Big Data
(6CS030)

Report - Coursework Part 1

Group member Id	: 2036886
	: 2040266
Group member Name	: Sakshyat Sharma
	: Venisha Pandey
Cohort/Batch	: 4
Module Leader	: Jnaneshwar Bohara
Submitted on	: 26-04-2021
Word Count	: 1510

Table of Contents

Table of Figures

Table of Tables

1 Introduction to Big Data

Big data is the collection of large and complicated datasets which include an enormous amount of real-time data. It is also the combination of structured, semi-structured, and unstructured data that is provided by the organization. It is explained by the following 3Vs dimension such as volume, velocity, and variety. Lately, two more dimensions are added veracity and value. The volume in big data shows the size of data, velocity shows the speed of data where variety refers to the nature of data. It has multiple advantages such as organizations can use outside insight while making choices, better operational productivity where it helps the association to offload rarely used data. Big data technologies such as Hadoop, cloud computing makes the work easy by storing huge amount of data also they find an effective way of doing business. It can be found in both public and private areas from marketing to education, healthcare, banking, etc. (J.Anuradha & Ishwarappa, 2015)

2 Introduction to Datasets

2.1 Justification for choice

The CSV and JSON dataset that are used, are both related to crime. The CSV data represents the crime rates recorded in US police departments in different years. The key essence for choosing this dataset is to analyze how much crime is prevailing in different departments of US police. Similarly, the JSON dataset is the record of different types of crimes committed in different places in the UK over a certain period. As the crime rates are increasing day by day in every part of the world, the analysis of this data allows us to find out how rapidly the crime rates are increasing or decreasing in the US as well as the UK over years.

2.2 CSV Dataset

CSV data contains the crime rates in different Police agencies in the US from the year 2001-2014. The data is taken from 'https://data.world/'. The raw dataset has 1160 rows of data and 16 columns which represents the name of the Police Agencies, State, and the number of crime rates in each year from 2001 to 2014. This data allows analyzing the maximum number of crimes, total crime rates, average crime rates recorded in each police department in different years.

2.3 JSON Dataset

JSON dataset contains different types of crimes committed in different months in the UK from the year 2001-2019. The data is taken from 'https://data.gov.uk/'. The data contains a total of 159840 documents that represent different types of crimes committed in the UK during different months in different years. The analysis of this JSON dataset allows getting an overview of what type of crimes are most prevalent in different places in the UK.

3 Import/Cleaning of Data

3.1 Discuss any issue with the data, including a summary of any cleansing needed.

The CSV dataset that has been chosen for analysis, is not suitable to be directly imported and analyzed. Various issues need to be considered before the use of the chosen dataset such as missing data, null values, unnecessary rows, columns and so on which creates redundancy and difficulty in data analysis. For handling the issues as mentioned here, the unnecessary rows and columns and the columns with a large number of null values are completely removed from the dataset. Similarly the mean and median are calculated to fill out any missing data in the dataset. The pivot table is created for better summarization and analysis of the data.

A detailed explanation of data issues and cleansing is found in the appendix section. **[7.1]**

3.2 Discuss which of the 3 databases were appropriate for each dataset and why.

The CSV and JSON datasets are used for analysis using three different databases, Oracle, MongoDB, and Hadoop.

Oracle is usually suitable for analyzing CSV data because it does not directly take in the JSON dataset. Also, it is easier to manipulate and visualize the CSV dataset in oracle in comparison to the JSON data format.

MongoDB works better with the JSON dataset. As it is known that MongoDB is for semi-structured data format and works with key-value pair, the JSON dataset is already in key-value pair making it easier to import and work with. CSV data, on the other hand, is first converted into the key-value pair by MongoDB, before it can be manipulated. Hence, JSON data is better preferred while working in MongoDB.

Hadoop is suited for both CSV and JSON datasets. For the Map Reduce program, CSV data is better preferred to generate the output. In Spark, as it supports SQL queries as well, both CSV and JSON datasets are best fitted for manipulating and analyzing the data.

4 Analysis of the data and visualizations

4.1 Discuss what techniques can be used to query the data. Show the results of the investigation here

4.1.1 Oracle Analysis

CSV dataset has been used for querying the data using oracle. Different SQL queries are used for analyzing these CSV data using Oracle. Similarly, for faster and better analysis, the OLAP technique is used to query the data in oracle. The approach is also beneficial for analyzing statistics as the data is stored in form of OLAP cubes allowing the user to perform dynamic multidimensional analysis of the data.

The following two main OLAP queries are executed for the analysis of data in the oracle

Query:

Here, ROLLUP generates aggregated results for the selected column in a hierarchical pattern. It creates subtotals of each state_department in each year.

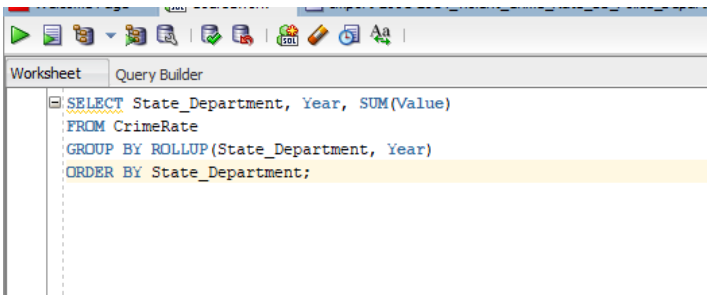


Figure 1 Oracle Query: Rollup

Output:

	STATE_DEPARTMENT	YEAR	SUM(VALUE)
1	Anaheim Police Dept	2001	219.4
2	Anaheim Police Dept	2002	243.2
3	Anaheim Police Dept	2003	245.1
4	Anaheim Police Dept	2004	275.1
5	Anaheim Police Dept	2005	289
6	Anaheim Police Dept	2006	245.8
7	Anaheim Police Dept	2007	217.5
8	Anaheim Police Dept	2008	193.3
9	Anaheim Police Dept	2009	178.3
10	Anaheim Police Dept	2010	170.7
11	Anaheim Police Dept	2011	210.2
12	Anaheim Police Dept	2012	215.4
13	Anaheim Police Dept	2013	173.8
14	Anaheim Police Dept	2014	170.1

Figure 2 Oracle Output: Rollup

Query:

Here, CUBE is used, creates all the combinations of all values in all selected columns, enabling a single statement to calculate all possible combinations of subtotals of each state_departments in each year in this case.

Worksheet | Query Builder

```
SELECT State_Department, Year, SUM(Value)
FROM CrimeRate
GROUP BY CUBE(State_Department, Year)
ORDER BY State_Department;
```

Figure 3 Oracle Query: Cube

Output:

	STATE_DEPARTMENT	YEAR	SUM(VALUE)
1	Anaheim Police Dept	2001	219.4
2	Anaheim Police Dept	2002	243.2
3	Anaheim Police Dept	2003	245.1
4	Anaheim Police Dept	2004	275.1
5	Anaheim Police Dept	2005	289
6	Anaheim Police Dept	2006	245.8
7	Anaheim Police Dept	2007	217.5
8	Anaheim Police Dept	2008	193.3
9	Anaheim Police Dept	2009	178.3
10	Anaheim Police Dept	2010	170.7
11	Anaheim Police Dept	2011	210.2
12	Anaheim Police Dept	2012	215.4
13	Anaheim Police Dept	2013	173.8
14	Anaheim Police Dept	2014	170.1

Figure 4 Oracle Output: Cube

Further analysis of the data in oracle is shown in the appendix section. **[1]**

4.1.2 MongoDB Analysis

MongoDB is used for the analysis of data in JSON format. Its schema-less approach supports every kind of structured, semi-structured, or unstructured data. Because of the JSON format used in MongoDB, it is easy to store arrays and queries. It works in key-value pairs for analysis and manipulation of the data.

The following queries show the name of the databases that are present, the name of the collections, and the total number of data that are present in the collection.

```
---
> show dbs
admin          0.000GB
climate        0.001GB
config         0.000GB
d              0.000GB
kathmandupost  0.001GB
local          0.000GB
policerecord   0.006GB
> use policerecord
switched to db policerecord
> db.crime.count()
159840
>
```

Figure 5 MongoDB – Show how many documents are present

Here, 'show dbs' shows the total number of databases that are available. 'use policerecord' denotes that the database named 'policerecord' will be used for the analysis of data. 'Count()' shows the total number of documents present in the collection, which is a crime in this case. Here, a total of 159840 data are present in the collection named 'crime'.

Further analysis of the data in MongoDB is shown in the appendix section. **[2]**

4.1.3 Hadoop Analysis

Hadoop is capable of analyzing both CSV as well as JSON data formats. It has a distributed file system where, data is distributed on multiple machines as a cluster, which makes it capable to stripe and mirror the data without the use of any third-party tools. Map Reduce and Apache Spark are the two main techniques that have been used here for the analysis of data in Hadoop.

Map Reduce works like an algorithm based on the YARN framework, allowing to perform the distributed processing in parallel Hadoop clusters for faster analysis. Here, the Map-Reduce technique is used for the analysis of the CSV dataset, as seen in the figure below:

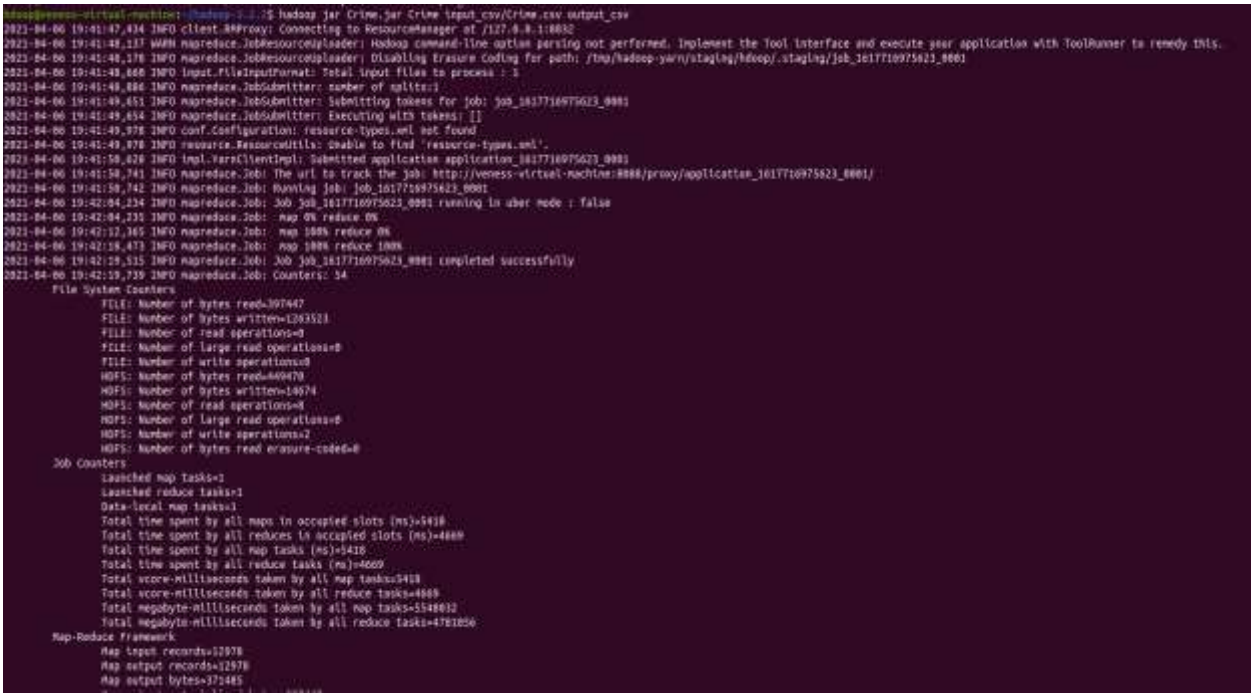


Figure 6 Hadoop - Map Reduce

Similarly, Apache spark is used for data analysis in both CSV as well as JSON datasets. It covers different sorts of batch applications, data processing, and interactive queries making it faster and suitable for general purpose uses. The following are the spark queries executed for the analysis of data in CSV and JSON format.

```
>>> df = spark.read.csv('/home/sakshyat/Downloads/crime.csv')
>>> df.show()
+--------------------+----+------+
|                 _c0| _c1|   _c2|
+--------------------+----+------+
|    State_Department|Year| Value|
|Birmingham Police...|2001| 682.6|
|Birmingham Police...|2002| 692.7|
|Birmingham Police...|2003| 710.3|
|Birmingham Police...|2004| 668.9|
|Birmingham Police...|2005| 714.1|
|Birmingham Police...|2006| 608.8|
|Birmingham Police...|2007| 613.1|
|Birmingham Police...|2008| 637.7|
|Birmingham Police...|2009| 615.3|
|Birmingham Police...|2010|756.85|
|Birmingham Police...|2011| 898.4|
|Birmingham Police...|2012| 954.2|
|Birmingham Police...|2013| 774.5|
|Birmingham Police...|2014| 982.5|
|Huntsville Police...|2001| 358.9|
|Huntsville Police...|2002| 350.2|
|Huntsville Police...|2003|   349|
|Huntsville Police...|2004| 391.4|
|Huntsville Police...|2005| 369.4|
+--------------------+----+------+
only showing top 20 rows

>>> 
```

