10,4,8,0,8,12,12,0,1,0,4,118,231,1,230,70,18,53,0,34,679,6,38,47,108,0,167,926,2572
ANDHRA PRADESH,VIJAYAWADA,2001,53,44,5,25,0,25,70,48,22,3,0,27,491,2057,264,1793,19,34,614,10,17,1578,2,84,122,520,0,234,724,6733
ANDHRA PRADESH,VIJAYAWADA RLY.,2001,2,1,0,1,0,1,0,0,0,0,0,1,0,265,0,265,1,2,3,6,0,9,0,1,1,0,0,2,25,320
ANDHRA PRADESH,VISAKHA RURAL,2001,58,29,0,12,0,12,12,12,0,4,0,3,76,165,0,165,138,19,39,3,31,1476,3,67,48,99,0,216,685,3183
ANDHRA PRADESH,VISAKHAPATNAM,2001,22,10,1,13,0,13,13,6,7,1,0,5,323,630,172,458,9,37,192,3,15,325,0,33,462,204,0,196,928,3422
ANDHRA PRADESH,VIZIANAGARAM,2001,33,14,1,8,0,8,8,2,6,0,0,2,99,144,0,144,36,11,41,2,12,529,0,40,22,121,0,175,880,2178
```



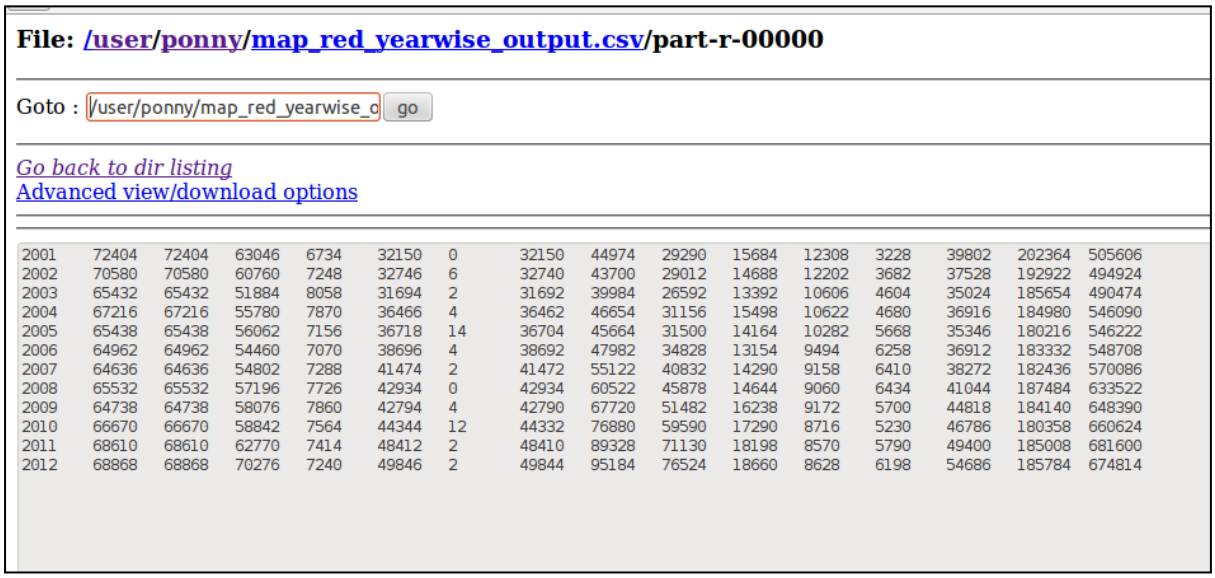
File: /user/ponny/map_red_total_output.csv/part-r-00000

Goto : /user/ponny/map_red_total_outp go

Go back to dir listing
Advanced view/download options

```
A & N ISLANDS   320
ANDHRA PRADESH  63512
ARUNACHAL PRADESH       1682
ASSAM   30864
BIHAR   82490
CHANDIGARH      486
CHHATTISGARH    24120
D & N HAVELI    216
DAMAN & DIU     136
DELHI UT        12410
GOA     974
GUJARAT 27550
HARYANA 20942
HIMACHAL PRADESH        2908
JAMMU & KASHMIR 12608
JHARKHAND       38120
KARNATAKA       39874
KERALA  9466
LAKSHADWEEP     6
MADHYA PRADESH  56798
MAHARASHTRA     65534
MANIPUR 3910
MEGHALAYA       3460
MIZORAM 806
NAGALAND        2076
ODISHA  28906
```

2) We ran our Second Map reduce query to find the count of different crime factors (such as murder, rape, auto theft, burglary etc) that happened between 2001 and 2012, listed in year wise. The output file after executing the query is map_reduce_yearwise_ output.csv



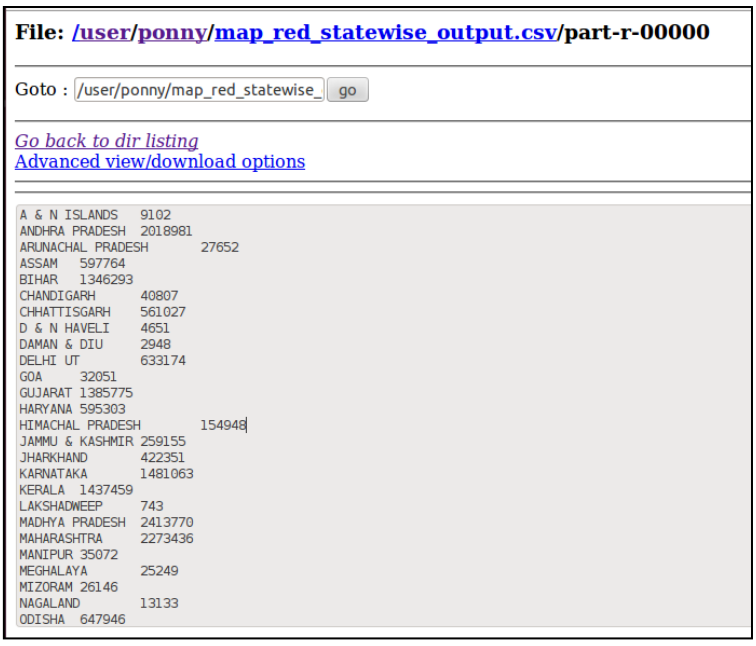
File: /user/ponny/map_red_yearwise_output.csv/part-r-00000

Goto : /user/ponny/map_red_yearwise_o go

Go back to dir listing
Advanced view/download options

2001	72404	72404	63046	6734	32150	0	32150	44974	29290	15684	12308	3228	39802	202364	505606
2002	70580	70580	60760	7248	32746	6	32740	43700	29012	14688	12202	3682	37528	192922	494924
2003	65432	65432	51884	8058	31694	2	31692	39984	26592	13392	10606	4604	35024	185654	490474
2004	67216	67216	55780	7870	36466	4	36462	46654	31156	15498	10622	4680	36916	184980	546090
2005	65438	65438	56062	7156	36718	14	36704	45664	31500	14164	10282	5668	35346	180216	546222
2006	64962	64962	54460	7070	38696	4	38692	47982	34828	13154	9494	6258	36912	183332	548708
2007	64636	64636	54802	7288	41474	2	41472	55122	40832	14290	9158	6410	38272	182436	570086
2008	65532	65532	57196	7726	42934	0	42934	60522	45878	14644	9060	6434	41044	187484	633522
2009	64738	64738	58076	7860	42794	4	42790	67720	51482	16238	9172	5700	44818	184140	648390
2010	66670	66670	58842	7564	44344	12	44332	76880	59590	17290	8716	5230	46786	180358	660624
2011	68610	68610	62770	7414	48412	2	48410	89328	71130	18198	8570	5790	49400	185008	681600
2012	68868	68868	70276	7240	49846	2	49844	95184	76524	18660	8628	6198	54686	185784	674814

3) We ran our Third Map reduce query to find the total crimes that happened between 2001 and 2012, listed in state wise. The output file after executing the query is map_reduce_statewise _output.csv



File: /user/ponny/map_red_statewise_output.csv/part-r-00000

Goto : /user/ponny/map_red_statewise_ go

Go back to dir listing
Advanced view/download options

```
A & N ISLANDS   9102
ANDHRA PRADESH  2018981
ARUNACHAL PRADESH       27652
ASSAM   597764
BIHAR   1346293
CHANDIGARH      40807
CHHATTISGARH    561027
D & N HAVELI    4651
DAMAN & DIU     2948
DELHI UT        633174
GOA     32051
GUJARAT 1385775
HARYANA 595303
HIMACHAL PRADESH        154948
JAMMU & KASHMIR 259155
JHARKHAND       422351
KARNATAKA       1481063
KERALA  1437459
LAKSHADWEEP     743
MADHYA PRADESH  2413770
MAHARASHTRA     2273436
MANIPUR 35072
MEGHALAYA       25249
MIZORAM 26146
NAGALAND        13133
ODISHA  647946
```

(In the appendix, you'll find the Map Reduce codes.)

4.2 Analysis using Hive:

1. Creating Table:

```
create table crime2(state string,dist string,year int,murder int,attempt int,culp int,rape int,custodial_rape int,o_rape int,kid_adb int,kid_adb_w int,kid_adb_o int,dac int,p_a int,robbery int,burg int,theft int,a_theft int,o_theft int,riot int,c_breach int,cheat int,cf int,arson int,h_g int,dowr int,assault int,insult_m int,cruel_hub int,import_girls int,cau_death int,o_ipc int,tot_ipc int) row format delimited fields terminated by ',' stored as textfile;
```

2. Loading the dataset:

```
load data local inpath '/home/ponny/Desktop/crime_data2.csv' overwrite into table crime2;
```

3. Executing the hive queries:

```
//query 1
select state,dist from crime where rape>50 and year=2001;

