Big Data Analysis of Chicago Crime Dataset

GitHub

Manish Tandel (Voice of the team)
Chirag Bhansali
Bingxin Xu

04/30/2020

—

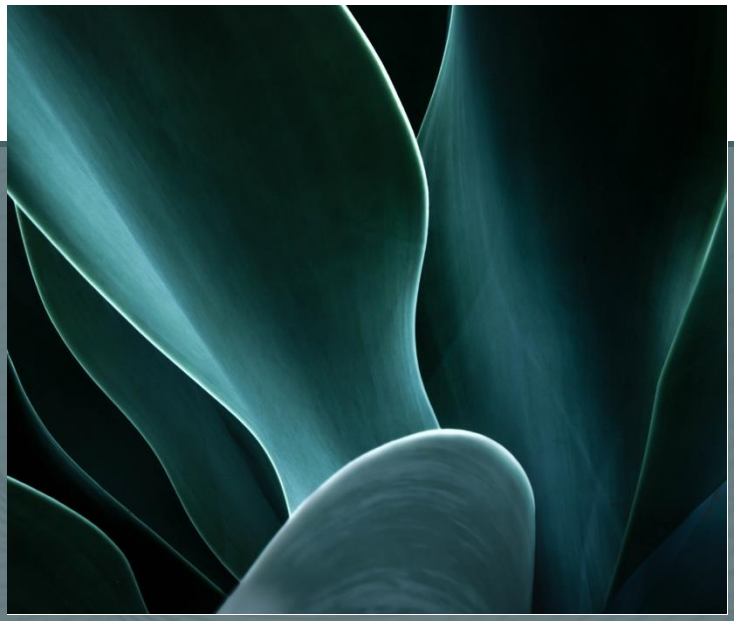
Big Data Technologies

—

Prof. Joseph Rosen

ABSTRACT

In this project, we used Big Data Technologies to analyze the Chicago Crime Dataset and derive useful insights from it. We address 9 questions in the study. The answers to these questions and visualization of the result provide useful information to the Chicago Police Department. We use data mining techniques like decision tree and K-means clustering to achieve our aim.



Introduction

The Chicago Police Department's Bureau of Records has been keeping records of all the crimes in Chicago since the beginning of the 20th century [1][DS]. This presents us a great opportunity to get our hands on a massive real-world data. The overall crime rate in Chicago, especially the violent crime rate, is higher than the US average [2]. Even though, the nation's crime rates remained near historic lows, in 2016, Chicago was responsible for nearly half of 2016's increase in homicides in the US [3].

We came up with 9 questions and classified them in two categories-Exploratory Data Analysis (EDA) and analysis using data mining techniques. We realized answers to EDA questions can be found using either Pig scripts or Hive queries. We also realized we need to apply data mining techniques to solve more advanced data mining questions. We used SparkR for data mining questions.

Research Questions

Exploratory Data Analysis Questions

1.What are the most occurring crimes in the city?
2.How many crimes are being committed at a specific location? (e.g. street, residence)
3.Which crimes are being committed at a specific time of the day?
4.Analysis of crimes with respect to locations and time of the day
5.What are the most occurring crimes in different seasons?
6. Analysis of crimes with respect to locations and seasons
7.Analysis of a particular crime type over the years (Theft)

Data Mining Questions

1.What kind of a crime random person is likely to face at a given location (e.g. street, residence), area (district), time of the day (e.g. morning, evening)?
2.What are the most occurring crime in every part of the city?

Related work:

Ingilevich et al. ’s (2018) [6] research intended for providing insight into different statistical tools that can be implemented in the approximation of crime numbers. Linear regression, logistic regression, and gradient boosting (GB) where the predictive model's types that were used. Data of Saint-Petersburg used in this research was provided by the Ministry of Internal Affairs of the Russian Federation and was containing information about the Date a crime occurred, the Coordinates, and the Description. The data covered the timeframe of 1/1/2014 to 2/28/2017. Characteristics including GDP, population density, unemployment, and homeless population numbers, as well as, other spatial criminal patterns were considered as the external factors responsible for the occurrence of crime in a location. Afterward, street crimes were prioritized and categorized to banditry, massacre, and robbery. The results of the research suggested robbery was the most commonly occurring crime type with some 142,452 occurrences. Then, through clustering, the sub-features of each type were examined. The clusters included the number of populations, police stations, schools, malls, churches, alcohol shops, numbers of buildings, and bars. It is explained that the feature selection technique is the solution of creating strong predictive models and Gradient Boosting was proven to be the most effective in preventing crime rate predictions.