Figure 7 Spark Query in CSV data

```
>>> df = spark.read.json('/home/sakshyat/Downloads/police-recorded-crime-data.json')
>>> df.show()
+-----+--------------------+--------------------+-----+-----------------+--------------+
|Count|          Crime_Type|        Data_Measure|Month|Policing_District|Calendar_Year|
+-----+--------------------+--------------------+-----+-----------------+--------------+
|  738|Violence with inj...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1498|Violence without ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  133|     Sexual offences|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  158|             Robbery|Police Recorded C...|  Apr| Northern Ireland|          2001|
|    0|Theft - burglary ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|    0|Theft - burglary ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  750|Theft - domestic ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  670|Theft - non-domes...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  101|Theft from the pe...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1829|Theft - vehicle o...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   75|       Bicycle theft|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  442| Theft - shoplifting|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1231|All other theft o...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 3309|     Criminal damage|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   22|Trafficking of drugs|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   83| Possession of drugs|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   36|Possession of wea...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   38|Public order offe...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  244|Miscellaneous cri...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|11357|Total police reco...|Police Recorded C...|  Apr| Northern Ireland|          2001|
+-----+--------------------+--------------------+-----+-----------------+--------------+
only showing top 20 rows

>>> 
```

Figure 8 Spark Query in JSON Data

Further analysis of the data in Hadoop Map-Reduce and Spark is shown in the appendix section. **[3]**

4.2 Show two visualizations appropriate to the analysis

The line graph and bar graph below both shows the overall number of crime rates in the US from the year 2001 to 2014.

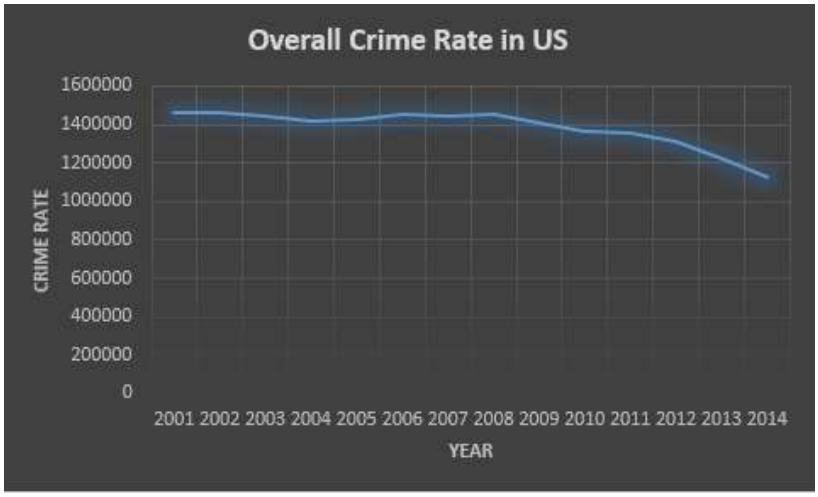


Figure 9 Line Graph (Overall US crime rates)

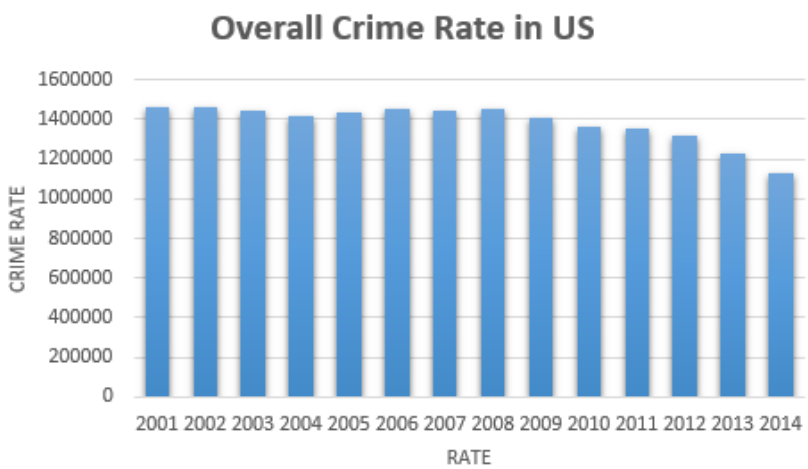


Figure 10 Bar Graph (Overall US crime rates)

Here, it can be seen that year 2002 recorded the highest number of crimes, and the year 2014 recorded the lowest number of crime rates in overall departments of the US. Hence, it can be predicted that there are chances of a decrease in the crime rates in the further years if it goes by the same rate as seen in the graphs above.

Other visualizations of the data can be seen in the appendix section. **[7.4]**

5 Comparison Table

5.1 Advantages of using Oracle, MongoDB and Hadoop for big data

Table 1 Advantages of Oracle, MongoDB and Hadoop

Oracle	MongoDB	Hadoop
It can handle a large number of relational datasets and perform complex analyses with a single query.	Its schema-less approach supports every kind of structured, semi-structured or unstructured data.	It is highly scalable and can store and distribute very large datasets.
It supports more than 90 hardware policies in comparison to another database.	As there is no relationship among data, there is no need of complex joins in MongoDB.	It offers cost effective storage to process massive volumes of data.
It supports flashback technology which allows for efficient recovery of data that has been incorrectly lost.	It features auto-sharding which partitions a large dataset into several small databases, resulting in horizontal scaling.	It provides enough flexibility to handle any kind of structured and unstructured data.
It provides high data availability system by using real application clusters.	The document-based queries used in it are not as complex as SQL queries.	It offers faster processing of data as data is stored in clusters in the distributed system.

(EDUCBA, 2020) (tutorialspoint, 2021) (Nemschoff, 2013)

5.2 Disadvantages of using Oracle, MongoDB and Hadoop for big data

Table 2 Disadvantages of Oracle, MongoDB and Hadoop

Oracle	MongoDB	Hadoop
It is expensive in comparison to other databases.	It has a limit for document size.	It is not suitable for small data sizes.
It is complex to study and use.	It utilizes the large spaces of memory.	It lacks encryption of data at storage.
It is not cost effective to be used by small and medium sized organizations.	It does not have any transaction support.	It is difficult to use and requires complex and detail knowledge.
It only supports a particular kind of dataset.	Mongo caches are stored on device hardware, which can cause memory to crash.	It has stability issues.

(Reference, 2020) (acodez, 2019) (mindsmapped, 2015)

6 Conclusions and Recommendations

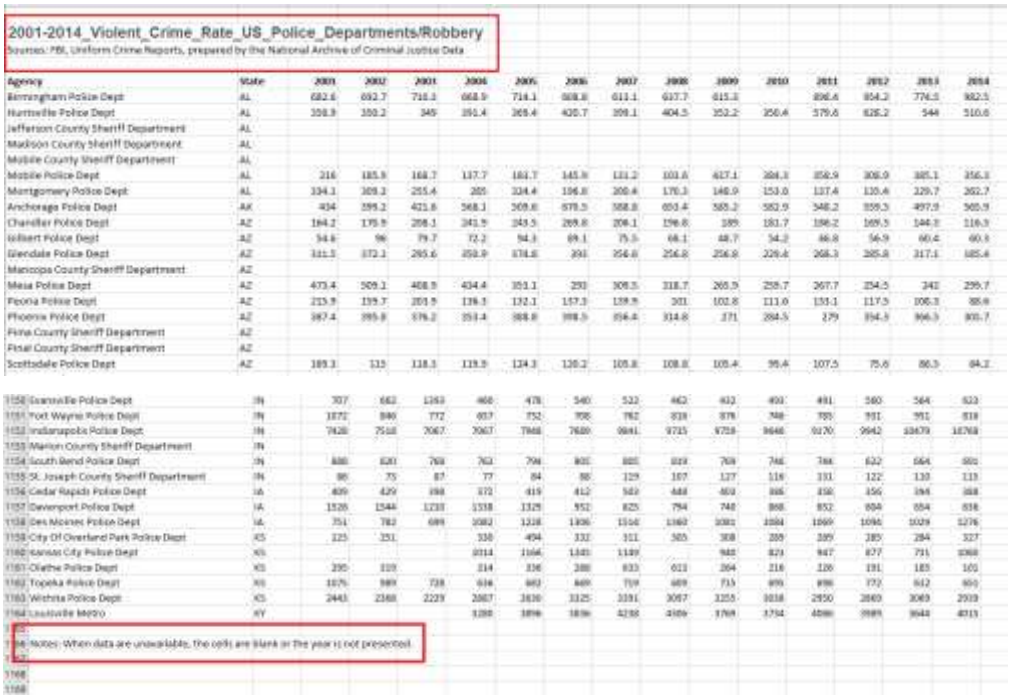
By analyzing and working with all these three databases, it is clear that every database is better in its way and field of use. Oracle database is more applicable at an enterprise level for handling structured data. Similarly, MongoDB is better suited for semi-structured datasets which require real-time processing. Likewise, Hadoop is the best choice for handling and processing a huge volume of a dataset. So, depending upon the field of task and type of data, these databases are chosen as appropriate.

7 Code Appendix

7.1 Cleaning and Manipulation of the Data

Removing Unnecessary Rows and Columns from the csv file

As it can be seen in the figure below, the first 3 rows and the last 1 row of the data is of no use, hence, it is removed to clean the data.



2001-2014_Violent_Crime_Rate_US_Police_Departments/Robbery

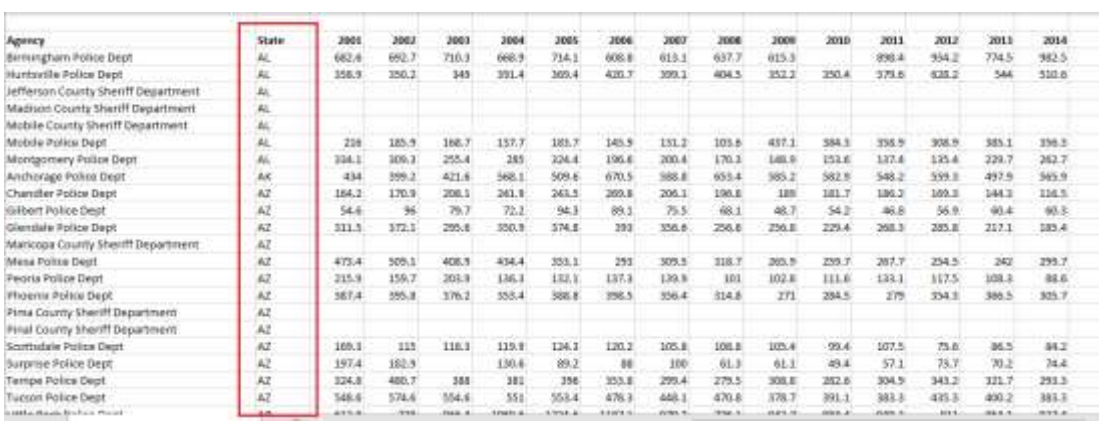
Sources: FBI, Uniform Crime Reports, prepared by the National Archive of Criminal Justice Data