//query2
select SUM(rape) from crime  where state="ANDHRA PRADESH" and year="2001";

//query3
select COUNT(dist) from crime  where state="ANDHRA PRADESH" and year="2001";
//query 4
select state,dist,murder from crime where state="TAMIL NADU" and dist="CHENNAI" and year=2005;
```

```
hive>  select state,dist,murder from crime where state="TAMIL NADU" and dist="CHENNAI" and year=2005;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_202112122235_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_202112122235_0002
Kill Command = /home/ponny/hadoop/libexec/../bin/hadoop job  -Dmapred.job.tracker=localhost:54311 -kill job_202112122235_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-12-12 23:26:30,460 Stage-1 map = 0%,  reduce = 0%
2021-12-12 23:26:36,535 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:37,555 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:38,584 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:39,594 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:40,606 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:41,644 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2021-12-12 23:26:42,671 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 0.79 sec
MapReduce Total cumulative CPU time: 790 msec
Ended Job = job_202112122235_0002
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.79 sec   HDFS Read: 962371 HDFS Write: 23 SUCCESS
Total MapReduce CPU Time Spent: 790 msec
OK
TAMIL NADU      CHENNAI 123
Time taken: 21.25 seconds
hive>
```

//query 5

```
select Sum(murder) from crime;
```

//query6

//creating partition

```
create table part_crime_state(murder int, rape int)
partitioned by (state string);

insert overwrite table part_crime_state partition (state)
select * from crime;
```

4. Executing UDF Queries:

(i) UDF query to convert the Uppercase word to lowercase word.

//UDF function

```
//lowercaseudf.java
package udf_hive;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
public class lowercaseudf extends UDF {
public Text evaluate(final Text s) {
if (s == null) { return null; }
return new Text(s.toString().toLowerCase());
}
}
```

//execute

```
add jar lowercase.jar;
create temporary function lower_letter as
'udf_hive.lowercaseudf';
```

```
select lower_letter(state) from crime;
```

(ii)UDF query to find the maximum of three crimes