ToppiReddy et al. (2018) [7] used R and libraries including RgoogleMaps and googleVis to present crime statistics visually and consequently predict crime occurrence using Machine Learning. The result of this research was the visualization of exact crime locations on google maps, the visualization of data based on the crime type, the visualization of crime occurrence frequency, and the graphical representations and bar charts of crime frequency. ToppiReddy et al. ’s (2018) suggested framework implements Data Mining techniques and applies Rational Choice Theory, Routine Activity Theory, the K-Nearest Neighbour method m, and the Naïve Bayes theorem to conduct a predictive analysis.
Belesiotis et al. (2018) [8] examine crime hotspots in order to provide insight that could help security agencies and citizens. The research takes advantage of data-driven methodology and implements data mining from online sources. The experimentation process of the authors started by identifying valuable data sources. Then they retrieved the data and used it to produce statistics. Afterward, they trained prediction algorithms with data considered appropriate. Finally, they tested the model used to test its performance. Data used in this research was extracted from the Greater London geographic region after it was separated geographically to 4831 zones (Belesiotis et al. 2018). The researchers used Web APIs to mine data from 6 sources that include features related to demographics, points of interest, transportation, land use, and photos (Belesiotis et al. 2018). It was proved that crime prediction can be more successful thanks to data-driven research and can help both citizens be vigilant of specific areas and law enforcement agencies are more efficient (Belesiotis et al. 2018).

Proposed framework

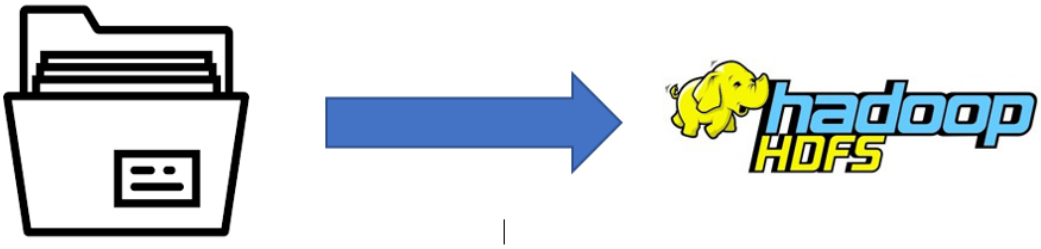
Storing data:

HDFS:

HDFS is the file system component of Hadoop. While the interface to HDFS is patterned after the UNIX file system, faithfulness to standards was sacrificed in favor of improved performance for the applications at hand. HDFS stores file system metadata and application data separately. As in other distributed file systems, like PVFS, Lustre and GFS, HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols [9].

HDFS is a self-healing, distributed file system that provides reliable, scalable and fault tolerant data storage on commodity hardware. It works closely with MapReduce by distributing storage and computation across large clusters by combining storage resources that can scale depending upon requests and queries while remaining inexpensive and in budget [18]. HDFS accepts data in any format like text, images, videos regardless of architecture and automatically optimizes for high bandwidth streaming. The foremost advantage of HDFS is fault tolerance. By provisioning fast data transfer between the nodes and enabling Hadoop to continue to provide service even in event of node failures decreases the risk of catastrophic failure. HDFS is also capable of providing scale-out storage solution for Hadoop [9].

Dataset is fetched from City of Chicago | Data Portal. url: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2 and stored in HDFS, moveFromLocal command is used to copy all the dataset files from the local file system to HDFS



For our study we needed to access data from Hive, Pig and SparkR. Hence, using HDFS for storing data for our study made sense as all the team members had to access data from different platforms at the same time.