Agency	State	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
Birmingham Police Dept	AL	682.6	692.7	710.3	668.9	714.1	608.8	613.1	637.7	615.3		898.4	954.2	774.5	982.5
Huntsville Police Dept	AL	358.9	350.2	349	391.4	369.4	420.7	399.1	404.5	352.2	350.4	379.6	628.2	544	510.6
Jefferson County Sheriff Department	AL														
Madison County Sheriff Department	AL														
Mobile County Sheriff Department	AL														
Mobile Police Dept	AL	216	185.9	168.7	157.7	181.7	145.9	151.2	103.6	437.1	384.3	356.9	308.9	385.1	356.3
Montgomery Police Dept	AL	334.1	309.3	255.4	285	324.4	196.6	200.4	170.3	148.9	153.6	137.4	125.4	229.7	262.7
Anchorage Police Dept	AK	434	399.2	421.6	568.1	509.6	670.5	588.8	653.4	585.2	582.9	548.2	559.3	497.9	565.9
Chandler Police Dept	AZ	184.2	170.9	208.1	241.9	243.5	269.8	206.1	196.8	189	181.7	186.2	169.3	144.3	116.5
Gilbert Police Dept	AZ	54.6	96	79.7	72.2	94.3	89.1	75.5	68.1	48.7	54.2	46.8	56.9	60.4	60.3
Glendale Police Dept	AZ	311.3	372.1	285.6	350.9	374.8	393	356.6	256.8	256.8	229.4	268.3	285.8	217.1	185.4
Maricopa County Sheriff Department	AZ														
Mesa Police Dept	AZ	473.4	509.1	408.9	434.4	353.1	293	309.5	318.7	265.9	259.7	287.7	254.5	242	299.7
Peoria Police Dept	AZ	215.9	159.7	203.9	136.3	132.1	137.3	139.9	101	102.8	111.6	133.1	117.5	108.3	88.6
Phoenix Police Dept	AZ	587.4	595.8	576.2	553.4	588.8	598.5	556.4	514.8	271	284.5	279	354.3	386.5	305.7
Pima County Sheriff Department	AZ														
Pinal County Sheriff Department	AZ														
Scottsdale Police Dept	AZ	189.3	113	118.3	119.9	124.3	120.2	105.8	108.8	105.4	99.4	107.5	75.6	86.5	84.2
1150 Evansville Police Dept	IN	707	662	1263	466	478	540	532	462	612	490	491	560	564	623
1151 Fort Wayne Police Dept	IN	1072	840	772	657	752	705	782	938	878	748	705	931	951	838
1152 Indianapolis Police Dept	IN	7628	7516	7067	7067	7868	7689	8841	9715	9759	9646	9170	9642	10679	10768
1153 Marion County Sheriff Department	IN														
1154 South Bend Police Dept	IN	880	620	769	763	796	805	885	819	769	746	744	622	664	691
1155 St. Joseph County Sheriff Department	IN	86	75	87	77	84	88	129	167	127	118	131	122	133	113
1156 Cedar Rapids Police Dept	IA	409	429	398	372	419	412	549	448	459	386	356	356	394	388
1157 Davenport Police Dept	IA	1526	1544	1233	1538	1325	951	825	794	748	868	852	894	854	838
1158 Des Moines Police Dept	IA	751	782	699	1082	1218	1306	1516	1360	1081	1084	1069	1096	1029	1276
1159 City Of Overland Park Police Dept	KS	125	151		338	494	332	311	505	308	289	289	285	284	327
1160 Kansas City Police Dept	KS				2014	1164	1345	1149		948	821	947	877	791	1068
1161 Olathe Police Dept	KS	295	319		214	336	288	833	613	264	216	226	191	183	161
1162 Topeka Police Dept	KS	1075	989	738	616	692	669	759	689	719	899	898	772	612	691
1163 Wichita Police Dept	KS	2443	2368	2229	2087	2630	3125	3191	3067	3255	3018	2950	2860	3063	2919
1164 Louisville Metro	KY				1288	1898	1836	4238	4106	3769	3794	4086	3989	3644	4013

Notes: When data are unavailable, the cells are blank or the year is not presented.

Figure 11 Unnecessary Rows removal

Here, the state column might seem important but it has no any significant effect on the rest of the columns of data. Also, although the states are represented with different names, it denotes the same data as represented by the 'Agency' column. Hence, it is completely removed.



Agency	State	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
Birmingham Police Dept	AL	682.6	692.7	710.3	668.9	714.1	608.8	613.1	637.7	615.3		898.4	954.2	774.5	982.5
Huntsville Police Dept	AL	358.9	350.2	349	391.4	369.4	420.7	399.1	404.5	352.2	350.4	379.6	628.2	544	510.6
Jefferson County Sheriff Department	AL														
Madison County Sheriff Department	AL														
Mobile County Sheriff Department	AL														
Mobile Police Dept	AL	216	185.9	168.7	157.7	181.7	145.9	151.2	103.6	437.1	384.3	356.9	308.9	385.1	356.3
Montgomery Police Dept	AL	334.1	309.3	255.4	285	324.4	196.6	200.4	170.3	148.9	153.6	137.4	125.4	229.7	262.7
Anchorage Police Dept	AK	434	399.2	421.6	568.1	509.6	670.5	588.8	653.4	585.2	582.9	548.2	559.3	497.9	565.9
Chandler Police Dept	AZ	184.2	170.9	208.1	241.9	243.5	269.8	206.1	196.8	189	181.7	186.2	169.3	144.3	116.5
Gilbert Police Dept	AZ	54.6	96	79.7	72.2	94.3	89.1	75.5	68.1	48.7	54.2	46.8	56.9	60.4	60.3
Glendale Police Dept	AZ	311.3	372.1	285.6	350.9	374.8	393	356.6	256.8	256.8	229.4	268.3	285.8	217.1	185.4
Maricopa County Sheriff Department	AZ														
Mesa Police Dept	AZ	473.4	509.1	408.9	434.4	353.1	293	309.5	318.7	265.9	259.7	287.7	254.5	242	299.7
Peoria Police Dept	AZ	215.9	159.7	203.9	136.3	132.1	137.3	139.9	101	102.8	111.6	133.1	117.5	108.3	88.6
Phoenix Police Dept	AZ	587.4	595.8	576.2	553.4	588.8	598.5	556.4	514.8	271	284.5	279	354.3	386.5	305.7
Pima County Sheriff Department	AZ														
Pinal County Sheriff Department	AZ														
Scottsdale Police Dept	AZ	189.3	113	118.3	119.9	124.3	120.2	105.8	108.8	105.4	99.4	107.5	75.6	86.5	84.2
Surprise Police Dept	AZ	197.4	182.9		130.6	89.2	88	100	61.3	61.1	49.4	57.1	73.7	70.2	74.4
Tempe Police Dept	AZ	324.8	480.7	388	381	398	353.8	299.4	279.5	308.8	282.6	304.9	343.2	321.7	293.3
Tucson Police Dept	AZ	548.6	574.6	554.6	551	553.4	476.3	448.1	470.8	378.7	391.1	383.3	435.3	400.2	383.3

Figure 12 Unwanted Column Removal

Pivoting

The pivot table is created for better summarization of the data.

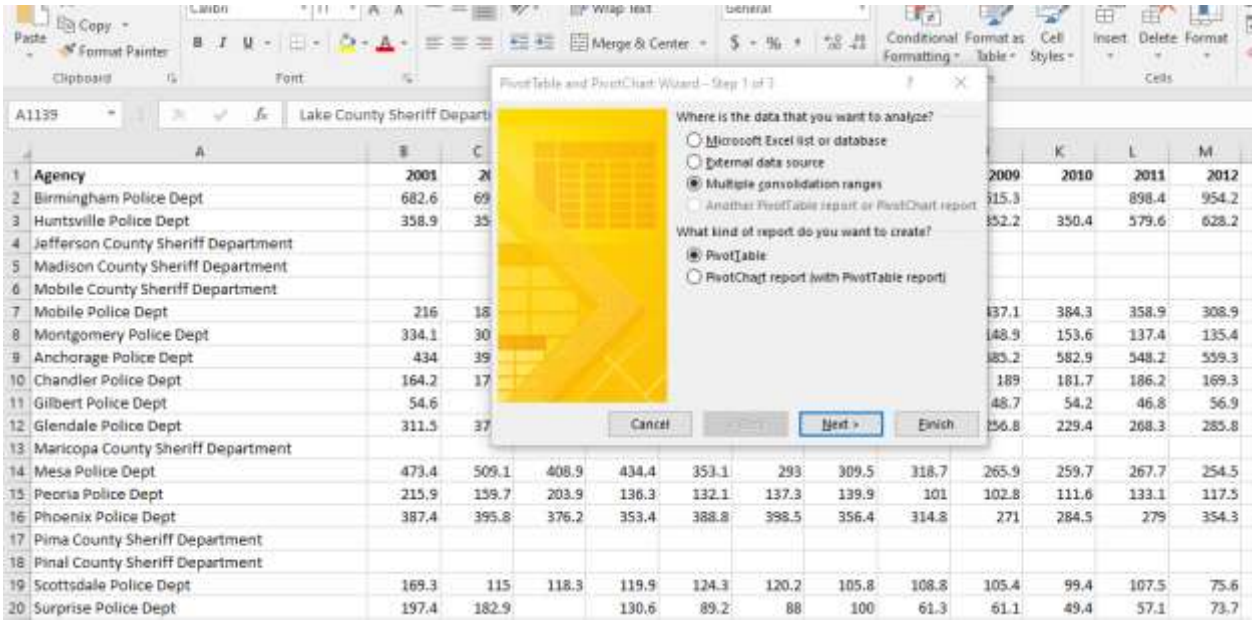


Figure 13 Pivot Wizard

	Row	Column	Value
2	Birmingham Police Dept	2001	682.6
3	Birmingham Police Dept	2002	692.7
4	Birmingham Police Dept	2003	710.3
5	Birmingham Police Dept	2004	668.9
6	Birmingham Police Dept	2005	714.1
7	Birmingham Police Dept	2006	608.8
8	Birmingham Police Dept	2007	613.1
9	Birmingham Police Dept	2008	637.7
10	Birmingham Police Dept	2009	615.3
11	Birmingham Police Dept	2010	
12	Birmingham Police Dept	2011	898.4
13	Birmingham Police Dept	2012	954.2
14	Birmingham Police Dept	2013	774.5
15	Birmingham Police Dept	2014	982.5
16	Huntsville Police Dept	2001	358.9
17	Huntsville Police Dept	2002	350.2
18	Huntsville Police Dept	2003	349
19	Huntsville Police Dept	2004	391.4
20	Huntsville Police Dept	2005	369.4
21	Huntsville Police Dept	2006	420.7
22	Huntsville Police Dept	2007	399.1
23	Huntsville Police Dept	2008	404.5

Figure 14 Pivot table

Deleting Rows with complete NULL Values

The rows having completely null value from the year 2001-2014 is completely removed from the dataset.

30	Jefferson County Sheriff Department	2001	
31	Jefferson County Sheriff Department	2002	
32	Jefferson County Sheriff Department	2003	
33	Jefferson County Sheriff Department	2004	
34	Jefferson County Sheriff Department	2005	
35	Jefferson County Sheriff Department	2006	
36	Jefferson County Sheriff Department	2007	
37	Jefferson County Sheriff Department	2008	
38	Jefferson County Sheriff Department	2009	
39	Jefferson County Sheriff Department	2010	
40	Jefferson County Sheriff Department	2011	
41	Jefferson County Sheriff Department	2012	
42	Jefferson County Sheriff Department	2013	
43	Jefferson County Sheriff Department	2014	

Figure 15 Deleting Rows with null data

Missing values:

Many of the rows have some missing data. In such case, if the data is missing in the middle, the mean value is calculated using the succeeding and preceding data to fill out the missing data. Similarly, if the data is missing in the first or last, the median is calculated to find out the missing value.

Missing Value:

	Row	Column	Value
2	Birmingham Police Dept	2001	682.6
3	Birmingham Police Dept	2002	692.7
4	Birmingham Police Dept	2003	710.3
5	Birmingham Police Dept	2004	668.9
6	Birmingham Police Dept	2005	714.1
7	Birmingham Police Dept	2006	608.8
8	Birmingham Police Dept	2007	613.1
9	Birmingham Police Dept	2008	637.7
10	Birmingham Police Dept	2009	615.3
11	Birmingham Police Dept	2010	
12	Birmingham Police Dept	2011	898.4
13	Birmingham Police Dept	2012	954.2
14	Birmingham Police Dept	2013	774.5

Figure 16 Row with missing value

Mean Value:

6	Birmingham Police Dept	2005	714.1
7	Birmingham Police Dept	2006	608.8
8	Birmingham Police Dept	2007	613.1
9	Birmingham Police Dept	2008	637.7
10	Birmingham Police Dept	2009	615.3
11	Birmingham Police Dept	2010	756.85
12	Birmingham Police Dept	2011	898.4
13	Birmingham Police Dept	2012	954.2
14	Birmingham Police Dept	2013	774.5
15	Birmingham Police Dept	2014	982.5

Figure 17 Filling missing data with mean value

Renaming the column head

The column names are changed into appropriate forms so that it can be easier for analysis of the data.