```
//UDF function
      package udf_hive;
      import org.apache.hadoop.hive.ql.exec.UDF;
      import org.apache.hadoop.io.Text;
      public class maximum_udf extends UDF {
      public int evaluate(int murder, int robbery, int theft)
      {
            int maxi=robbery;
            if(murder>maxi)
            {
                  maxi= murder;
            }
            if(theft>maxi)
            {
                  maxi= theft;
            }
            return maxi;
      }
      }
```

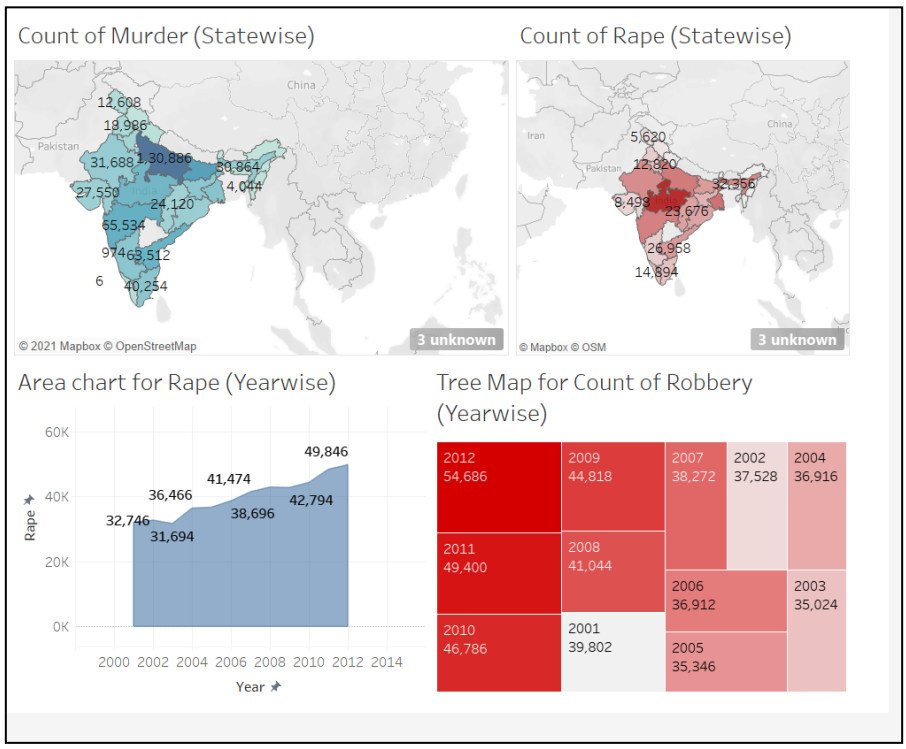
//execute

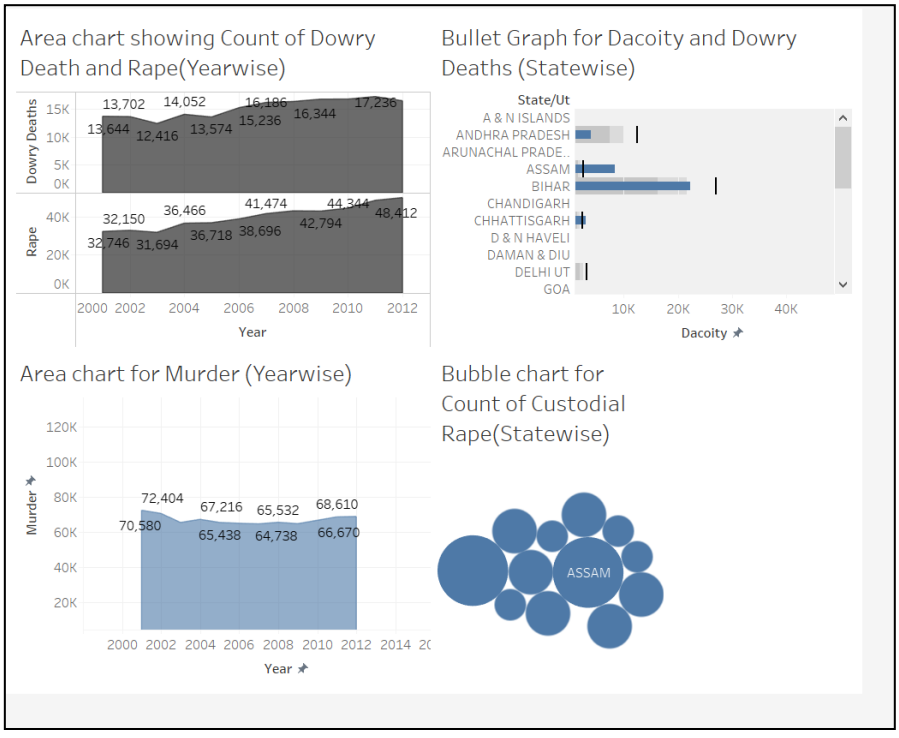
```
add jar maxim.jar;
 create temporary function highest as 'udf_hive.maximum_udf';
 select state,highest(murder,robbery,theft) from crime where
year=2010 and dist="CHENNAI";
```

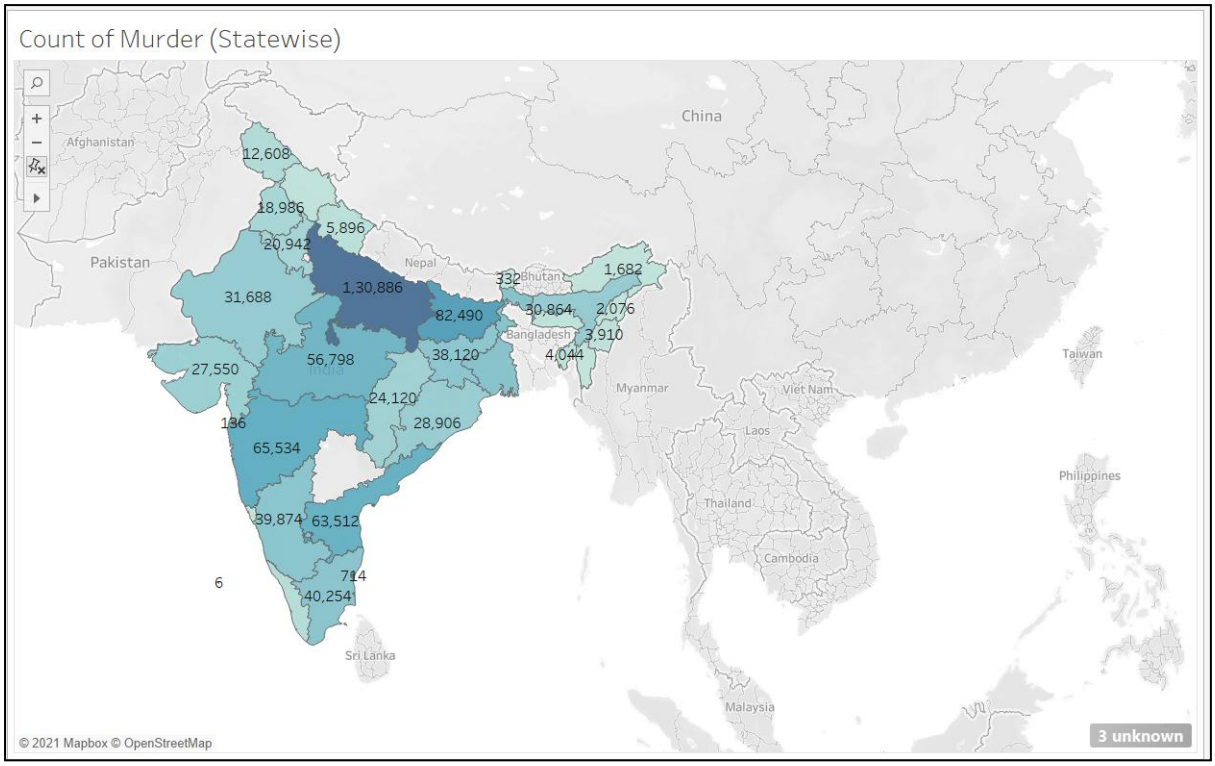
```
hive> add jar maxim.jar;
 create temporary function highest as 'udf_hive.maximum_udf';
 select state,highest(murder,robbery,theft) from crime where year=2010 and dist="CHENNAI";
Added maxim.jar to class path
Added resource: maxim.jar
OK
Time taken: 0.004 seconds
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_202112122235_0004, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_202112122235_0004
Kill Command = /home/ponny/hadoop/libexec/../bin/hadoop job  -Dmapred.job.tracker=localhost:54311 -kill job_202112122235_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2021-12-12 23:32:07,001 Stage-1 map = 0%,  reduce = 0%
2021-12-12 23:32:13,039 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:14,047 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:15,058 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:16,074 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:17,082 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:18,089 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2021-12-12 23:32:19,102 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 0.94 sec
MapReduce Total cumulative CPU time: 940 msec