Analysis of Data

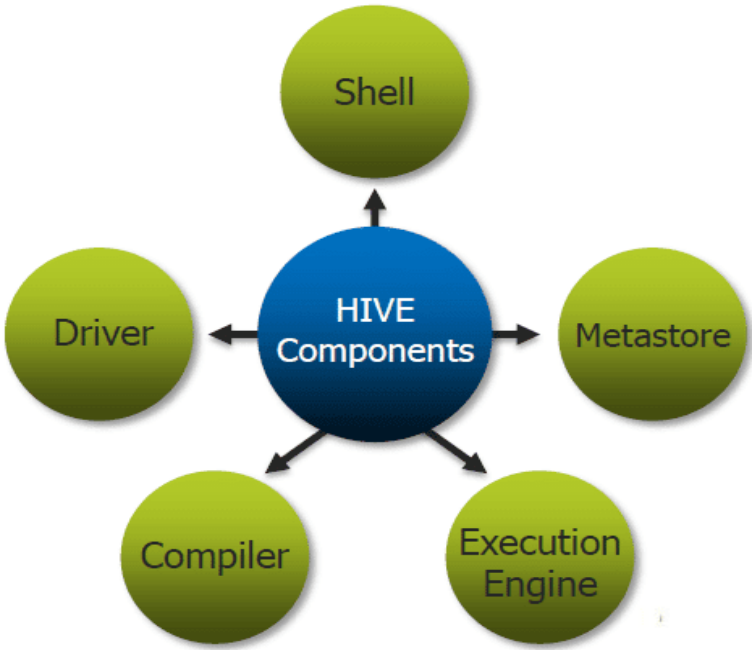
Hive:

Apache Hive is an open-source relational database system for analytic big-data workloads. Apache Hive is a data warehouse system for Apache Hadoop. It has been widely used in organizations to manage and process large volumes of data, such as eBay, Facebook, LinkedIn, Spotify, Taobao, Tencent, and Yahoo! [10]. When Hive was first introduced over 10 years ago, the motivation of the authors was to expose a SQL-like interface on top of Hadoop MapReduce to abstract the users from dealing with low level implementation details for their parallel batch processing jobs.

As Hadoop became a ubiquitous platform for inexpensive data storage with HDFS, developers focused on increasing the range of workloads that could be executed efficiently within the platform. YARN, a resource management framework for Hadoop, was introduced, and shortly afterwards, data processing engines (other than MapReduce) such as Spark or Flink were enabled to run on Hadoop directly by supporting YARN.

Initially, Hive only had support to insert and drop full partitions from a table. Although the lack of row level operations was acceptable for ETL workloads, as Hive evolved to support many traditional data warehousing workloads, there was an increasing requirement for full DML support and ACID transactions. No, Hive includes support to execute INSERT, UPDATE, DELETE, and MERGE statements. It provides ACID guarantees via Snapshot Isolation on read and well-defined semantics in case of failure using a transaction manager built on top of the Hive Metastore [11].

We decided to go with Hive because Hive provides the basic SQL operations like filtering rows, selecting certain rows, joins, aggregation using group by. Hive enforces schema on read which makes for a fast-initial load since the data does not have to be read, parsed, and serialized to disk in the database's internal format.

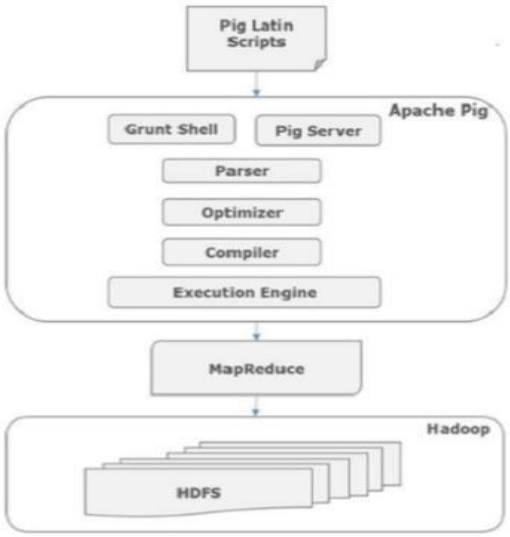


Pig:

At a growing number of organizations, innovation revolves around the collection and analysis of enormous data sets such as web crawls, search logs, and click streams. Internet companies such as Amazon, Google, Microsoft, and Yahoo! are prime examples. Analysis of this data constitutes the innermost loop of the product improvement cycle. For example, the engineers who develop search engine ranking algorithms spend much of their time analyzing search logs looking for exploitable trends. The sheer size of these data sets dictates that it be stored and processed on highly parallel systems, such as shared nothing clusters. Parallel database products, e.g., Teradata, Oracle RAC, Netezza, offer a solution by providing a simple SQL query interface and hiding the complexity of the physical cluster. These products, however, can be prohibitively expensive at web scale. Besides, they wrench programmers away from their preferred method of analyzing data, namely writing imperative scripts or code, toward writing declarative queries in SQL, which they often find unnatural, overly restrictive [12].

As evidence of the above, programmers have been flocking to the more procedural map-reduce [13] programming model. Unfortunately, the map-reduce model has its own set of limitations [14]. Its one-input, two-stage data flow is extremely rigid. To perform tasks having a different data flow, e.g., joins or n stages, inelegant workarounds must be devised. Also, custom code must be written for even the most common operations, e.g., projection and filtering. These factors lead to code that is difficult to reuse and maintain, and in which the semantics of the analysis task are obscured. Moreover, the opaque nature of the map and reduce functions impedes the ability of the system to perform optimizations. We have developed a new language called Pig Latin that combines the best of both worlds: high-level declarative querying in the spirit of SQL, and low-level, procedural programming `a la map-reduce.

We chose Pig because it is a simple SQL-like scripting language that parses, optimizes, and automatically executes Pig Latin scripts as a series of MapReduce jobs on a Hadoop cluster. Pig is much easier and faster to write than MapReduce, and it's easier to decipher as well. Pig Latin is a data flow language- it allows users to describe how data from one or more inputs should be read, transformed, and after that stored to one or more outputs in parallel. We had to keep in mind although Pig Latin programs supply an explicit sequence of operations, it is not necessary that the operations be executed in that order [12].



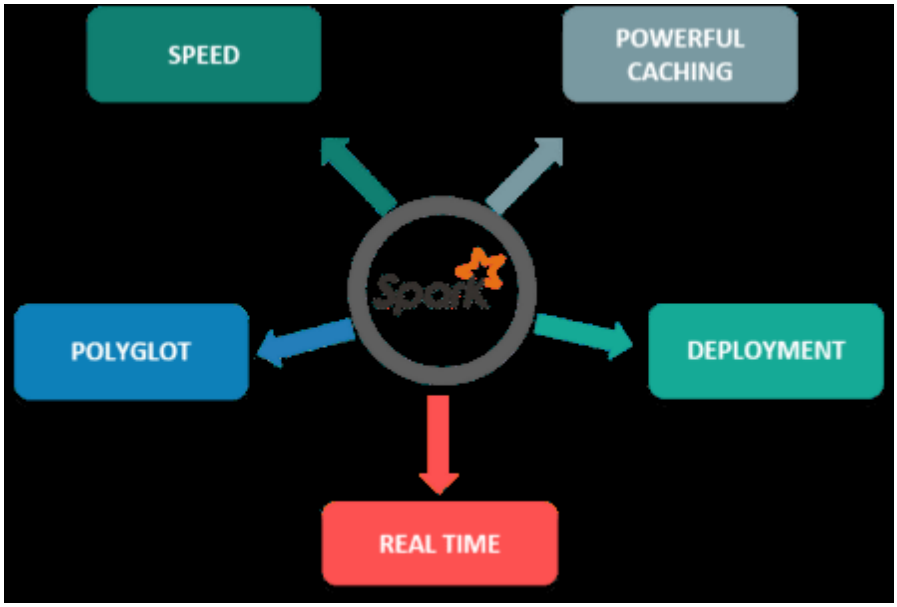
SparkR:

Recent trends in big data analytics indicate the growing need for interactive analysis of large datasets. In response to this trend, several academic and commercial systems have been developed to support such use cases. However, data science surveys [15] show that in addition to relational query processing, data scientists often use tools like R to perform more advanced analysis on data. R is particularly popular as it provides support for structured data processing using data frames and includes a number of packages for statistical analysis and visualization.