	A	B	C	D
1	State_Department	Year	Value	
2	Birmingham Police Dept	2001	682.6	
3	Birmingham Police Dept	2002	692.7	
4	Birmingham Police Dept	2003	710.3	
5	Birmingham Police Dept	2004	668.9	
6	Birmingham Police Dept	2005	714.1	
7	Birmingham Police Dept	2006	608.8	
8	Birmingham Police Dept	2007	613.1	
9	Birmingham Police Dept	2008	637.7	
10	Birmingham Police Dept	2009	615.3	
11	Birmingham Police Dept	2010	756.85	
12	Birmingham Police Dept	2011	898.4	
13	Birmingham Police Dept	2012	954.2	
14	Birmingham Police Dept	2013	774.5	

Figure 18 Renaming Columns

7.2 Importing the Data

Oracle

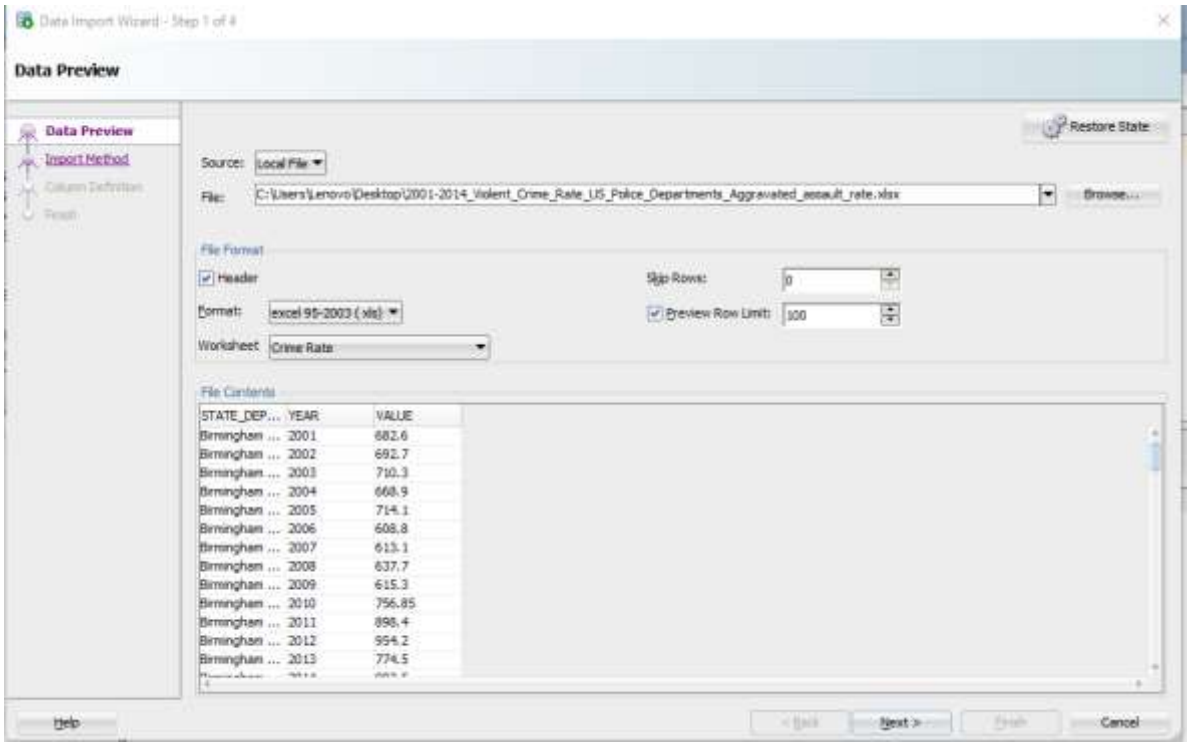


Figure 19 Oracle Import: Data Preview

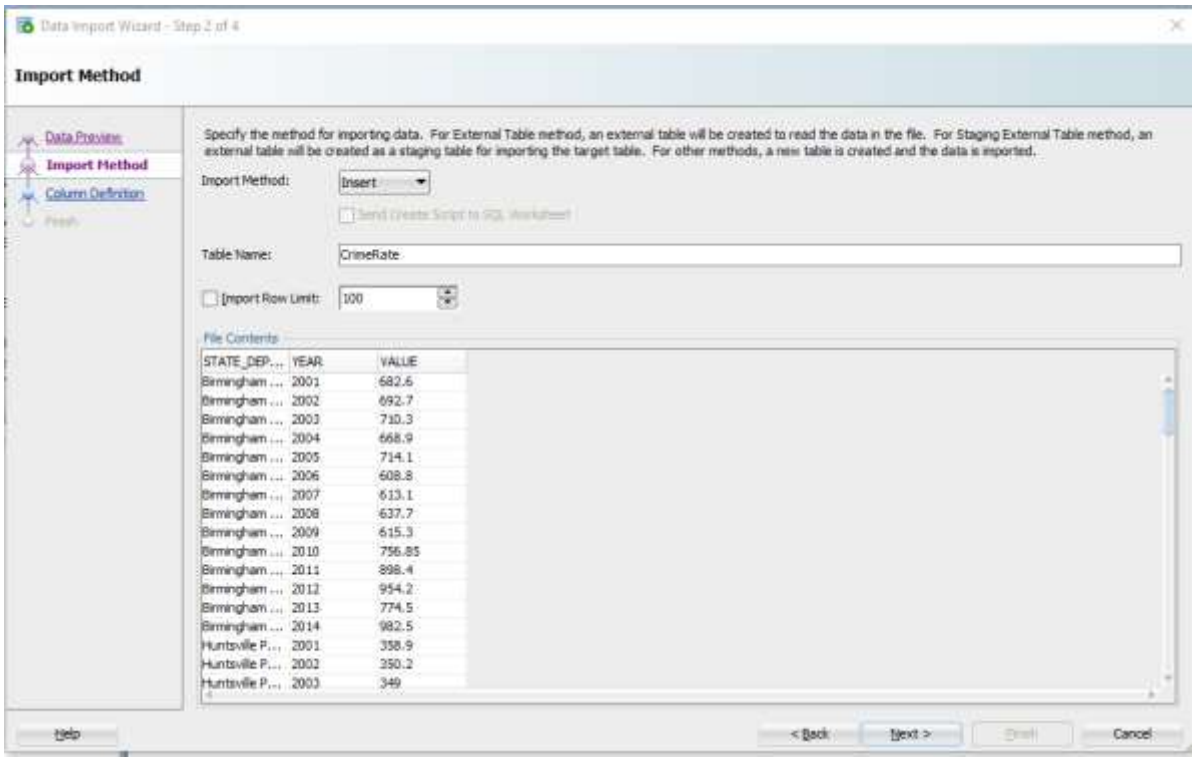


Figure 20 Oracle Import: Import Method

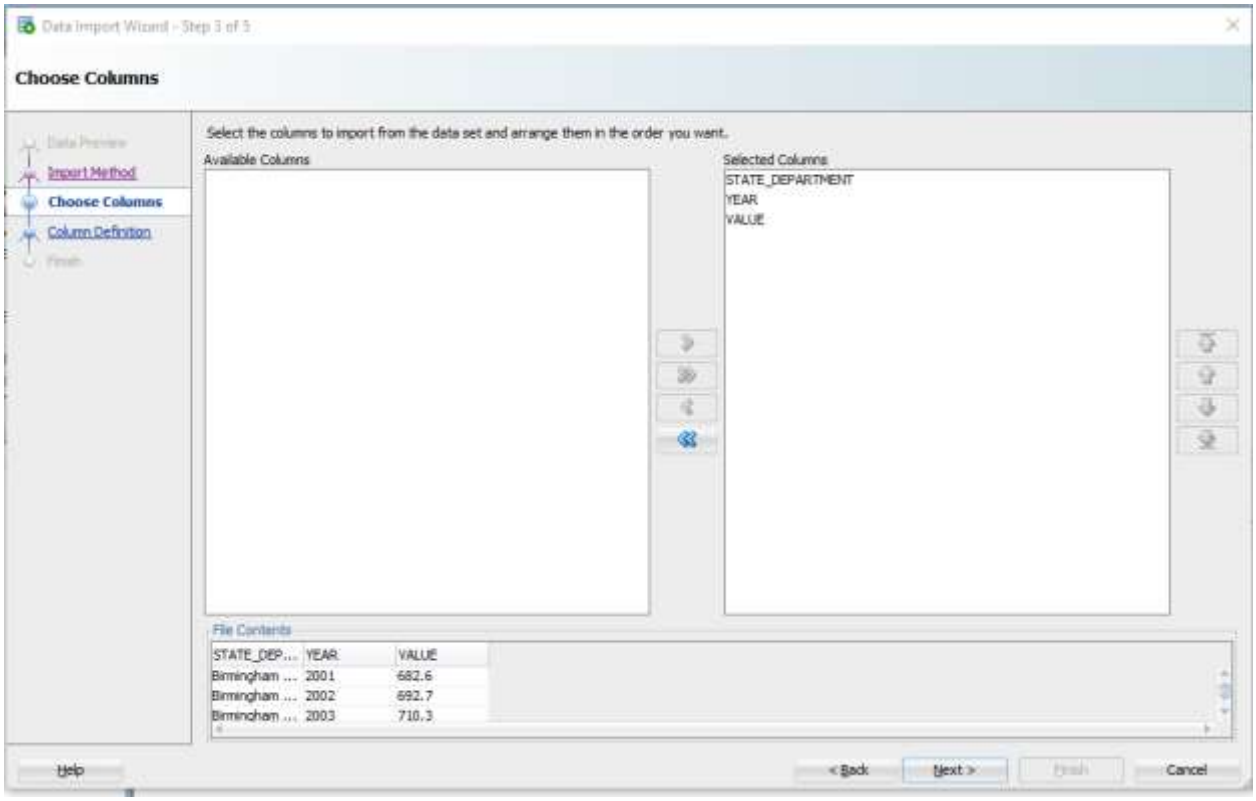


Figure 21 Oracle Import: Choose Columns

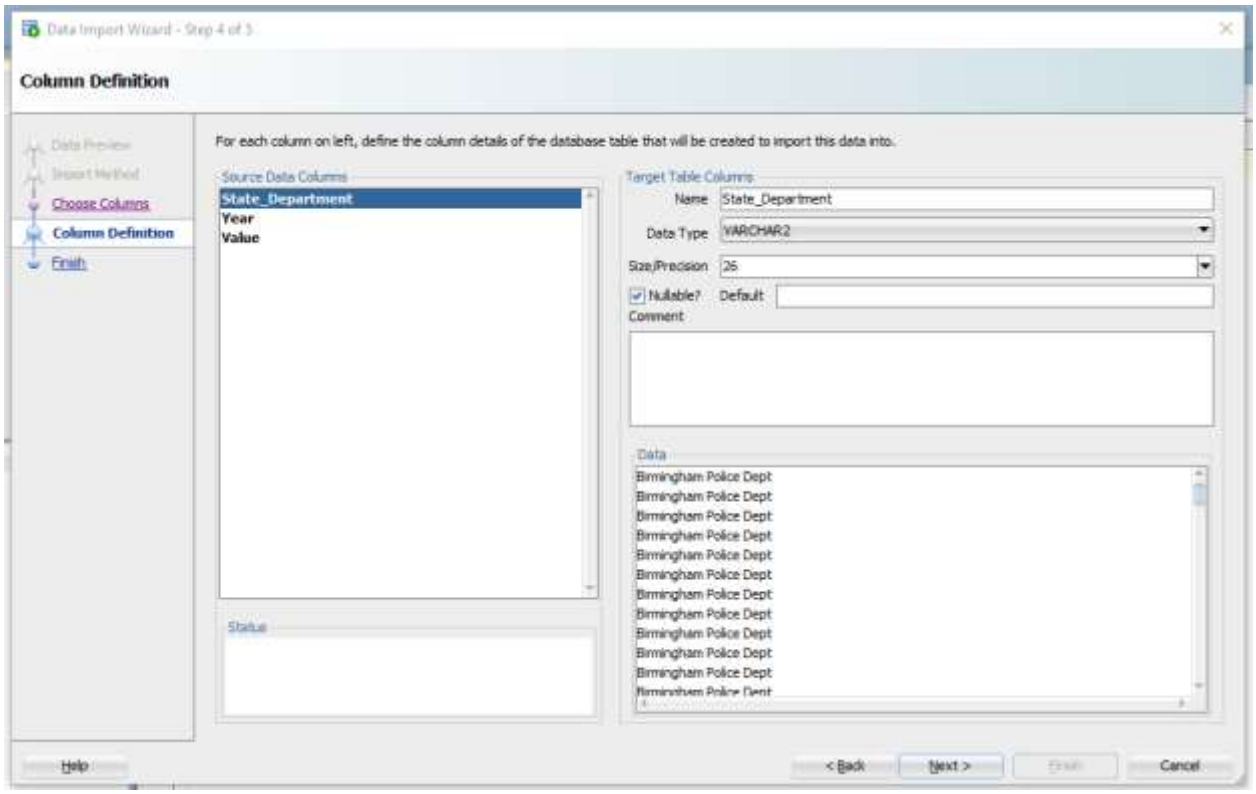


Figure 22 Oracle Import: Column Definition

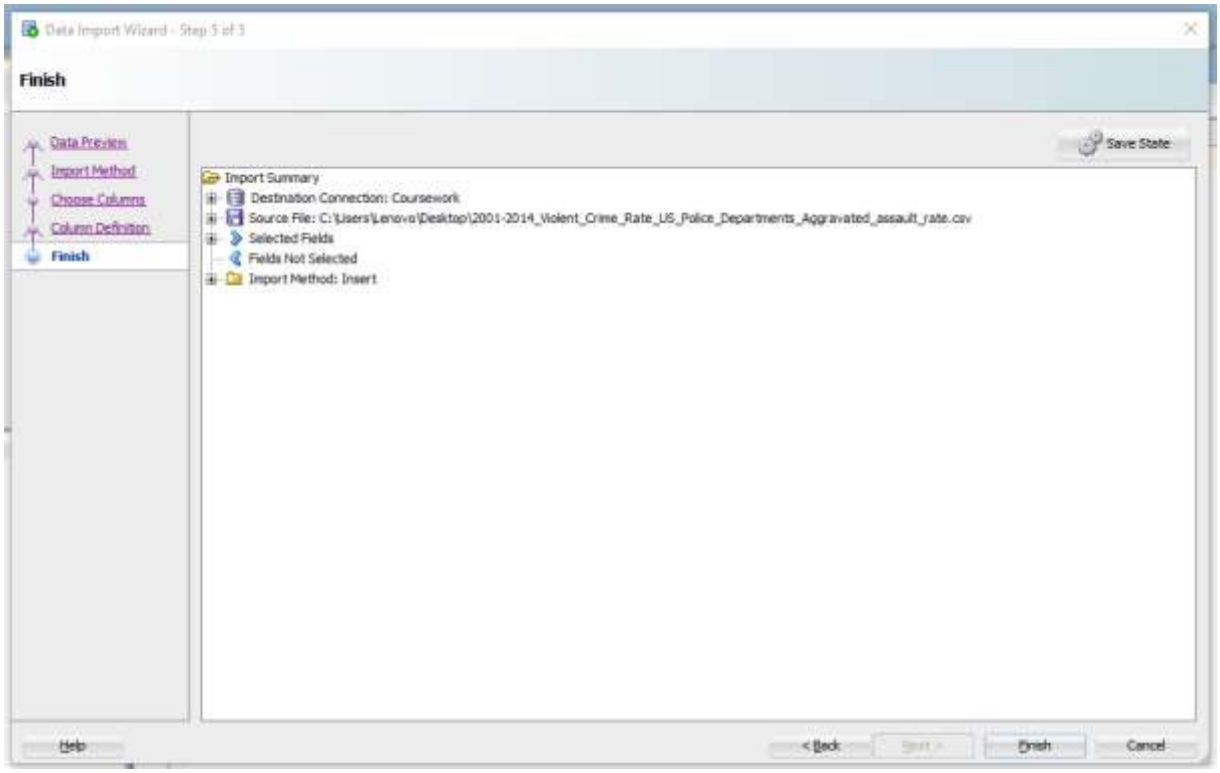


Figure 23 Oracle Import: Finished

Imported Data:

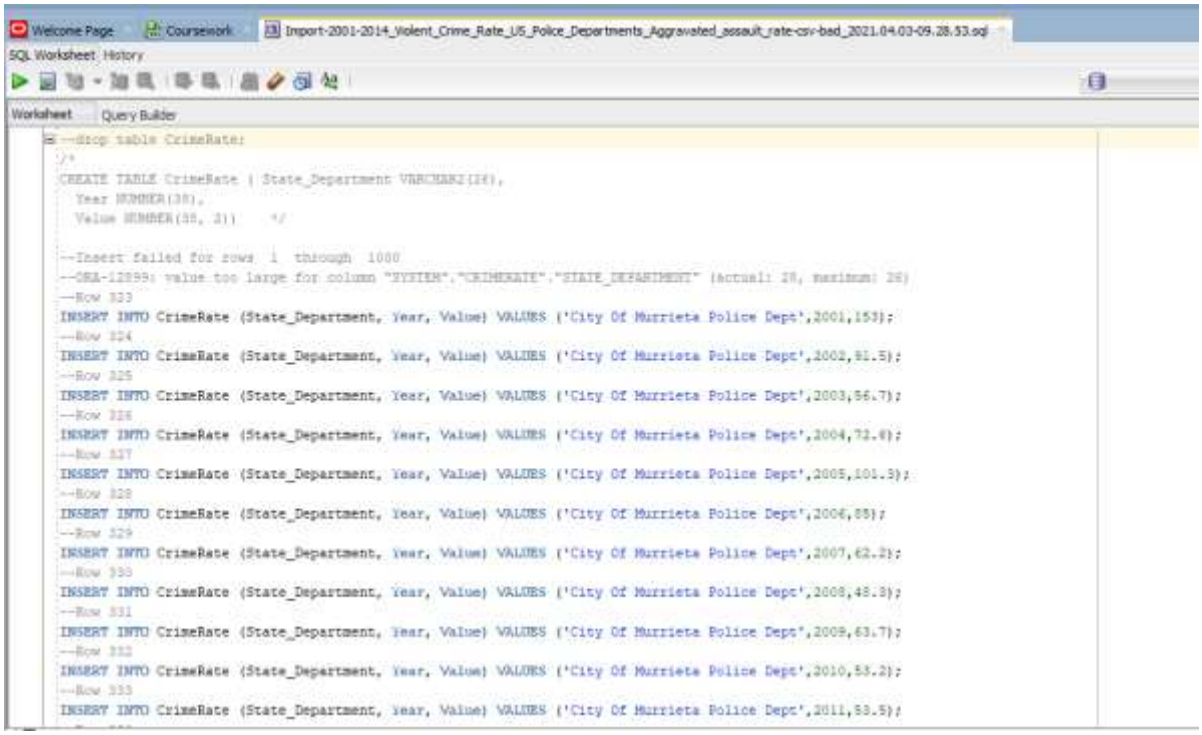
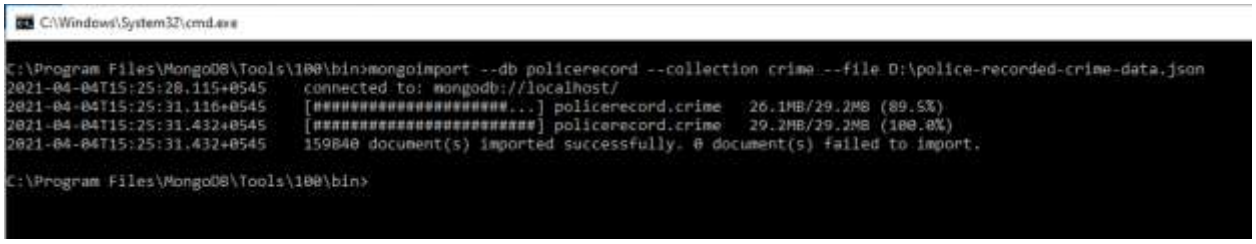


Figure 24 Oracle Imported Data

MongoDB



```
C:\Windows\System32\cmd.exe

C:\Program Files\MongoDB\Tools\100\bin>mongoimport --db policerecord --collection crime --file D:\police-recorded-crime-data.json
2021-04-04T15:25:28.115+0545    connected to: mongodb://localhost/
2021-04-04T15:25:31.116+0545    [#####################...] policerecord.crime   26.1MB/29.2MB (89.5%)
2021-04-04T15:25:31.432+0545    [#########################] policerecord.crime   29.2MB/29.2MB (100.0%)
2021-04-04T15:25:31.432+0545    159840 document(s) imported successfully. 0 document(s) failed to import.

C:\Program Files\MongoDB\Tools\100\bin>
```

Figure 25 MongoDB Data import

Here, ‘mongoimport’ command is used to import the JSON dataset from the specified path into the MongoDB database.

7.3 Analysis of Data

1. Oracle

Query:

The following query displays the total number of data in the CrimeRate table.

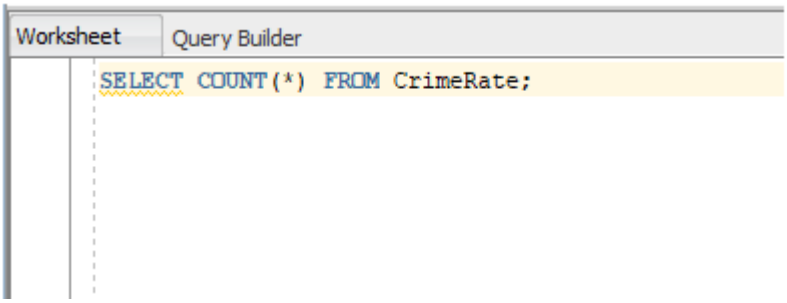


Figure 26 Oracle Query: Count

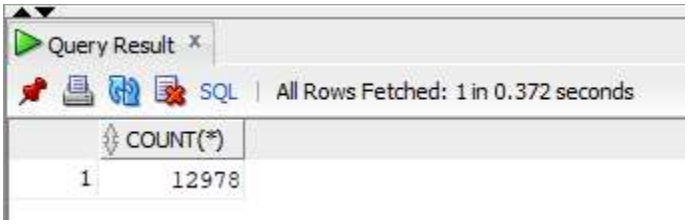


Figure 27 Oracle Output: Count

Query:

The following query shows all the data in the CrimeRate table.

Worksheet | Query Builder

```
SELECT * FROM CrimeRate;
```

Figure 28 Oracle Query: Display all data

Output:

	STATE_DEPARTMENT	YEAR	VALUE
1	Birmingham Police Dept	2001	682.6
2	Birmingham Police Dept	2002	692.7
3	Birmingham Police Dept	2003	710.3
4	Birmingham Police Dept	2004	668.9
5	Birmingham Police Dept	2005	714.1
6	Birmingham Police Dept	2006	608.8
7	Birmingham Police Dept	2007	613.1
8	Birmingham Police Dept	2008	637.7
9	Birmingham Police Dept	2009	615.3
10	Birmingham Police Dept	2010	756.85
11	Birmingham Police Dept	2011	898.4

Figure 29 Oracle Output: Display all data

Query:

The following query shows the total number of crime cases in each police department. Here, TO_CHAR is used to format the numbers to include 1000 separators.

Worksheet | Query Builder

```
SELECT State_Department, TO_CHAR(SUM(Value), '999,999,999') AS "Total Crimes"
FROM CrimeRate
Group BY State_Department
ORDER BY State_Department;
```

Figure 30 Oracle Query: SUM

Output:

Query Result

SQL | Fetched 50 rows in 0.071 seconds

	STATE_DEPARTMENT	Total Crimes
1	Abilene Police Dept	22,045
2	Ada County Sheriff Department	5,677
3	Aiken County Sheriff Department	14,701
4	Akron City Police Dept	52,558
5	Alachua County Sheriff Department	25,775
6	Alameda County Sheriff Department	17,694
7	Albuquerque Police Dept	93,047
8	Alexandria Police Dept	6,659
9	Allentown City Police Dept	20,789
10	Amarillo Police Dept	39,808
11	Amherst Town Police Dept	3,712

Figure 31 Oracle output: SUM

Query:

The following query searches the data based on some criteria using WHERE clause. Here, all the data where the department is ‘Palmdale’ is shown.