Ended Job = job_202112122235_0004
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.94 sec   HDFS Read: 962371 HDFS Write: 16 SUCCESS
Total MapReduce CPU Time Spent: 940 msec
OK
TAMIL NADU      1540
Time taken: 21.744 seconds
hive>
```

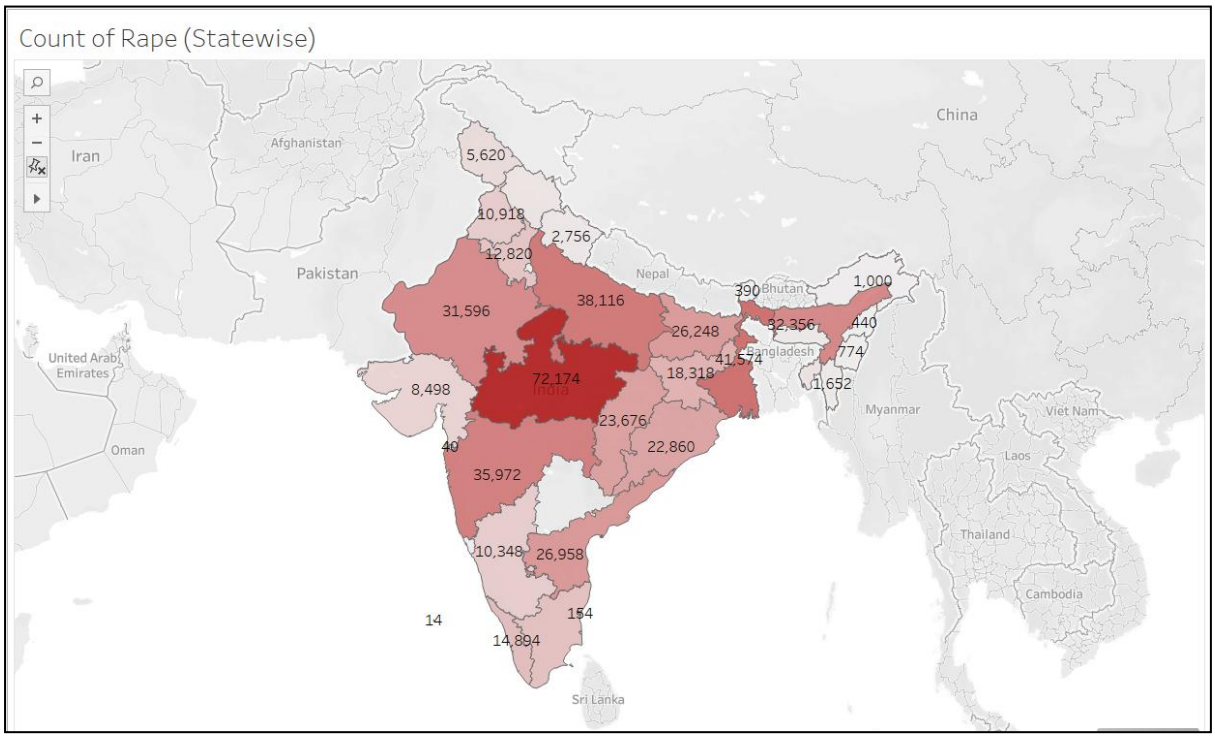
4.3 Visualisation using Tableau:

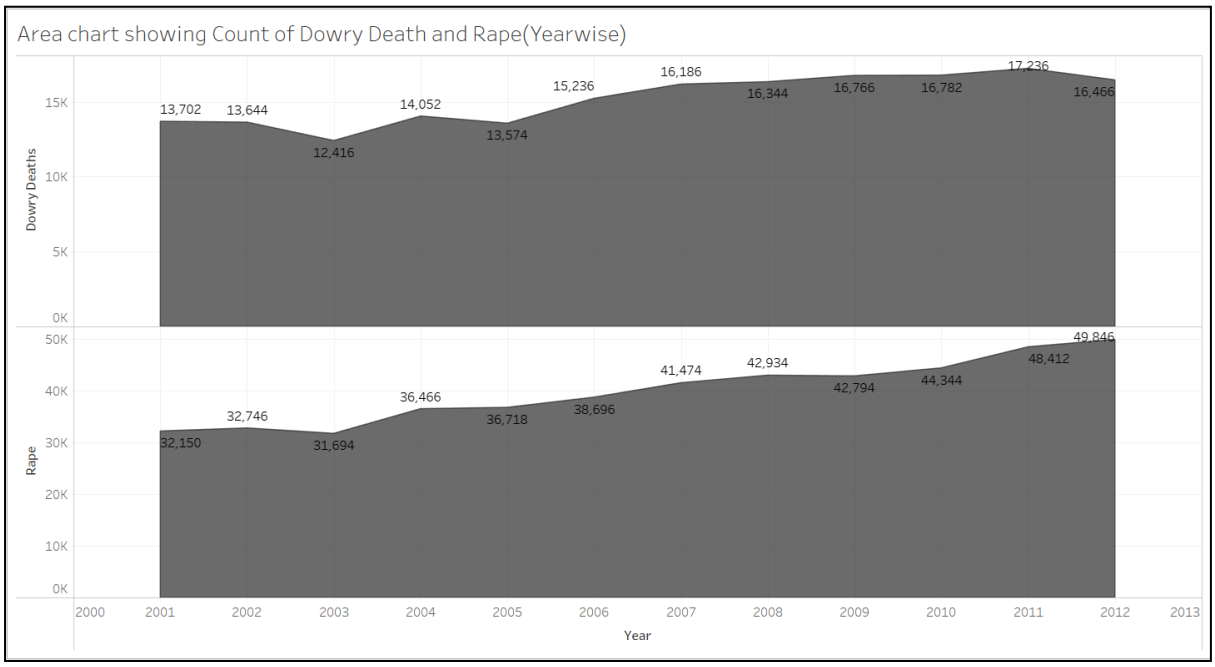
Dash Boards:

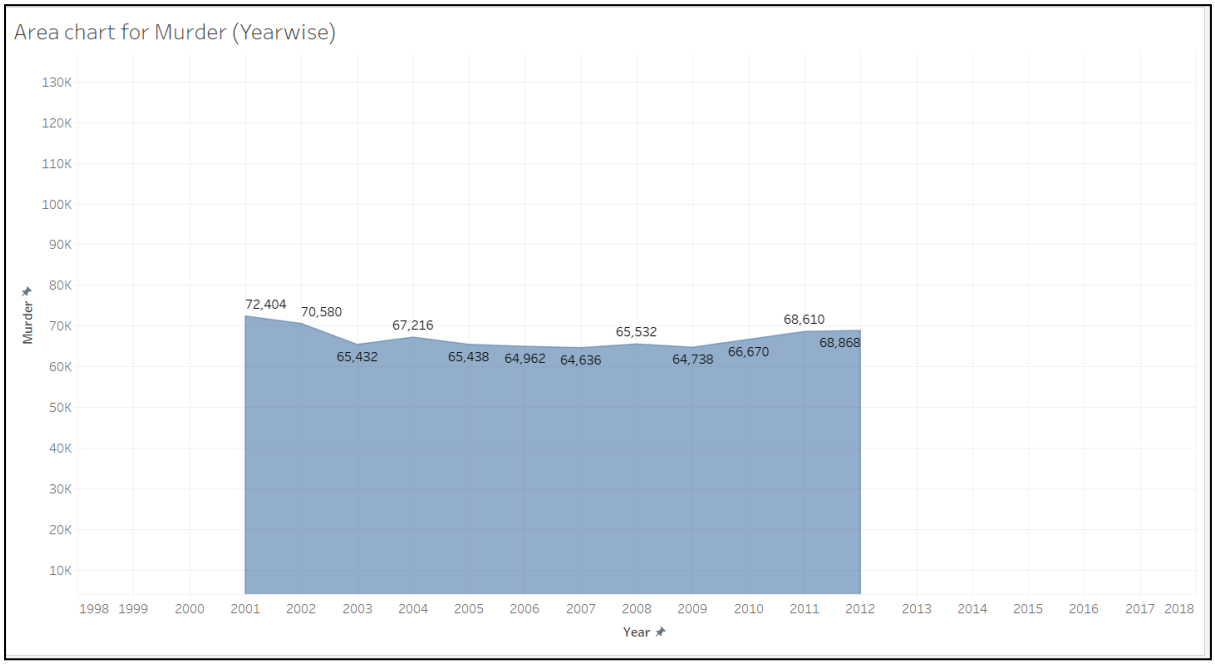


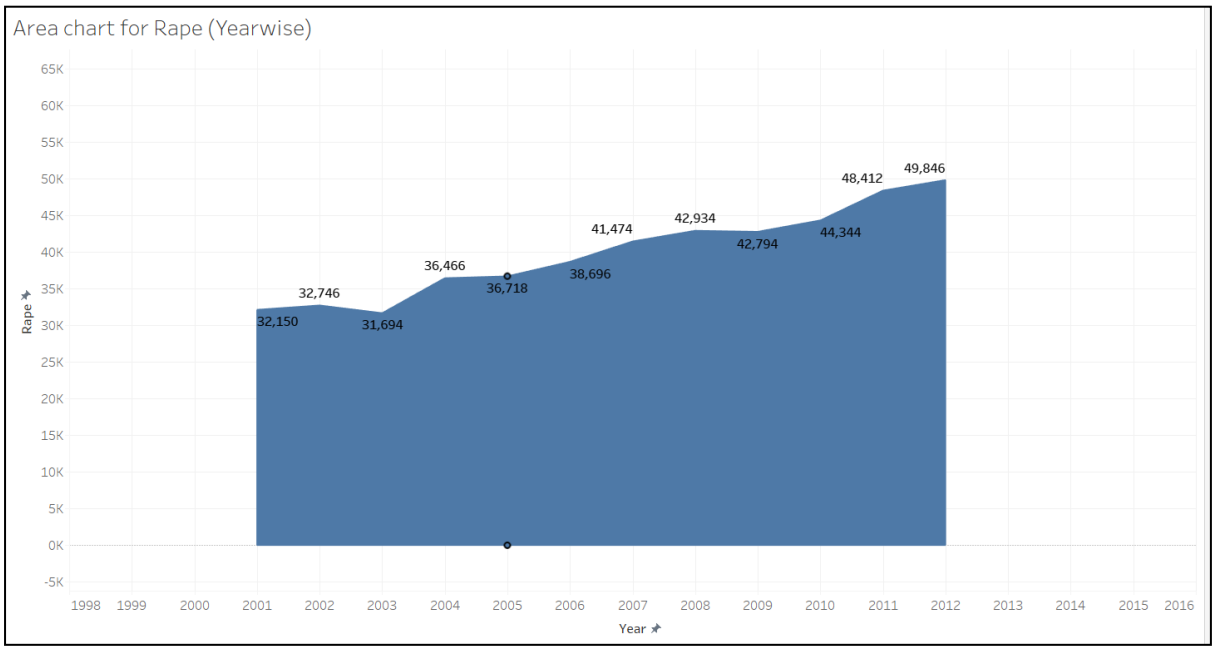


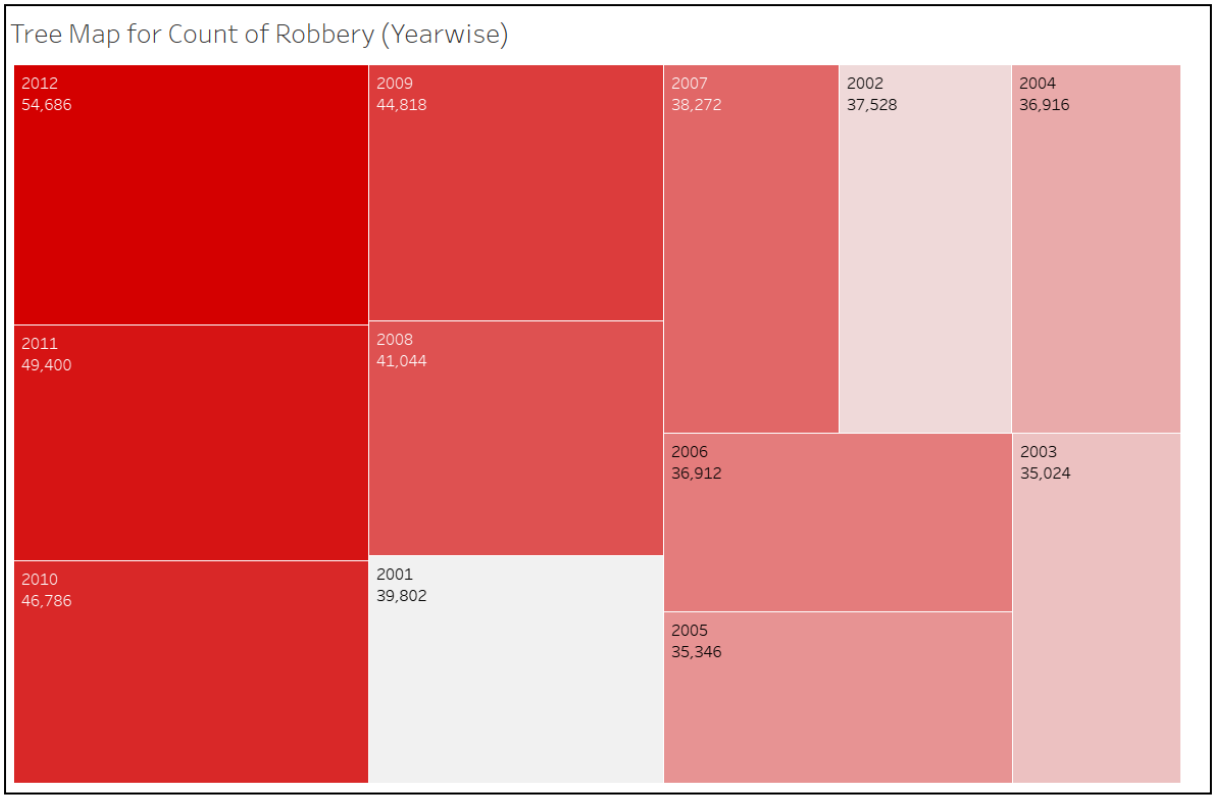

Count of Murder (Statewise)
China
Afghanistan
Pakistan
Nepal
Bhutan
Bangladesh
Myanmar
India
Taiwan
Viet Nam
Laos
Philippines
Thailand
Cambodia
Malaysia
Sri Lanka
12,608
18,986
5,896
20,942
1,30,886
332
1,682
31,688
82,490
30,864
2,076
3,910
4,044
27,550
56,798
38,120
24,120
186
28,906
65,534
39,874
63,512
714
6
40,254
© 2021 Mapbox © OpenStreetMap
3 unknown

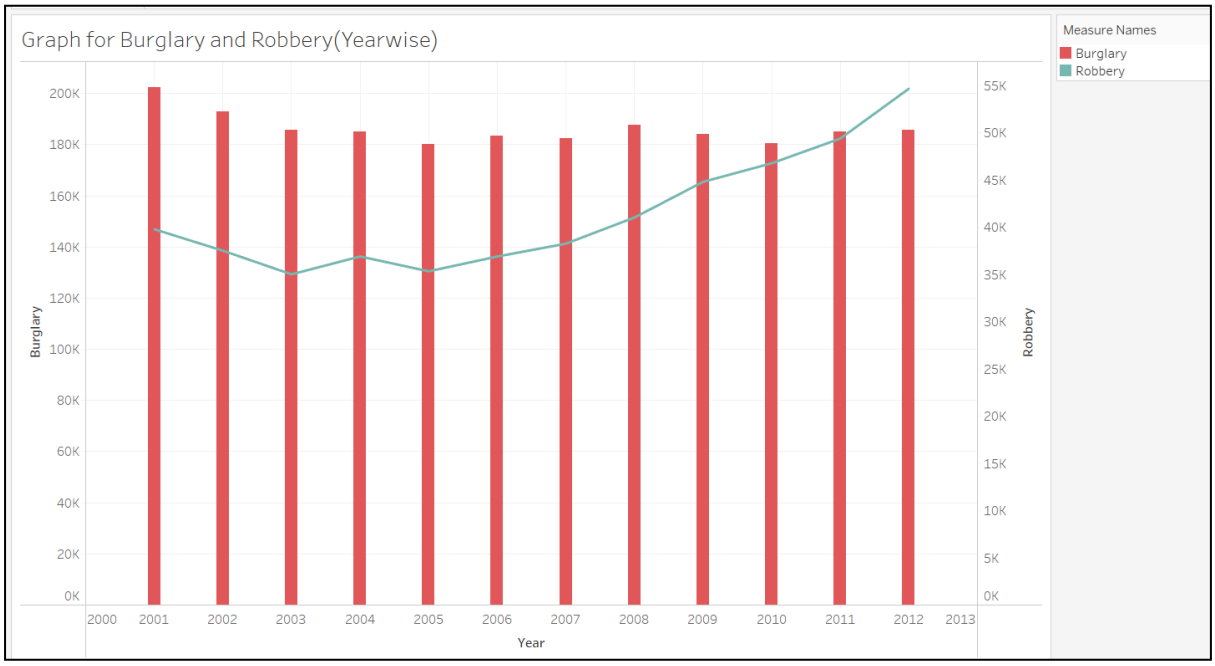

Count of Rape (Statewise)
China
Iran
Afghanistan
Pakistan
Nepal
Bhutan
Bangladesh
Myanmar
India
United Arab Emirates
Oman
Viet Nam
Laos
Thailand
Cambodia
Sri Lanka
5,620
10,918
2,756
12,820
390
1,000
38,116
31,596
26,248
32,356
440
41,574
774
18,318
72,174
8,498
1,652
23,676
40
22,860
35,972
10,348
26,958
14
154
14,894

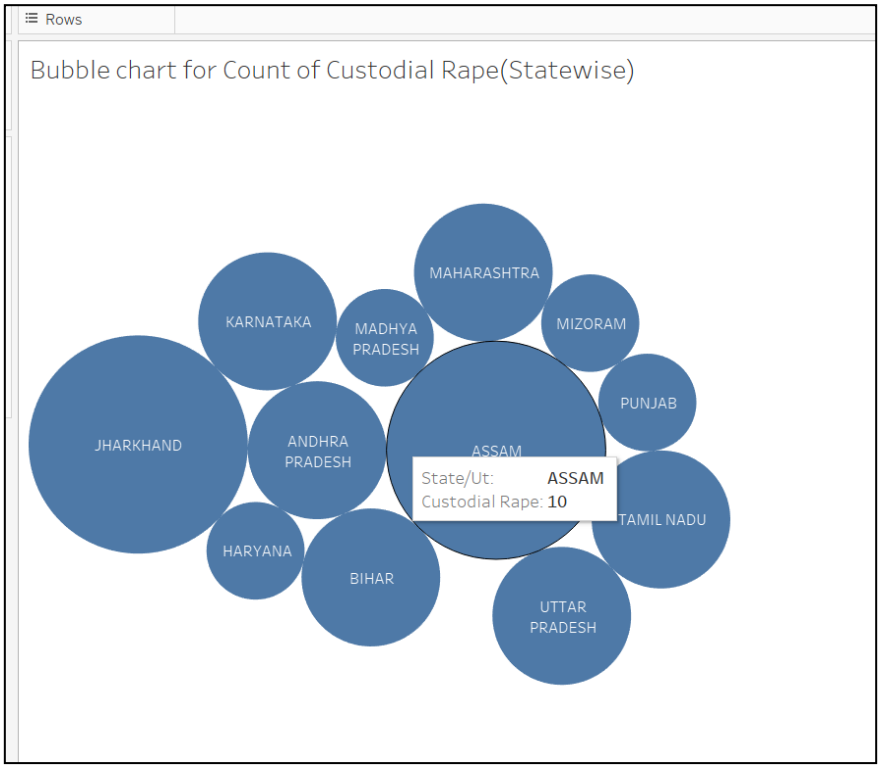

Area chart showing Count of Dowry Death and Rape(Yearwise)
Dowry Deaths
15K
10K
5K
0K
13,702
13,644
12,416
14,052
13,574
15,236
16,186
16,344
16,766
16,782
17,236
16,466
Rape
50K
40K
30K
20K
10K
0K
32,150
32,746
31,694
36,466
36,718
38,696
41,474
42,934
42,794
44,344
48,412
49,846
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
Year

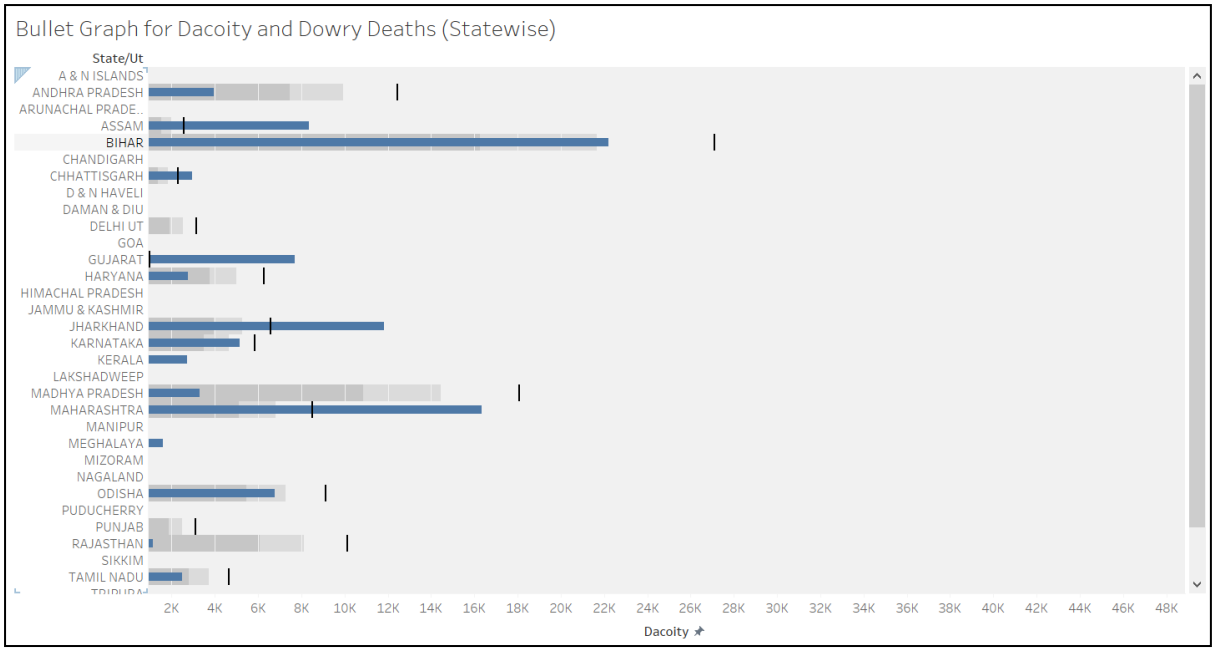

Area chart for Murder (Yearwise)
Murder
130K
120K
110K
100K
90K
80K
70K
60K
50K
40K
30K
20K
10K
72,404
70,580
65,432
67,216
65,438
64,962
64,636
65,532
64,738
66,670
68,610
68,868
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Year

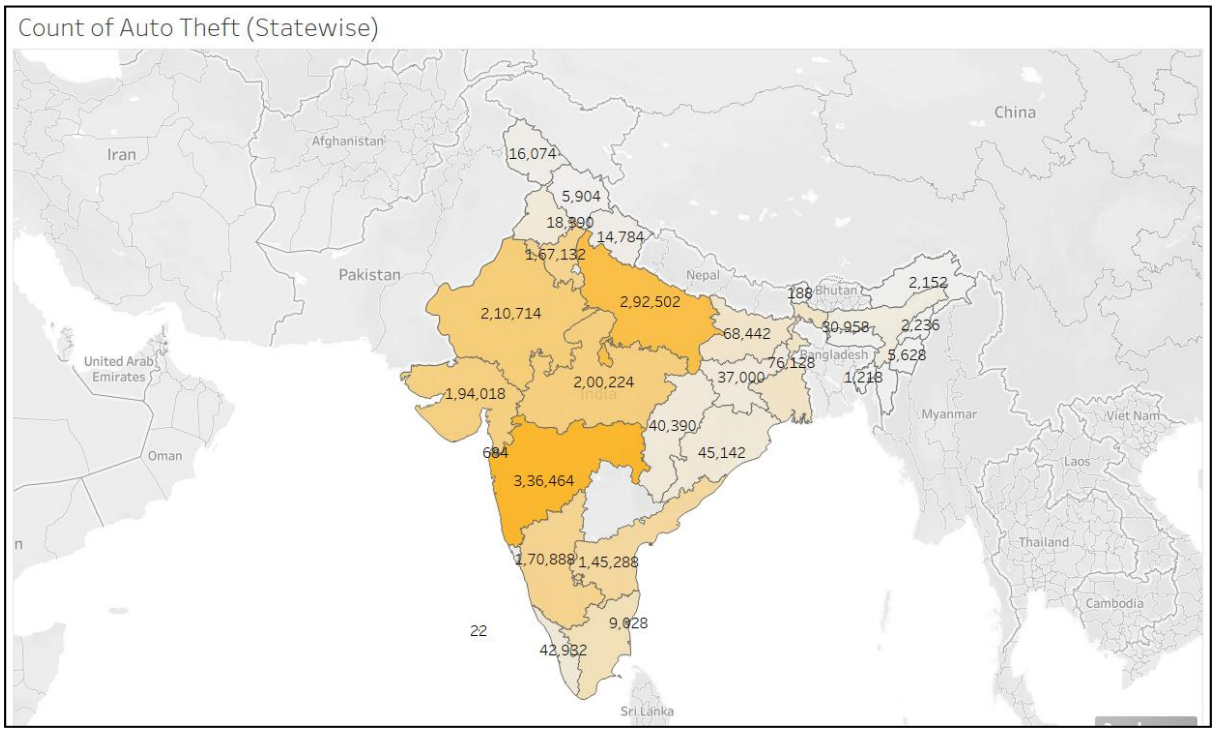