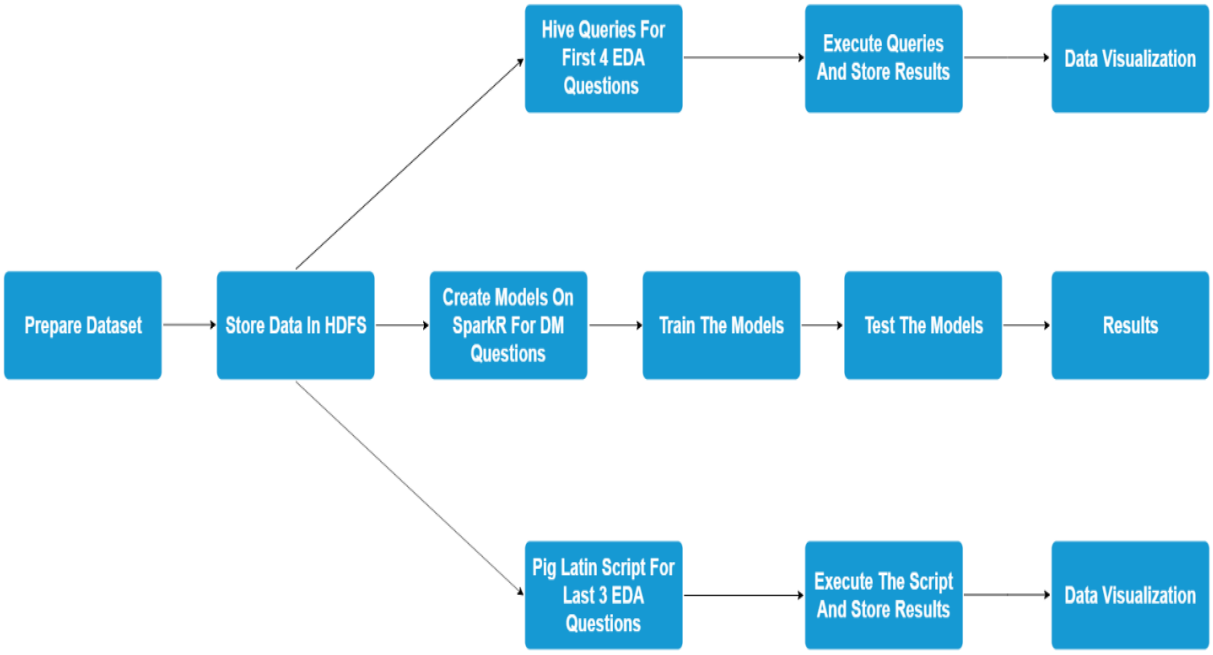
However, data analysis using R is limited by the amount of memory available on a single machine and further as R is single threaded it is often impractical to use R on large datasets. Prior research has addressed some of these limitations through better I/O support, integration with Hadoop and by designing distributed R runtimes that can be integrated with DBMS engines.

SparkR is built as an R package and requires no changes to R. The central component of SparkR is a distributed data frame that enables structured data processing with a syntax familiar to R users. To improve performance over large datasets, SparkR performs lazy evaluation on data frame operations and uses Spark's relational query optimizer to optimize execution. The Spark project contains libraries for running SQL queries, distributed machine learning, graph analytics and SparkR can reuse well-tested, distributed implementations for these domains. Further, Spark SQL's data sources API provides support for reading input from a variety of systems including HDFS, HBase, Cassandra and a number of formats like JSON, Parquet, etc. Integrating with the data source API enables R users to directly process data sets from any of these data sources [16].

We used SparkR as Spark MLlib contains several machine learning algorithms including K-Means clustering and decision tress.