Worksheet | Query Builder

```
SELECT * FROM CrimeRate
WHERE State_Department = 'Palmdale'
ORDER BY Year;
```

Figure 32 Oracle Query: WHERE Clause

Output:

	STATE_DEPARTMENT	YEAR	VALUE
1	Palmdale	2001	907
2	Palmdale	2002	1155
3	Palmdale	2003	920
4	Palmdale	2004	996
5	Palmdale	2005	1042
6	Palmdale	2006	989
7	Palmdale	2007	1043
8	Palmdale	2008	876
9	Palmdale	2009	818
10	Palmdale	2010	832
11	Palmdale	2011	785
12	Palmdale	2012	812
13	Palmdale	2013	759
14	Palmdale	2014	840

Figure 33 Oracle Output: WHERE Clause

Query:

Here, the query shows the average crime cases of each department. ROUND is used to round up the decimal value to an integer.

Worksheet | Query Builder

```
SELECT State_Department, ROUND(AVG(Value)) AS "Average Crimes"
FROM CrimeRate
Group BY State_Department
ORDER BY State_Department;
```

Figure 34 Oracle Query: Average

Output:

Query Result

SQL | Fetched 50 rows in 0.052 seconds

	STATE_DEPARTMENT	Average Crimes
1	Abilene Police Dept	787
2	Ada County Sheriff Department	203
3	Aiken County Sheriff Department	1050
4	Akron City Police Dept	1877
5	Alachua County Sheriff Department	921
6	Alameda County Sheriff Department	632
7	Albuquerque Police Dept	3323
8	Alexandria Police Dept	238
9	Allentown City Police Dept	742
10	Amarillo Police Dept	1422
11	Amherst Town Police Dept	133

Figure 35 Oracle Output: Average

Query:

The query below is the nested query, which displays the departments with the highest number of crime record in each year from 2001 to 2014.

Worksheet | Query Builder

```
SELECT Year, State_Department, Value
FROM CrimeRate cr1
WHERE value = (SELECT MAX(cr2.Value) FROM CrimeRate cr2 WHERE cr1.year = cr2.year)
ORDER BY Year;
```

Figure 36 Qracle Query: Nested Query (Department with maximum crime rate each year)

Output:

Query Result

SQL | All Rows Fetched: 14 in 5.225 seconds

	YEAR	STATE_DEPARTMENT	VALUE
1	2001	Los Angeles Police Dept	52243
2	2002	Los Angeles Police Dept	51695
3	2003	Los Angeles Police Dept	48824
4	2004	Los Angeles Police Dept	42786
5	2005	Los Angeles Police Dept	31767
6	2006	Los Angeles Police Dept	30526
7	2007	Houston Police Dept	29044
8	2008	Houston Police Dept	26947
9	2009	Houston Police Dept	29279
10	2010	Houston Police Dept	27924
11	2011	Houston Police Dept	27459
12	2012	Houston Police Dept	26630
13	2013	Houston Police Dept	23733
14	2014	Houston Police Dept	21629

Figure 37 Qracle Output: Nested Query (Department with maximum crime rate each year)

2. MongoDB

The following query shows one document form the collection.

```
C:\Windows\System32\cmd.exe - mongo
> db.crime.findOne()
{
        "_id" : ObjectId("6069898c381605d79e398295"),
        "Count" : "133",
        "Crime_Type" : "Sexual offences",
        "Data_Measure" : "Police Recorded Crime",
        "Month" : "Apr",
        "Policing_District" : "Northern Ireland",
        " Calendar_Year" : "2001"
}
>
```

Figure 38 MongoDB: Show one document

The following query shows the unique values in a field. Here, in this case, the unique Policing_Districts names are shown by using the distinct() query.

```
> db.crime.distinct("Policing_District")
[
        "Antrim & Newtownabbey",
        "Ards & North Down",
        "Armagh City Banbridge & Craigavon",
        "Belfast City",
        "Causeway Coast & Glens",
        "Derry City & Strabane ",
        "Fermanagh & Omagh",
        "Lisburn & Castlereagh City",
        "Mid & East Antrim",
        "Mid Ulster",
        "Newry Mourne & Down",
        "Northern Ireland"
]
>
```

Figure 39 MongoDB: Show unique values

The following query displays the result based on some criteria. Here, all the documents having the crime_type as ‘Bicycle theft’ is displayed. Also, only two fields from the document that are ‘crime_type’ and ‘policing_district’ are displayed.

```
> db.crime.find({"Crime_Type": "Bicycle theft"},{"Crime_Type":1, "Policing_District":1, "_id":0})
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Northern Ireland" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Belfast City" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Lisburn & Castlereagh City" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Ards & North Down" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Newry Mourne & Down" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Armagh City Banbridge & Craigavon" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Mid Ulster" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Fermanagh & Omagh" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Derry City & Strabane " }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Causeway Coast & Glens" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Mid & East Antrim" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Antrim & Newtownabbey" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Northern Ireland" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Belfast City" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Lisburn & Castlereagh City" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Ards & North Down" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Newry Mourne & Down" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Armagh City Banbridge & Craigavon" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Mid Ulster" }
{ "Crime_Type" : "Bicycle theft", "Policing_District" : "Fermanagh & Omagh" }
Type "it" for more
>
```

Figure 40 MongoDB: Show result based on some criteria

The following query uses regular expression to search the result with certain criteria. Here, $regex denotes the regular expression, and all the documents containing the word ‘drugs’ is displayed. Also, ‘i’ denotes case insensitivity.

```
> db.crime.find({"Crime_Type": {$regex: /drugs/i}},{"Crime_Type":1})
{ "_id" : ObjectId("6069898c381605d79e3982a0"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982a2"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982b7"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982bd"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982ca"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982ce"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982e0"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982e2"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982f2"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982f4"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398305"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398307"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39831b"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39831c"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39832d"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398331"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398343"), "Crime_Type" : "Possession of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398344"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398354"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398358"), "Crime_Type" : "Possession of drugs" }
Type "it" for more
>
```

Figure 41 MongoDB: Regular Expression $regex

The following query is executed in order to update the fields in a document. Here, updateMany() is used to update multiple fields in a document all at the same time. In this case, the field ‘Crime_Type’ having the name ‘Trafficking of drugs’ is renamed to ‘Drug trafficking’.

```
> db.crime.updateMany({"Crime_Type": "Trafficking of drugs"},{$set: {"Crime_Type": "Drug Trafficking"}})
{ "acknowledged" : true, "matchedCount" : 7990, "modifiedCount" : 7990 }
>
```

Figure 42 MongoDB: Update Many fields in a document

Update results can be seen as follows:

Before update:

```
> db.crime.find({"Crime_Type": "Trafficking of drugs"},{"Crime_Type":1})
{ "_id" : ObjectId("6069898c381605d79e3982a0"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982b7"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982ce"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982e2"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3982f2"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398305"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39831b"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39832d"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398344"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398354"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39836c"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398380"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398399"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3983a7"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3983b9"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3983cd"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3983e3"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e3983f6"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e398409"), "Crime_Type" : "Trafficking of drugs" }
{ "_id" : ObjectId("6069898c381605d79e39841e"), "Crime_Type" : "Trafficking of drugs" }
Type "it" for more
>
```

Figure 43 MongoDB: Before Update

After Update:

```
> db.crime.find({"Crime_Type": "Trafficking of drugs"},{"Crime_Type":1})
>
>
> db.crime.find({"Crime_Type": "Drug Trafficking"},{"Crime_Type":1})
{ "_id" : ObjectId("6069898c381605d79e3982a0"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3982b7"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3982ce"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3982e2"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3982f2"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398305"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e39831b"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e39832d"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398344"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398354"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e39836c"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398380"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398399"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3983a7"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3983b9"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3983cd"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3983e3"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e3983f6"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e398409"), "Crime_Type" : "Drug Trafficking" }
{ "_id" : ObjectId("6069898c381605d79e39841e"), "Crime_Type" : "Drug Trafficking" }
Type "it" for more
>
```

Figure 44 MongoDB: After Update

The following query creates a new document as a subset of the existing document. That means a new document is created which contains only certain fields from the current existing document. $project here, denotes that the document with the requested fields is now passed on to the next stage in a pipeline, and $out takes the document returned by the aggregation pipeline and writes it to the specified collection, which is “newPoliceRecord” in this case.

```
> db.crime.aggregate([{$project: {"Crime_Type":1, "Policing_District":1, " Calendar_Year":1, "Month":1}},{$out: "newCPoliceRecord"}])
>
> db.newCPoliceRecord.findOne()
{
        "_id" : ObjectId("6069898c381605d79e398295"),
        "Crime_Type" : "Sexual offences",
        "Month" : "Apr",
        "Policing_District" : "Northern Ireland",
        " Calendar_Year" : "2001"
}
>
```

Figure 45 MongoDB: Aggregation

3. Hadoop

The following command is used for copying the csv dataset file to the Hadoop home directory for analysis purpose.

```
hdoop@veness-virtual-machine:~$ cp /home/veness/Downloads/crime.csv /home/hdoop/hadoop-3.2.2/
hdoop@veness-virtual-machine:~$
```

Figure 46 Hadoop - Copying the dataset into hadoop directory

The following command is used for copying the java file used for word count, to the Hadoop home directory.

```
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ cp /home/veness/Downloads/Crime.java /home/hdoop/hadoop-3.2.2/
hdoop@veness-virtual-machine:~/hadoop-3.2.2$
```

Figure 47 Hadoop - Copying the java file into hadoop directory

The first javac command here, is used for compiling the 'Crime.java' filed which was copied to the Hadoop dome directly before. The next 'jar cf' command creates a jar file named 'Crime.jar' in order to execute the java class.

```
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ javac -classpath $(hadoop classpath) Crime.java
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ jar cf Crime.jar Crime*.class
hdoop@veness-virtual-machine:~/hadoop-3.2.2$
```

Figure 48 Hadoop - Compiling java file and creating jar file

The command below creates a input directory named 'input_csv' on the hdfs.

```
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ hdfs dfs -mkdir input_csv
```

Figure 49 Hadoop - Creating input directory

The following command is used to put the csv file into the input directory that was created before.

```
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ hdfs dfs -put crime.csv input_csv
hdoop@veness-virtual-machine:~/hadoop-3.2.2$
```

Figure 50 Hadoop - Putting csv file into input directory

Here, hadoop jar command is used for running the Map Reduce Program, as seen below, and the output is stored in the output directory on the hdfs as specified.