Area chart for Rape (Yearwise)
65K
60K
55K
50K
45K
40K
35K
30K
25K
20K
15K
10K
5K
0K
-5K
Rape
32,150
32,746
31,694
36,466
36,718
38,696
41,474
42,934
42,794
44,344
48,412
49,846
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
Year


Tree Map for Count of Robbery (Yearwise)
2012
54,686
2009
44,818
2007
38,272
2002
37,528
2004
36,916
2011
49,400
2008
41,044
2006
36,912
2003
35,024
2010
46,786
2001
39,802
2005
35,346


Graph for Burglary and Robbery(Yearwise)
Measure Names
Burglary
Robbery
Burglary
Robbery
Year
200K
180K
160K
140K
120K
100K
80K
60K
40K
20K
0K
55K
50K
45K
40K
35K
30K
25K
20K
15K
10K
5K
0K
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013


Rows
Bubble chart for Count of Custodial Rape(Statewise)
MAHARASHTRA
KARNATAKA
MADHYA PRADESH
MIZORAM
PUNJAB
JHARKHAND
ANDHRA PRADESH
ASSAM
State/Ut: ASSAM
Custodial Rape: 10
TAMIL NADU
HARYANA
BIHAR
UTTAR PRADESH


Bullet Graph for Dacoity and Dowry Deaths (Statewise)
State/Ut
A & N ISLANDS
ANDHRA PRADESH
ARUNACHAL PRADE..
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
2K 4K 6K 8K 10K 12K 14K 16K 18K 20K 22K 24K 26K 28K 30K 32K 34K 36K 38K 40K 42K 44K 46K 48K
Dacoity

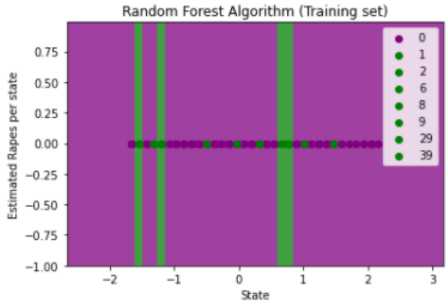

Count of Auto Theft (Statewise)
China
Iran
Afghanistan
Pakistan
Nepal
Bhutan
Bangladesh
India
Myanmar
Viet Nam
Laos
Thailand
Cambodia
United Arab Emirates
Oman
Sri Lanka
16,074
5,904
18,390
14,784
1,67,132
2,10,714
2,92,502
188
2,152
68,442
30,958
2,236
76,128
5,628
37,000
1,218
1,94,018
2,00,224
40,390
684
45,142
3,36,464
1,70,888
1,45,288
9,028
22
42,932

4.4 Implementation of Algorithms:

i) Random-Forest Algorithm:

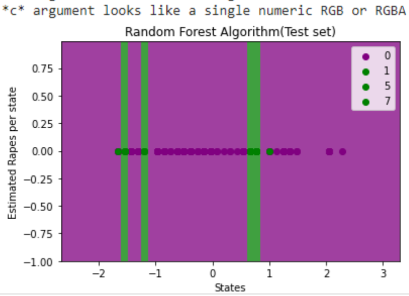
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple','green','green','green','green','green','green' ))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('State')
mtp.ylabel('Estimated Rapes per state')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will h
```