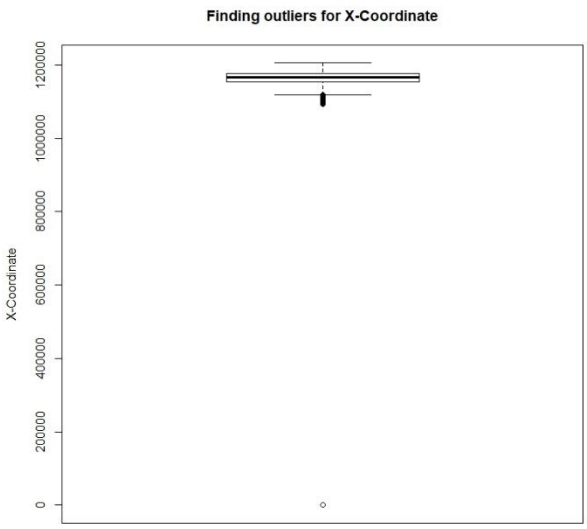


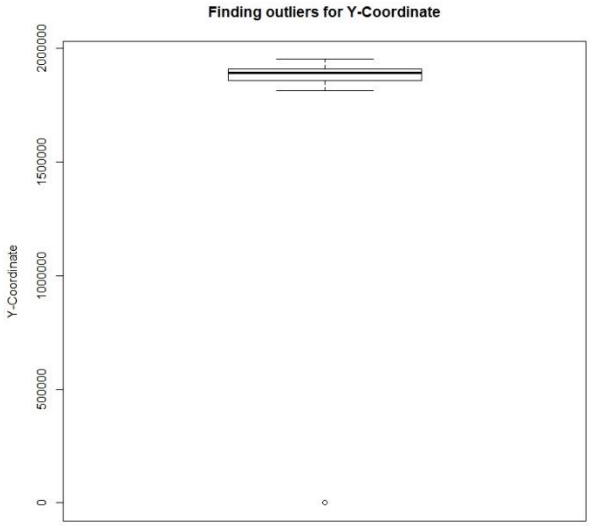
The Flow

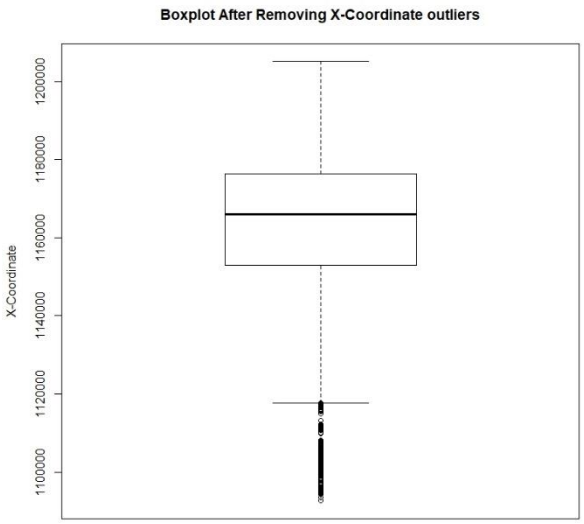

Prepare Dataset
Store Data In HDFS
Hive Queries For First 4 EDA Questions
Execute Queries And Store Results
Data Visualization
Create Models On SparkR For DM Questions
Train The Models
Test The Models
Results
Pig Latin Script For Last 3 EDA Questions
Execute The Script And Store Results
Data Visualization

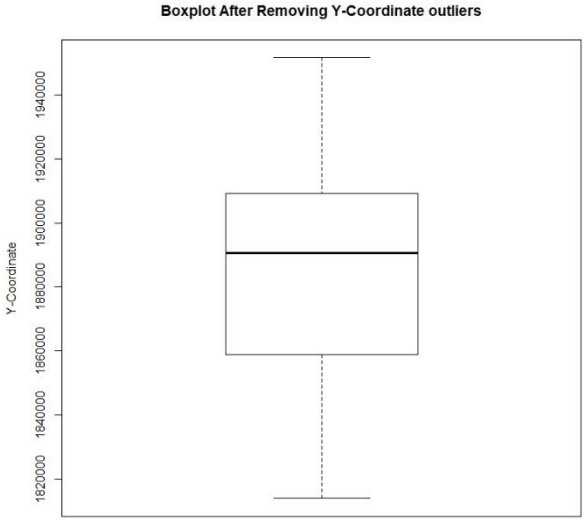
Prepare Dataset

The dataset had 7 million rows but around 60,000 records had missing values. We removed the records with missing values. For K-Means clustering algorithm, attributes X-Coordinate and Y-Coordinate were crucial. K-Means clustering assigns all the data points give as an input. In other words, K-Means algorithm doesn't handle the noise points unlike DBSCAN clustering does. Hence, to find out the noisy data in our dataset we plotted the boxplots for X-Coordinate and Y-Coordinate. We found that for ????? records X-Coordinate and Y-Coordinate values were 0. We removed such records.