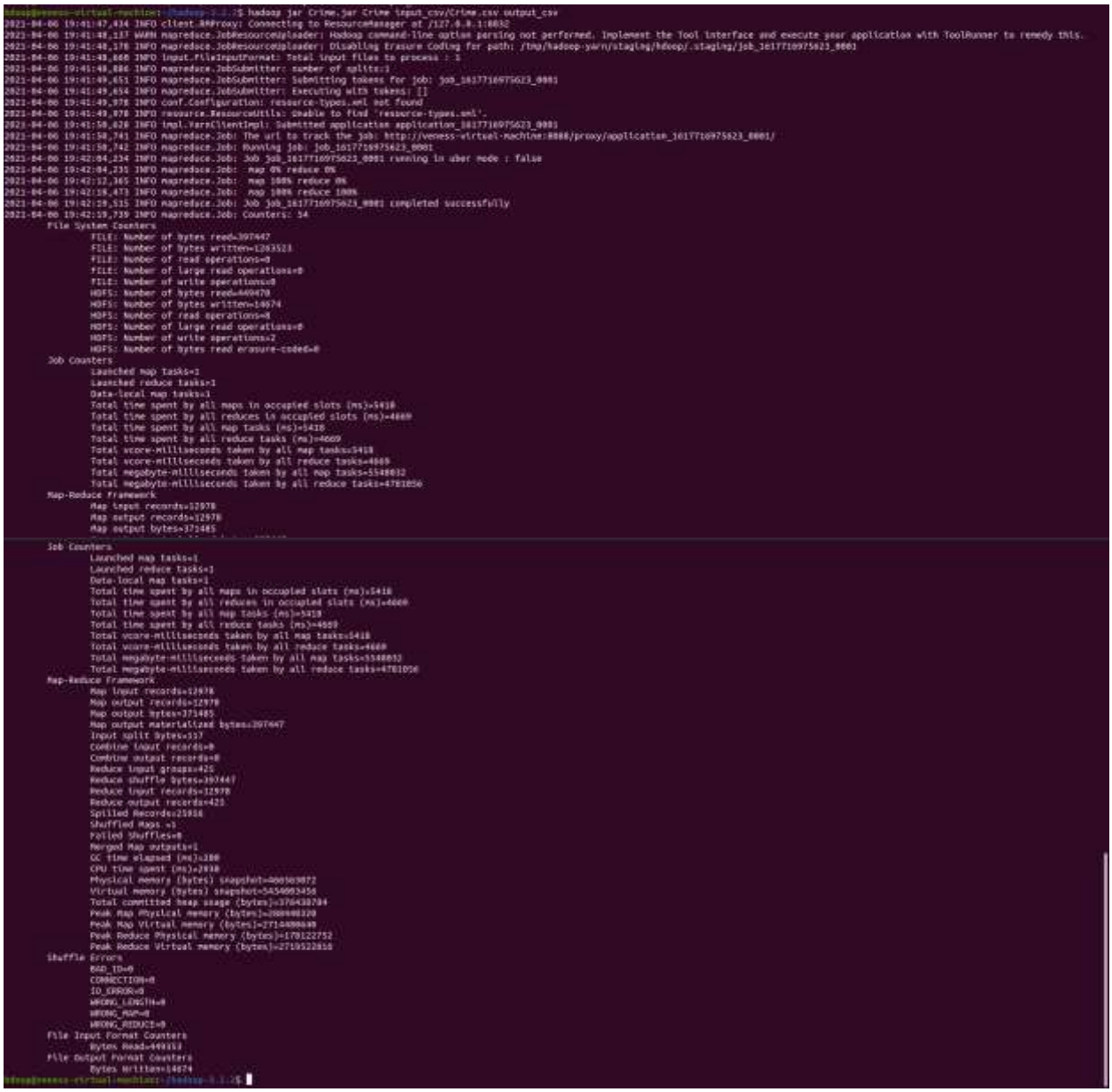


Figure 51 Hadoop - Running the Map Reduce Program

The command below is used to check what files are there in the Hadoop output directory.

```
hdoop@veness-virtual-machine:~/hadoop-3.2.2$ hdfs dfs -ls output_csv
Found 2 items
-rw-r--r--   1 hdoop supergroup          0 2021-04-06 19:42 output_csv/_SUCCESS
-rw-r--r--   1 hdoop supergroup      14674 2021-04-06 19:42 output_csv/part-r-00000
```

Figure 52 Hadoop - Listing files in output directory

Here, -cat command is used to check and retrieve the information from the output file.

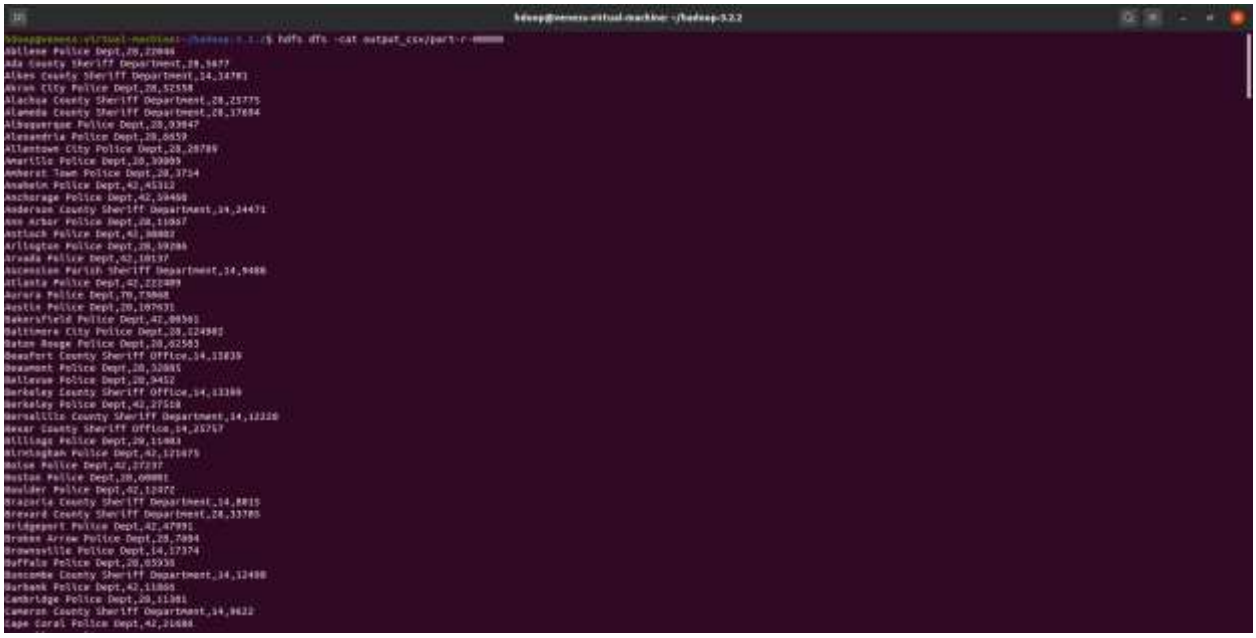


Figure 53 Hadoop - Retrieving data from output file

4. Spark

The following command is used for copying the CSV and JSON dataset into the spark home directory.

```
sakshyat@sakshyat-Lenovo-ideapad-320-15IKB:~$ cp /home/sakshyat/Downloads/crime.csv /home/sakshyat/Downloads/spark
sakshyat@sakshyat-Lenovo-ideapad-320-15IKB:~$ cp /home/sakshyat/Downloads/police-recorded-crime-data.json /home/sakshyat/Downloads/spark
sakshyat@sakshyat-Lenovo-ideapad-320-15IKB:~$
```

Figure 54 Spark - Copying the csv and json dataset

The read.csv command here reads all the data from the csv file that was copied before, and stores it in a data frame 'df' as specified. 'df.show command displays the first 20 rows of the csv dataset that was loaded.

```
>>> df = spark.read.csv('/home/sakshyat/Downloads/crime.csv')
>>> df.show()
+--------------------+----+------+
|                 _c0| _c1|   _c2|
+--------------------+----+------+
|    State_Department|Year| Value|
|Birmingham Police...|2001| 682.6|
|Birmingham Police...|2002| 692.7|
|Birmingham Police...|2003| 710.3|
|Birmingham Police...|2004| 668.9|
|Birmingham Police...|2005| 714.1|
|Birmingham Police...|2006| 608.8|
|Birmingham Police...|2007| 613.1|
|Birmingham Police...|2008| 637.7|
|Birmingham Police...|2009| 615.3|
|Birmingham Police...|2010|756.85|
|Birmingham Police...|2011| 898.4|
|Birmingham Police...|2012| 954.2|
|Birmingham Police...|2013| 774.5|
|Birmingham Police...|2014| 982.5|
|Huntsville Police...|2001| 358.9|
|Huntsville Police...|2002| 350.2|
|Huntsville Police...|2003|   349|
|Huntsville Police...|2004| 391.4|
|Huntsville Police...|2005| 369.4|
+--------------------+----+------+
only showing top 20 rows

>>>
```

Figure 55 Spark - Reading the csv dataset into the dataframe

The following command selects and displays 2 columns, '_c0' and '_c2' of the dataset.

```
>>> df.select(df['_c0'],df['_c2']).show()
+--------------------+------+
|                 _c0|   _c2|
+--------------------+------+
|    State_Department| Value|
|Birmingham Police...| 682.6|
|Birmingham Police...| 692.7|
|Birmingham Police...| 710.3|
|Birmingham Police...| 668.9|
|Birmingham Police...| 714.1|
|Birmingham Police...| 608.8|
|Birmingham Police...| 613.1|
|Birmingham Police...| 637.7|
|Birmingham Police...| 615.3|
|Birmingham Police...|756.85|
|Birmingham Police...| 898.4|
|Birmingham Police...| 954.2|
|Birmingham Police...| 774.5|
|Birmingham Police...| 982.5|
|Huntsville Police...| 358.9|
|Huntsville Police...| 350.2|
|Huntsville Police...|   349|
|Huntsville Police...| 391.4|
|Huntsville Police...| 369.4|
+--------------------+------+
only showing top 20 rows

>>> 
```

Figure 56 Spark - df.select query (csv)

The following command 'df.filter' is used to select some data from the data frame based on certain criteria, i.e data whose '_c2' is greater than 1000 is displayed in the following case.

```
>>> df.filter(df['_c2'] > 1000).show()
+--------------------+----+------+
|                 _c0| _c1|   _c2|
+--------------------+----+------+
|Little Rock Polic...|2004|1069.6|
|Little Rock Polic...|2005|1224.6|
|Little Rock Polic...|2006|1187.1|
| Oakland Police Dept|2008|1028.2|
|New Haven Police ...|2007|1003.8|
|   Miami Police Dept|2001|1158.2|
|   Miami Police Dept|2002|1150.5|
|   Miami Police Dept|2003|1060.7|
|   Miami Police Dept|2004|1018.5|
|   Miami Police Dept|2005|1029.9|
|St. Petersburg Po...|2001|1201.2|
|St. Petersburg Po...|2002|1251.6|
|St. Petersburg Po...|2003|1077.9|
|St. Petersburg Po...|2004|1134.3|
|St. Petersburg Po...|2005|1117.7|
|St. Petersburg Po...|2006|1021.8|
|St. Petersburg Po...|2007|  1082|
|   Tampa Police Dept|2001|1288.4|
|   Tampa Police Dept|2002|1170.4|
|   Tampa Police Dept|2003|1153.3|
+--------------------+----+------+
only showing top 20 rows

>>> 
```

Figure 57 Spark - df.filter query (csv)

The command below shows the count of data in each values of rows in '_c0' column.

```
>>> df.groupBy("_c0").count().show()
+--------------------+-----+
|                 _c0|count|
+--------------------+-----+
|Anchorage Police ...|   42|
|  Denver Police Dept|   42|
|Las Vegas Metropo...|   28|
|Greensboro Police...|   28|
|Seminole County S...|   28|
| Miramar Police Dept|   42|
|Milwaukee Police ...|   28|
|       El Paso County|   28|
|Wake County Sheri...|   14|
|Portland Police Dept|   28|
|  Denton Police Dept|   28|
|Santa Rosa Police...|   42|
|Lexington-Fayette...|   28|
|Long Beach Police...|   42|
|Vancouver Police ...|   28|
|Sarasota County S...|   28|
|Thornton Police Dept|   42|
|Newport News Poli...|   28|
|Sterling Heights ...|   14|
|Harford County Sh...|   14|
+--------------------+-----+
only showing top 20 rows

>>>
```

Figure 58 Spark - df.groupBy query (csv)

In order to use the SQL query, the data frame is registered as temporary SQL view, i.e. 'PoliceRecord' in the following case. This view can be later on used as a table name while executing the SQL queries.

```
>>> df.createOrReplaceTempView("PoliceRecord")
>>>
```

Figure 59 Spark - Creating Temporary SQL View (csv)

The SQL ‘SELECT’ query is used here for retrieving the information of the data, using the temporary SQL view that was created before. Here, the following is the SQL command for displaying the records where ‘_c1’ is 2012.

```
>>> sqlDF = spark.sql("SELECT _c0, _c1, _c2 FROM PoliceRecord WHERE _c1 = 2012")
>>> sqlDF.show()
+--------------------+----+-----+
|                 _c0| _c1|  _c2|
+--------------------+----+-----+
|Birmingham Police...|2012|954.2|
|Huntsville Police...|2012|628.2|
|  Mobile Police Dept|2012|308.9|
|Montgomery Police...|2012|135.4|
|Anchorage Police ...|2012|559.3|
|Chandler Police Dept|2012|169.3|
| Gilbert Police Dept|2012| 56.9|
|Glendale Police Dept|2012|285.8|
|    Mesa Police Dept|2012|254.5|
|  Peoria Police Dept|2012|117.5|
| Phoenix Police Dept|2012|354.3|
|Scottsdale Police...|2012| 75.6|
|Surprise Police Dept|2012| 73.7|
|   Tempe Police Dept|2012|343.2|
|  Tucson Police Dept|2012|435.3|
|Little Rock Polic...|2012|  811|
| Anaheim Police Dept|2012|215.4|
| Antioch Police Dept|2012|625.7|
|Bakersfield Polic...|2012|320.8|
|Berkeley Police Dept|2012| 93.9|
+--------------------+----+-----+
only showing top 20 rows

>>>
```