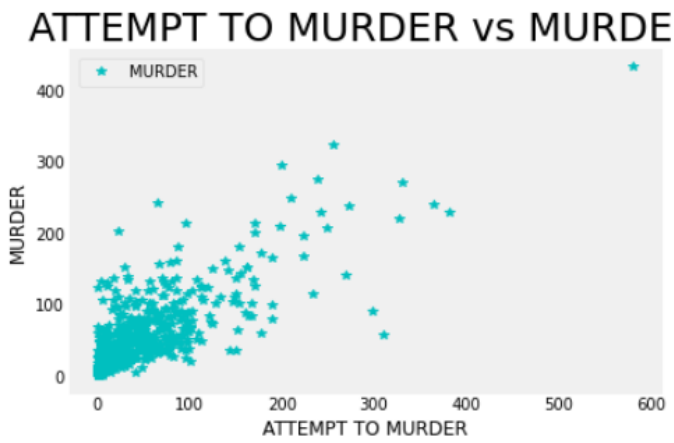

```
#Visualaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('States')
mtp.ylabel('Estimated Rapes per state')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
```

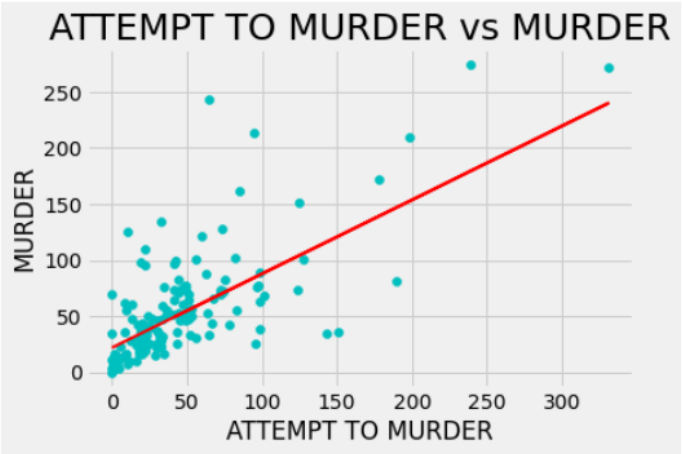


ii) Linear Regression

```
#(Linear Regression)
data2.plot(x='ATTEMPT TO MURDER',y='MURDER',style='*',color='c')
plt.title('ATTEMPT TO MURDER vs MURDER',fontdict={'family':'Sans Serif','size':25})
plt.xlabel('ATTEMPT TO MURDER')
plt.ylabel('MURDER')
plt.grid()
plt.show()
```



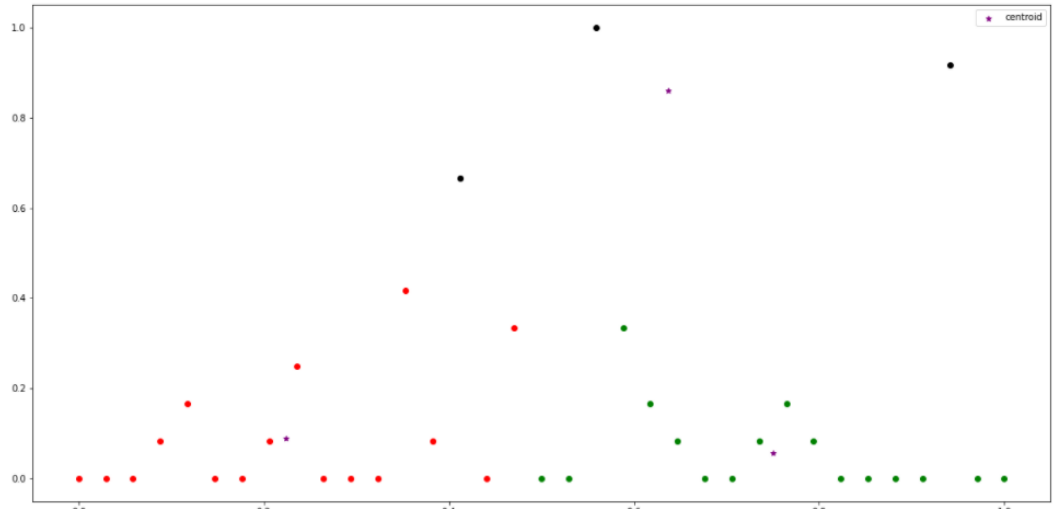

```
plt.scatter(x_test,y_test,color='c')
plt.plot(x_test,y_pred,color='r',linewidth=2)
plt.title('ATTEMPT TO MURDER vs MURDER',fontdict={'family':'Sans Serif','size':25})
plt.xlabel('ATTEMPT TO MURDER')
plt.ylabel('MURDER')
plt.show()


#y=mx+c Plotting the linear Regression Line
```



iii) K-Means:

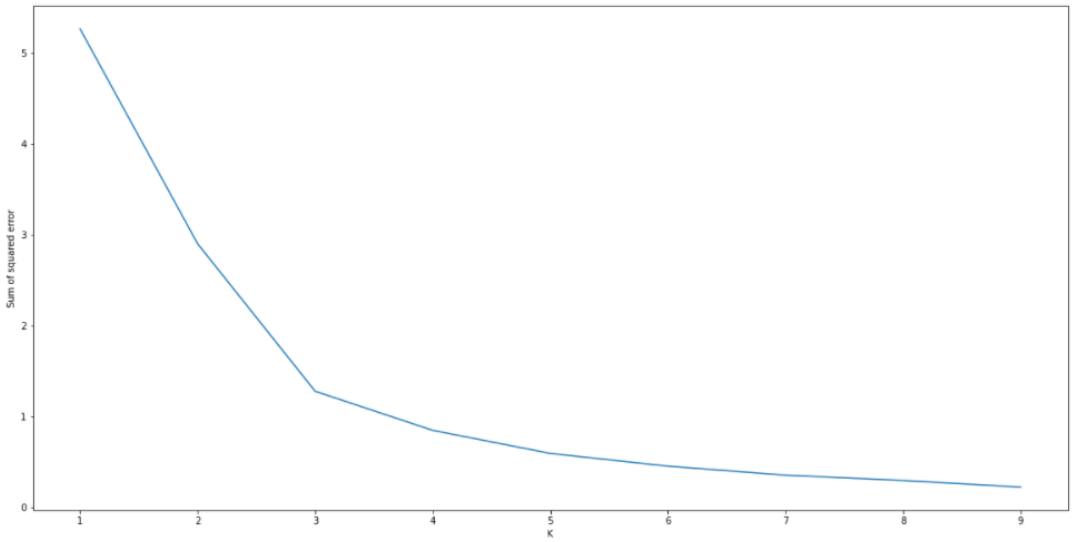
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.NumericSTATE,df1['Victims_Between_10-14_Yrs'],color='green')
plt.scatter(df2.NumericSTATE,df2['Victims_Between_10-14_Yrs'],color='red')
plt.scatter(df3.NumericSTATE,df3['Victims_Between_10-14_Yrs'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8334325310>
```




```
[ ] plt.xlabel('K')
    plt.ylabel('Sum of squared error')
    plt.plot(k_rng,sse)
```

```
[<matplotlib.lines.Line2D at 0x7f833ee26e90>]
```



iv) PCY Algorithm:

```
[23] data.take(12)

['MAHARASHTRA,UTTAR PRADESH,GUJARAT,MADHYA PRADESH,TAMIL NADU',
 'MAHARASHTRA,UTTAR PRADESH,GUJARAT,MADHYA PRADESH,RAJASTHAN',
 'MAHARASHTRA,GUJARAT,UTTAR PRADESH,RAJASTHAN,MADHYA PRADESH',
 'MAHARASHTRA,UTTAR PRADESH,GUJARAT,RAJASTHAN,MADHYA PRADESH',
 'MAHARASHTRA,UTTAR PRADESH,GUJARAT,RAJASTHAN,MADHYA PRADESH',
 'MAHARASHTRA,UTTAR PRADESH,MADHYA PRADESH,RAJASTHAN,GUJARAT',
 'MAHARASHTRA,UTTAR PRADESH,MADHYA PRADESH,RAJASTHAN,GUJARAT',
 'MAHARASHTRA,UTTAR PRADESH,MADHYA PRADESH,GUJARAT,RAJASTHAN',
 'MAHARASHTRA,UTTAR PRADESH,RAJASTHAN,GUJARAT,KARNATAKA',
 'MAHARASHTRA,UTTAR PRADESH,RAJASTHAN,MADHYA PRADESH,HARYANA',
 'UTTAR PRADESH,MAHARASHTRA,RAJASTHAN,HARYANA,MADHYA PRADESH',
 'UTTAR PRADESH,MAHARASHTRA,RAJASTHAN,HARYANA,MADHYA PRADESH']
```