Few of the questions we addressed in this study involved time of the day. In the dataset, we had a date field in MM-DD-YYYY HH:TT AM/PM format so we extracted the hour and converted in the 24-hour format and added that as a new column. For example, if the date value for a record is 10/20/2019 04:21 PM then Hour of The Day field will be 16. Also, couple of questions we addressed dealt with the seasons in Chicago. Hence, we derived season in which every crime had happened based on, again, the date field- this time we extracted the month from the date.

K-Means clustering

K-Means is a typical clustering algorithm in data mining, and which is widely used for clustering large set of data. The algorithm consists of two separate phases. The first phase selects k centers randomly, where the value k is fixed in advance. The next phase is to take each data object to the nearest center. Euclidean distance is generally considered to determine the distance between each data object and the cluster centers. When all the data objects are included in some clusters, the first step is completed, and an early grouping is done. Recalculating the average of the early formed clusters. This iterative process continues repeatedly until the criterion function becomes the minimum [17].

Number of clusters: 60

We used the elbow method to decide how many clusters we should have.

Input:

We created a data frame in R which had records containing only three fields- X-Coordinate, Y-Coordinate and Primary Type. Our aim is to find out most occurring crime (Primary Type) for every cluster.

Attributes used for clustering: X-Coordinates and Y-Coordinates.

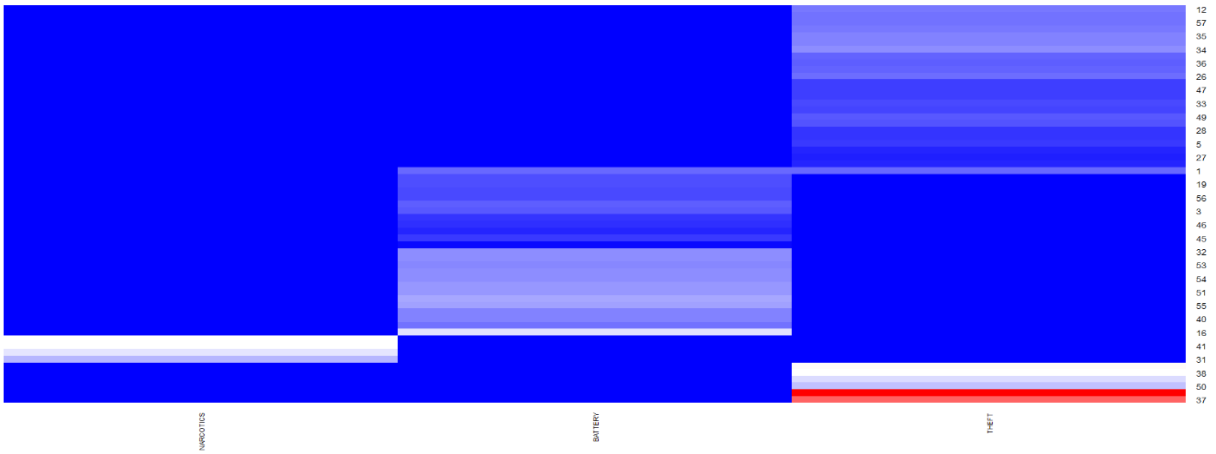
The dataset has millions of records and the crimes have occurred in all parts of the city. When we plot all the data points based on their X-Coordinates and Y-Coordinates, it seems like the map of Chicago is displayed on the screen. The K-Means algorithm creates 60 clusters out of all the points hence the city is divided in 60 parts.