Figure 60 Spark - Executing SQL query (csv)

Similar as csv file before, the following ‘read.json’ command is used for reading and loading the json dataset that was copied earlier. Here, ‘df.show’ displays the first 20 rows of the json dataset.

```
>>> df = spark.read.json('/home/sakshyat/Downloads/police-recorded-crime-data.json')
>>> df.show()
+-----+--------------------+--------------------+-----+----------------+--------------+
|Count|          Crime_Type|        Data_Measure|Month|Policing_District|Calendar_Year|
+-----+--------------------+--------------------+-----+----------------+--------------+
|  738|Violence with inj...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1498|Violence without ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  133|     Sexual offences|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  158|             Robbery|Police Recorded C...|  Apr| Northern Ireland|          2001|
|    0|Theft - burglary ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|    0|Theft - burglary ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  750|Theft - domestic ...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  670|Theft - non-domes...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  101|Theft from the pe...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1829|Theft - vehicle o...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   75|       Bicycle theft|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  442| Theft - shoplifting|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 1231|All other theft o...|Police Recorded C...|  Apr| Northern Ireland|          2001|
| 3309|     Criminal damage|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   22|Trafficking of drugs|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   83| Possession of drugs|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   36|Possession of wea...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|   38|Public order offe...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|  244|Miscellaneous cri...|Police Recorded C...|  Apr| Northern Ireland|          2001|
|11357|Total police reco...|Police Recorded C...|  Apr| Northern Ireland|          2001|
+-----+--------------------+--------------------+-----+----------------+--------------+
only showing top 20 rows

>>>
```

Figure 61 Spark - Reading the JSON dataset into the dataframe

The following command is used to know how many records are there in the given dataset.

```
>>> df.count()
159840
>>>
```

Figure 62 Spark - df.count query

The query below is equivalent to the normal ‘SELECT’ query in SQL, which displays the data from the columns ‘Policing_District’, ‘Crime_Type’ and ‘Count’.

```
>>> df.select(df['Policing_District'],df['Crime_Type'],df['Count']).show()
+-----------------+--------------------+-----+
|Policing_District|          Crime_Type|Count|
+-----------------+--------------------+-----+
| Northern Ireland|Violence with inj...|  738|
| Northern Ireland|Violence without ...| 1498|
| Northern Ireland|     Sexual offences|  133|
| Northern Ireland|             Robbery|  158|
| Northern Ireland|Theft - burglary ...|    0|
| Northern Ireland|Theft - burglary ...|    0|
| Northern Ireland|Theft - domestic ...|  750|
| Northern Ireland|Theft - non-domes...|  670|
| Northern Ireland|Theft from the pe...|  101|
| Northern Ireland|Theft - vehicle o...| 1829|
| Northern Ireland|       Bicycle theft|   75|
| Northern Ireland| Theft - shoplifting|  442|
| Northern Ireland|All other theft o...| 1231|
| Northern Ireland|     Criminal damage| 3309|
| Northern Ireland|Trafficking of drugs|   22|
| Northern Ireland| Possession of drugs|   83|
| Northern Ireland|Possession of wea...|   36|
| Northern Ireland|Public order offe...|   38|
| Northern Ireland|Miscellaneous cri...|  244|
| Northern Ireland|Total police reco...|11357|
+-----------------+--------------------+-----+
only showing top 20 rows

>>>
```

Figure 63 Spark - df.select query (JSON)

The following command displays the number of records in each “Crime_Type’

```
>>> df.groupBy('Crime_Type').count().show()
+--------------------+-----+
|          Crime_Type|count|
+--------------------+-----+
| Theft - shoplifting| 7992|
|Public order offe...| 7992|
|       Bicycle theft| 7992|
|Total police reco...| 7992|
|Theft - non-domes...| 7992|
|All other theft o...| 7992|
|             Robbery| 7992|
|Violence with inj...| 7992|
|     Sexual offences| 7992|
|Possession of wea...| 7992|
|Theft - burglary ...| 7992|
|Theft from the pe...| 7992|
|Violence without ...| 7992|
|Theft - burglary ...| 7992|
|Miscellaneous cri...| 7992|
|Trafficking of drugs| 7992|
| Possession of drugs| 7992|
|Theft - domestic ...| 7992|
|Theft - vehicle o...| 7992|
|     Criminal damage| 7992|
+--------------------+-----+

>>>
```

Figure 64 Spark - df.groupBy query (JSON)

Again, similar as before, a temporary view named “Crime” is created in order to run the SQL query to retrieve information in the given json dataset.

```
>>> df.createOrReplaceTempView("Crime")
>>>
```

Figure 65 Spaek - Creating Temporary SQL View (JSON)

The following is the SQL query executed on spark in order to retrieve the data based on the criteria as specified. Here, the query below displays the total number of ‘Crime_Type’ in different months. The ‘ORDER BY” statement here, displays the data in ascending order of ‘Crime_type’.

```
>>> sqlDF = spark.sql("SELECT Crime_Type, Month, COUNT(Crime_Type) FROM Crime GROUP BY  Month, Crime_Type ORDER BY Crime_Type")
>>> sqlDF.show()
+--------------------+-----+-----------------+
|          Crime_Type|Month|count(Crime_Type)|
+--------------------+-----+-----------------+
|All other theft o...|  Nov|              648|
|All other theft o...|  Feb|              648|
|All other theft o...|  Jul|              684|
|All other theft o...|  Sep|              684|
|All other theft o...|  Jun|              684|
|All other theft o...|  Jan|              648|
|All other theft o...|  Mar|              648|
|All other theft o...|  Aug|              684|
|All other theft o...|  Dec|              648|
|All other theft o...|  Oct|              648|
|All other theft o...|  Apr|              684|
|All other theft o...|  May|              684|
|        Bicycle theft|  May|              684|
|        Bicycle theft|  Sep|              684|
|        Bicycle theft|  Jan|              648|
|        Bicycle theft|  Feb|              648|
|        Bicycle theft|  Aug|              684|
|        Bicycle theft|  Oct|              648|
|        Bicycle theft|  Dec|              648|
|        Bicycle theft|  Mar|              648|
+--------------------+-----+-----------------+
only showing top 20 rows

>>>
```

Figure 66 Spark - Executing SQL query (JSON)

7.4 Visualization of Data

The following pie chart shows the total number of overall crime rates in the US. As it can be seen below, year 2002 has the highest recorded crime and year 2014 recorded the lowest crime rate. With this, the probability of occurrence of crime in the further years is likely to decrease.

Figure 67 Pie Chart (Overall US crime rates)

The following line graph demonstrate the rate of crime in the Brimingham Police Department. Here as it can be seen the crime rate is gradually increasing from year 2002 to 2014. From this, it can be predicted that the crime rate in this department is likely to increase more in the further years.

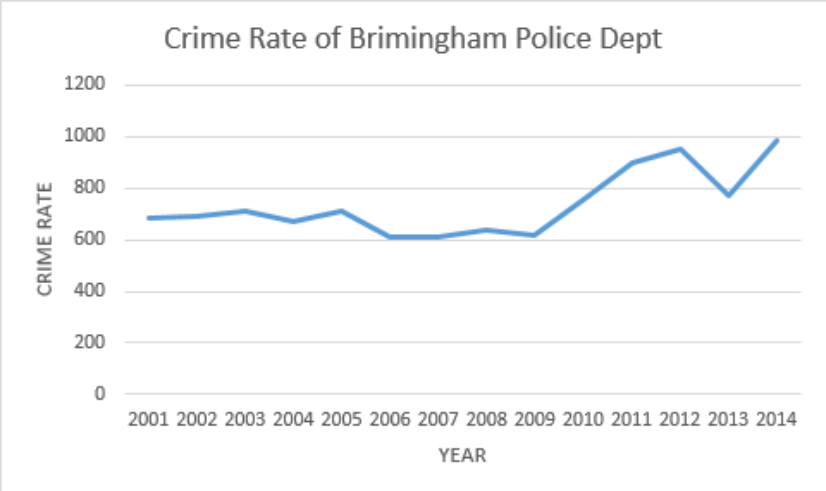


Figure 68 Line Graph - Crime rate in Brimingham Police Dept

The following graph shows the rate of crime in Phoenix Police Department. As it can be seen below, the crime rate is not consistent and is gradually increasing and decreasing in each years from 2002-2014. However, in comparison to the first year 2002, the crime rate is comparatively low in the year 2014. This draws to conclusion that the chances of crime rate is likely to decrease in further years as well.

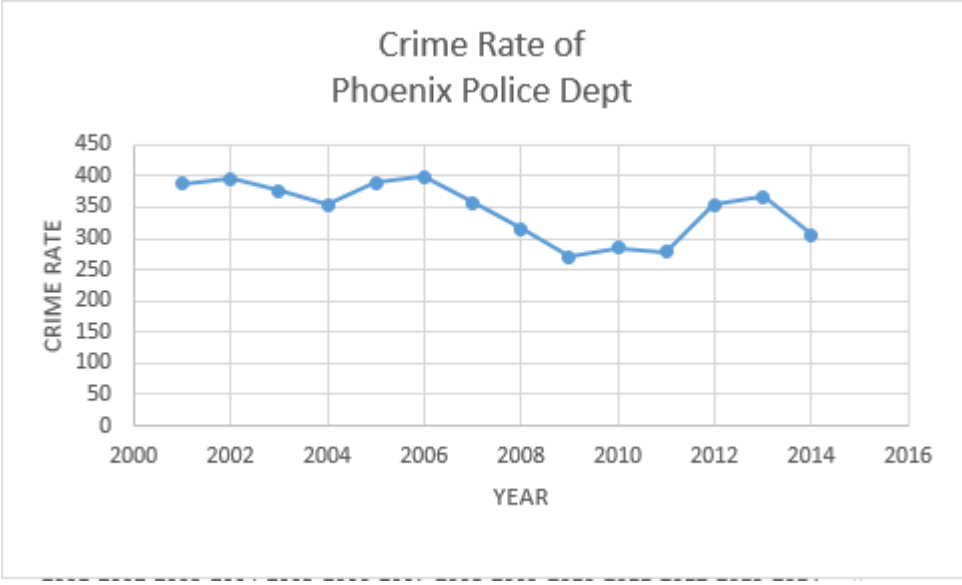


Figure 69 Line Graph - Crime Rate of Phoenix Police Dept

8 Division of Work

Team Member	Contribution
Sakshyat Sharma	• CSV dataset • Cleaning the data • Oracle analysis • Spark analysis • Comparison table (Advantages) • Data visualization (Overall Data charts)
Venisha Pandey	• Introduction to big data • JSON dataset • MongoDB analysis • Hadoop (Map Reduce) • Comparison Table (Disadvantages) • Visualizations (Individual Data charts)

9 References

- acodez, 2019. *ALL ABOUT MONGODB: THE NOSQL DATABASE.* [Online]
  Available at: https://acodez.in/mongodb-nosql-database/
  [Accessed 24 April 2021].
- EDUCBA, 2020. *What is Oracle?.* [Online]
  Available at: https://www.educba.com/what-is-oracle/
  [Accessed 24 April 2021].
- J.Anuradha & Ishwarappa, 2015. A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. *Procedia Computer Science,* Volume 48, pp. 319-324.
- mindsmapped, 2015. *Hadoop Advantages and Disadvantages.* [Online]
  Available at: https://www.mindsmapped.com/hadoop-advantages-and-disadvantages/
  [Accessed 24 April 2021].
- Nemschoff, M., 2013. *Big data: 5 major advantages of Hadoop.* [Online]
  Available at: https://www.itproportal.com/2013/12/20/big-data-5-major-advantages-of-hadoop/
  [Accessed 24 April 2021].
- Reference, 2020. *What Are Some Advantages and Disadvantages of Oracle Database?.* [Online]
  Available at: https://www.reference.com/world-view/advantages-disadvantages-oracle-database-dbec0904f3b40425
  [Accessed 24 April 2021].
- tutorialspoint, 2021. *MongoDB - Advantages.* [Online]
  Available at: https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm
  [Accessed 24 April 2021].