```
bits_1 = bits_pairs.filter(lambda x: (x[1] ==1))
bits_1.take(15)
freq_itemset_2 = bits_1.map(lambda x : list(x[0]))
freq_itemset_2.take(15)
```

```
[['MADHYA PRADESH', 'MAHARASHTRA'],
 ['MADHYA PRADESH', 'RAJASTHAN'],
 ['MADHYA PRADESH', 'UTTAR PRADESH'],
 ['MAHARASHTRA', 'RAJASTHAN'],
 ['MAHARASHTRA', 'UTTAR PRADESH'],
 ['RAJASTHAN', 'UTTAR PRADESH']]
```

(In the appendix, you'll find the Algorithm codes.)

Conclusion and future enhancements

From the datasets we have collected from Kaggle, we have trained algorithms to perform the test for the analysis of Crime. Then the comparison of the different sets of algorithms takes place here and the Random Forest algorithm seems to be the best model. Using PCY algorithm, we found out the frequent states having high rate of Murder. Then the visualization of the various parameters like Count of Robbery, Dowry Death, Rape, Robbery, Auto Theft etc, and the map visualisation of Crime cases across India is illustrated beautifully in the forms of plots and graphs. In future we are going to gather the crime data till present year and analyse the new crime factors that have emerged and identify the states with high crime rates.

References

- **https://www.kaggle.com/rajanand/crime-in-india**
- **https://www.kaggle.com/rajanand/crime-in-india?select=20_Victims_of_rape.csv**
- **https://data-flair.training/blogs/apache-hive-partitions/**
- **https://data-flair.training/blogs/hive-udf/**

Appendix

Map Reduce Codes:

1.

```
 package map_reduce_codes;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
public class map_red_total {
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
IntWritable one = new IntWritable(1);
Text new_key=new Text();

public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
String[] line = value.toString().split(",");

context.write(new Text(line[0]),new IntWritable(Integer.parseInt(line[3])));

}
}
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
int total = 0;
for (IntWritable val : values) {
total += val.get();
}
context.write(key, new IntWritable(total));
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf, "wordcount");
job.setJarByClass(map_red_total.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}
}
```

2.

```
package map_reduce_codes;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
public class map_red_yearwise {
public static class Map extends Mapper<LongWritable, Text, Text, Text>
{
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
String[] line = value.toString().split(",");
String y="";
y+=line[3];
for (int i=3;i<17;i++){
y+=","+line[i];
}
context.write(new Text(line[2]),new Text(y));
}
}
public static class Reduce extends Reducer<Text, Text, Text, Text> {
public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
int total = 0;
int t[]=new int[15];
for (Text val : values) {
String lin[]=val.toString().split(",");
for(int j=0;j<15;j++){

      t[j]+=Integer.parseInt(lin[j]);;
}
}
String out="";
for(int i=0;i<15;i++){
out+=t[i]+""+'\t';
}
context.write(key, new Text(out));
}}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf, "wordcount");
job.setJarByClass(map_red_yearwise.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}}
```

3.

```
package map_reduce_codes;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
public class map_red_statewise {
    public static class Map extends Mapper < LongWritable, Text, Text, IntWritable
> {
        IntWritable one = new IntWritable(1);
        Text new_key = new Text();
        public void map(LongWritable key, Text value, Context context) throws
IOException,
        InterruptedException {
            String[] line = value.toString().split(",");
            context.write(new Text(line[0]), new
IntWritable(Integer.parseInt(line[32])));
    }
}
public static class Reduce extends Reducer < Text, IntWritable, Text, IntWritable
> {


    public void reduce(Text key, Iterable < IntWritable> values, Context context)
    throws IOException,
    InterruptedException {
        int total = 0;
        for (IntWritable val: values) {
            total+=val.get();
        }
context.write(key,new IntWritable(total));

    }

}



    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");
        job.setJarByClass(map_red_statewise.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

Random Forest Classifier:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('01_District_wise_crimes_committed_IPC_2001_2012 (1).csv')
data_set.head()

data2 = pd.read_csv('01_District_wise_crimes_committed_IPC_2001_2012 (1).csv')
data2.loc[data2['STATE/UT'] == 'ANDHRA PRADESH', 'Numeric STATE'] = '1'
data2.loc[data2['STATE/UT'] == 'ARUNACHAL PRADESH', 'Numeric STATE'] = '2'
data2.loc[data2['STATE/UT'] == 'ASSAM', 'Numeric STATE'] = '3'
data2.loc[data2['STATE/UT'] == 'BIHAR', 'Numeric STATE'] = '4'
data2.loc[data2['STATE/UT'] == 'CHHATTISGARH', 'Numeric STATE'] = '5'
data2.loc[data2['STATE/UT'] == 'GOA', 'Numeric STATE'] = '6'
data2.loc[data2['STATE/UT'] == 'GUJARAT', 'Numeric STATE'] = '7'
data2.loc[data2['STATE/UT'] == 'HARYANA', 'Numeric STATE'] = '8'
data2.loc[data2['STATE/UT'] == 'HIMACHAL PRADESH', 'Numeric STATE'] = '9'
data2.loc[data2['STATE/UT'] == 'JAMMU & KASHMIR', 'Numeric STATE'] = '10'
data2.loc[data2['STATE/UT'] == 'JHARKHAND', 'Numeric STATE'] = '11'
data2.loc[data2['STATE/UT'] == 'KARNATAKA', 'Numeric STATE'] = '12'
data2.loc[data2['STATE/UT'] == 'KERALA', 'Numeric STATE'] = '13'
data2.loc[data2['STATE/UT'] == 'MADHYA PRADESH', 'Numeric STATE'] = '14'
data2.loc[data2['STATE/UT'] == 'MAHARASHTRA', 'Numeric STATE'] = '15'
data2.loc[data2['STATE/UT'] == 'MANIPUR', 'Numeric STATE'] = '16'
data2.loc[data2['STATE/UT'] == 'MEGHALAYA', 'Numeric STATE'] = '17'
data2.loc[data2['STATE/UT'] == 'MIZORAM', 'Numeric STATE'] = '18'
data2.loc[data2['STATE/UT'] == 'NAGALAND', 'Numeric STATE'] = '19'
data2.loc[data2['STATE/UT'] == 'ODISHA', 'Numeric STATE'] = '20'
data2.loc[data2['STATE/UT'] == 'PUNJAB', 'Numeric STATE'] = '21'
data2.loc[data2['STATE/UT'] == 'RAJASTHAN', 'Numeric STATE'] = '22'
data2.loc[data2['STATE/UT'] == 'SIKKIM', 'Numeric STATE'] = '23'
data2.loc[data2['STATE/UT'] == 'TAMIL NADU', 'Numeric STATE'] = '24'
data2.loc[data2['STATE/UT'] == 'TRIPURA', 'Numeric STATE'] = '25'
data2.loc[data2['STATE/UT'] == 'UTTAR PRADESH', 'Numeric STATE'] = '26'
data2.loc[data2['STATE/UT'] == 'UTTARAKHAND', 'Numeric STATE'] = '27'
data2.loc[data2['STATE/UT'] == 'WEST BENGAL', 'Numeric STATE'] = '28'
data2.loc[data2['STATE/UT'] == 'A & N ISLANDS', 'Numeric STATE'] = '29'
data2.loc[data2['STATE/UT'] == 'CHANDIGARH', 'Numeric STATE'] = '30'
data2.loc[data2['STATE/UT'] == 'D & N HAVELI', 'Numeric STATE'] = '31'
```

```
data2.loc[data2['STATE/UT'] == 'DAMAN & DIU', 'Numeric STATE'] = '32'
data2.loc[data2['STATE/UT'] == 'DELHI UT', 'Numeric STATE'] = '33'
data2.loc[data2['STATE/UT'] == 'LAKSHADWEEP', 'Numeric STATE'] = '34'
data2.loc[data2['STATE/UT'] == 'PUDUCHERRY', 'Numeric STATE'] = '35'

#Convert string to integer
pd.to_numeric(data2['Numeric STATE'])
a = data2.head(10)
a


#Extracting Independent and dependent Variable
x= data2.iloc[:, [33,2]].values
y= data2.iloc[:, 29].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
print(x_train,'\n')
print(y_train)

#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)

#Predicting the test set result
y_pred= classifier.predict(x_test)
y_pred

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
```

```
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c =
ListedColormap(('purple','green','green','green','green','green','green' ))(i),
label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('State')
mtp.ylabel('Estimated Rapes per state')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('States')
mtp.ylabel('Estimated Rapes per state')
mtp.legend()
mtp.show()
```

K-Means Clustering:

```
from sklearn import preprocessing
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
# %matplotlib inline

data2 = pd.read_csv("rape2.csv")
data2.head()

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,10)
data = pd.read_csv("sample_data/rape.csv")
plt.scatter(data.Area_Name,data['Victims_Between_10-14_Yrs'])
plt.xlabel('Area_Name')
plt.ylabel('Victims_Between_10-14_Yrs')

data2 = pd.read_csv('rape2.csv')
data2.loc[data2['Area_Name'] == 'ANDHRA PRADESH', 'NumericSTATE'] = '1'
data2.loc[data2['Area_Name'] == 'ARUNACHAL PRADESH', 'NumericSTATE'] = '2'
data2.loc[data2['Area_Name'] == 'ASSAM', 'NumericSTATE'] = '3'
data2.loc[data2['Area_Name'] == 'BIHAR', 'NumericSTATE'] = '4'
data2.loc[data2['Area_Name'] == 'CHHATTISGARH', 'NumericSTATE'] = '5'
data2.loc[data2['Area_Name'] == 'GOA', 'NumericSTATE'] = '6'
data2.loc[data2['Area_Name'] == 'GUJARAT', 'NumericSTATE'] = '7'
data2.loc[data2['Area_Name'] == 'HARYANA', 'NumericSTATE'] = '8'
data2.loc[data2['Area_Name'] == 'HIMACHAL PRADESH', 'NumericSTATE'] = '9'
data2.loc[data2['Area_Name'] == 'JAMMU & KASHMIR', 'NumericSTATE'] = '10'
data2.loc[data2['Area_Name'] == 'JHARKHAND', 'NumericSTATE'] = '11'
data2.loc[data2['Area_Name'] == 'KARNATAKA', 'NumericSTATE'] = '12'
data2.loc[data2['Area_Name'] == 'KERALA', 'NumericSTATE'] = '13'
data2.loc[data2['Area_Name'] == 'MADHYA PRADESH', 'NumericSTATE'] = '14'
data2.loc[data2['Area_Name'] == 'MAHARASHTRA', 'NumericSTATE'] = '15'
data2.loc[data2['Area_Name'] == 'MANIPUR', 'NumericSTATE'] = '16'
data2.loc[data2['Area_Name'] == 'MEGHALAYA', 'NumericSTATE'] = '17'
data2.loc[data2['Area_Name'] == 'MIZORAM', 'NumericSTATE'] = '18'
data2.loc[data2['Area_Name'] == 'NAGALAND', 'NumericSTATE'] = '19'
data2.loc[data2['Area_Name'] == 'ODISHA', 'NumericSTATE'] = '20'
data2.loc[data2['Area_Name'] == 'PUNJAB', 'NumericSTATE'] = '21'
data2.loc[data2['Area_Name'] == 'RAJASTHAN', 'NumericSTATE'] = '22'
data2.loc[data2['Area_Name'] == 'SIKKIM', 'NumericSTATE'] = '23'
```

```
data2.loc[data2['Area_Name'] == 'TAMIL NADU', 'NumericSTATE'] = '24'
data2.loc[data2['Area_Name'] == 'TRIPURA', 'NumericSTATE'] = '25'
data2.loc[data2['Area_Name'] == 'UTTAR PRADESH', 'NumericSTATE'] = '26'
data2.loc[data2['Area_Name'] == 'UTTARAKHAND', 'NumericSTATE'] = '27'
data2.loc[data2['Area_Name'] == 'WEST BENGAL', 'NumericSTATE'] = '28'
data2.loc[data2['Area_Name'] == 'ANDAMAN & NICOBAR ISLANDS', 'NumericSTATE'] =
'29'
data2.loc[data2['Area_Name'] == 'CHANDIGARH', 'NumericSTATE'] = '30'
data2.loc[data2['Area_Name'] == 'DADRA & NAGAR HAVELI', 'NumericSTATE'] = '31'
data2.loc[data2['Area_Name'] == 'DAMAN & DIU', 'NumericSTATE'] = '32'
data2.loc[data2['Area_Name'] == 'DELHI', 'NumericSTATE'] = '33'
data2.loc[data2['Area_Name'] == 'LAKSHADWEEP', 'NumericSTATE'] = '34'
data2.loc[data2['Area_Name'] == 'PUDUCHERRY', 'NumericSTATE'] = '35'

#Convert string to integer
pd.to_numeric(data2['NumericSTATE'])
a = data2.head(36)
a

pd.to_numeric(data2['Victims_Between_10-14_Yrs'])
df = pd.DataFrame(data2, columns= ['NumericSTATE', 'Victims_Between_10-14_Yrs'])
df['Victims_Between_10-14_Yrs'].isnull()
df = df.drop(df.index[[35]])
df
plt.rcParams["figure.figsize"] = (20,10)

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['NumericSTATE','Victims_Between_10-14_Yrs']])
y_predicted

df['cluster']=y_predicted
df.head()

km.cluster_centers_

plt.plot(km.cluster_centers_)

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.NumericSTATE,df1['Victims_Between_10-14_Yrs'],color='green')
plt.scatter(df2.NumericSTATE,df2['Victims_Between_10-14_Yrs'],color='yellow')
```

```
plt.scatter(df3.NumericSTATE,df3['Victims_Between_10-14_Yrs'],color='red')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marke
r='*',label='centroid')
plt.xlabel('NumericSTATE')
plt.ylabel('Victims_Between_10-14_Yrs')
plt.legend()
scaler = MinMaxScaler()
scaler.fit(df[['Victims_Between_10-14_Yrs']])
df['Victims_Between_10-14_Yrs'] = scaler.transform(df[['Victims_Between_10-
14_Yrs']])
scaler.fit(df[['NumericSTATE']])
df['NumericSTATE'] = scaler.transform(df[['NumericSTATE']])
df.head()
plt.scatter(df.NumericSTATE,df['Victims_Between_10-14_Yrs'])
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['NumericSTATE','Victims_Between_10-14_Yrs']])
y_predicted
df['cluster']=y_predicted
df.head()
km.cluster_centers_

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.NumericSTATE,df1['Victims_Between_10-14_Yrs'],color='green')
plt.scatter(df2.NumericSTATE,df2['Victims_Between_10-14_Yrs'],color='red')
plt.scatter(df3.NumericSTATE,df3['Victims_Between_10-14_Yrs'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marke
r='*',label='centroid')
plt.legend()

sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['NumericSTATE','Victims_Between_10-14_Yrs']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Linear Regression:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
# %matplotlib inline
import seaborn as sns
data2=pd.read_csv("/content/01_District_wise_crimes_committed_IPC_2001_2012 (1).csv")
data2
#(Linear Regression)
data2.plot(x='ATTEMPT TO MURDER',y='MURDER',style='*',color='c')
plt.title('ATTEMPT TO MURDER vs MURDER',fontdict={'family':'Sans Serif','size':25})
plt.xlabel('ATTEMPT TO MURDER')
plt.ylabel('MURDER')
plt.grid()
plt.show()

count=data2['MURDER'].isna().sum()
count

# we are converting data 2D into 1D data
x=data2['ATTEMPT TO MURDER'].values.reshape(-1,1)
y=data2['MURDER'].values.reshape(-1,1)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

lr=LinearRegression()
lr.fit(x_train,y_train)

print('Intercept of the curve is:',lr.intercept_)
print('Slope of the curve is:',lr.coef_)

y_pred=lr.predict(x_test)

pred=lr.predict(x_test)

print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, pred))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, pred))
print('Root Mean Squared Error:',np.sqrt(metrics.mean_squared_error(y_test, pred)))

x_test.shape

y_test.shape

#y=mx+c Plotting the linear Regression Line
plt.scatter(x_test,y_test,color='c')
plt.plot(x_test,y_pred,color='r',linewidth=2)
plt.title('ATTEMPT TO MURDER vs MURDER',fontdict={'family':'Sans Serif','size':25})
plt.xlabel('ATTEMPT TO MURDER')
plt.ylabel('MURDER')
plt.show()
```

PCY Algorithm:

```
pip install pyspark

from itertools import combinations
from pyspark import SparkContext
sc =SparkContext()

data = sc.textFile("/content/top_crime.txt")

data

data.take(12)

input_list = data.map(lambda x:x.split(','))
input_list = input_list.map(lambda x:sorted(x))
input_list.take(10)

#count each pair in data

pairs_input = input_list.map(lambda x: list(combinations(x,2)))
pairs_input =pairs_input.flatMap(lambda x:x)
count_pair =pairs_input.map(lambda x:(x,1)).sortByKey()
count_pair_data=count_pair.reduceByKey(lambda x,y:x+y)
count_pair_data.take(10)

#count of each item

data = input_list.flatMap(lambda x:x)
data_map = data.map(lambda x:(x,1))
data_count = data_map.reduceByKey(lambda x,y:x+y)
data_count.take(15)

#support =10

freq_item = data_count.filter(lambda x: x[1]>=10).sortByKey()
freq_item=freq_item.map(lambda x:x[0])
freq_item.take(10)

#finding pairs

combination = freq_item.map(lambda x: (1,x))
combination= combination.groupByKey().map(lambda x: (x[0],(list(x[1]))))
comb = combination.map(lambda x: (x[0],(list(combinations(x[1],2)))))
comb = comb.flatMap(lambda x:x[1])
comb.take(20)

#hash

comb=comb.zipWithIndex()
comb.take(100)
pairs_with_bucketno = comb.map(lambda x:(x[0],(x[1]%20)))
pairs_with_bucketno.take(30)
```

```
pairs_with_bucket = pairs_with_bucketno.map(lambda xy:(xy[1],xy[0]))

bucketcount = pairs_with_bucketno.join(count_pair_data)
bucketcount.take(20)

bucket_frequency=bucketcount.map(lambda x:(x[1][0],(x[0],x[1][1]))).sortByKey()
bucket_1 = bucket_frequency.map(lambda x:
(x[0],(x[1][1]))).groupByKey().sortByKey().map(lambda x :
(x[0],(sum(x[1])))).filter(lambda x:x[1]>=4)
freq_bucket = bucket_1.map(lambda x: x[0]).collect()
print(freq_bucket)

bitvector = pairs_with_bucketno.map(lambda x:(x,1 if x[1] in freq_bucket else 0 ))
bits_pairs= bitvector.map(lambda x:(x[0][0],x[1]))
bits_pairs.take(10)

bits_1 = bits_pairs.filter(lambda x: (x[1] ==1))
bits_1.take(15)
freq_itemset_2 = bits_1.map(lambda x : list(x[0]))
freq_itemset_2.take(15)

with open('/content/op.txt', 'w')  as file:
    freq_1=[]
    freq_2=[]
    for i in freq_item.collect():
        freq_1+=i
    file.write(str(freq_1))
    file.write("\n")
    freq_2+=freq_itemset_2.collect()
    file.write(str(freq_2))
```