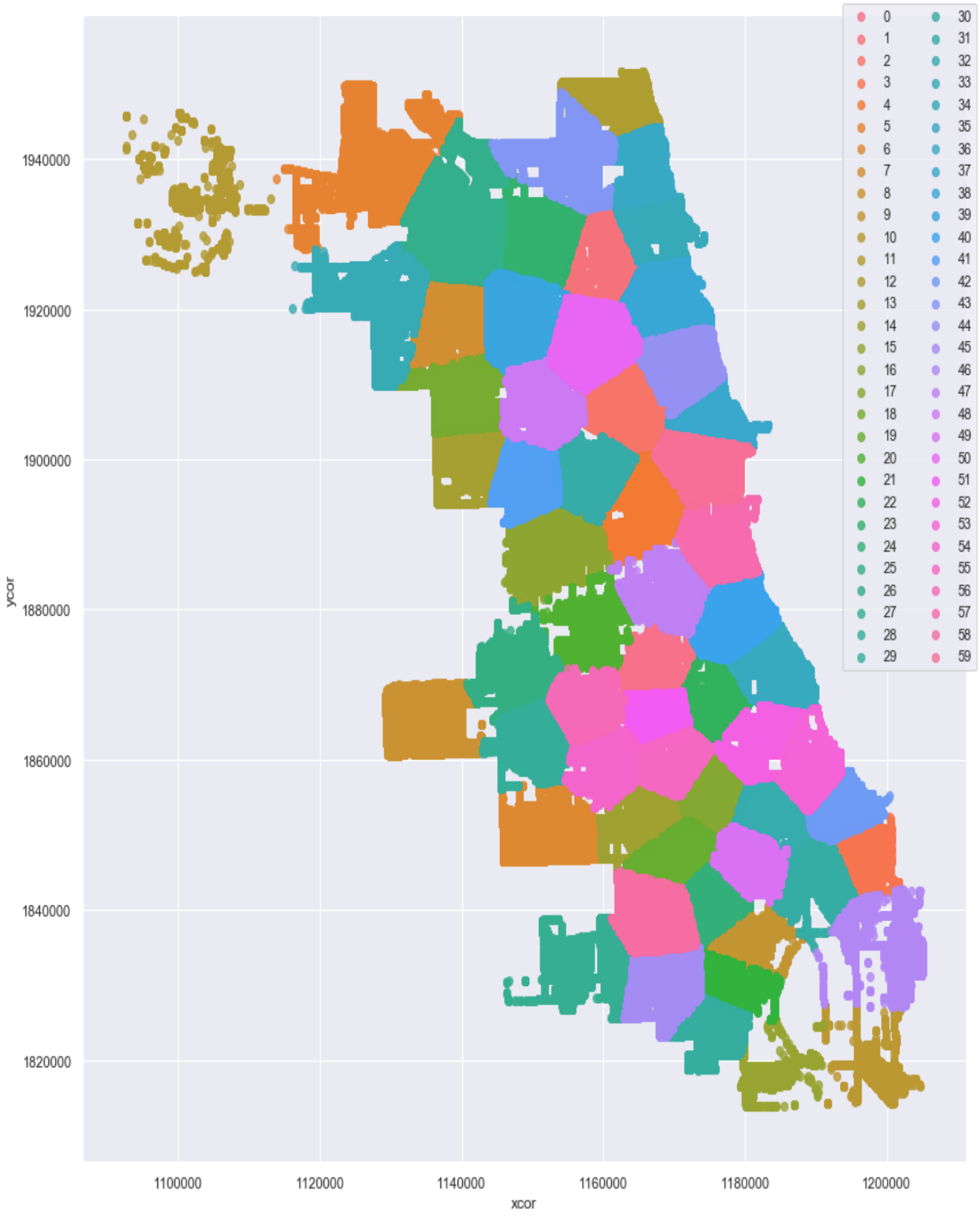
Outcome:

K-Means algorithm divides the city in 60 clusters and we then find out most occurring crime in every cluster.

```
>>> y=spark.sql('select cluster,primary,cnt from tt where (cluster,cnt) in (select cluster,max(cnt) from tt group by cluster) order by cluster')
>>>
>>> y.show(60)
+-------+---------+------+
|cluster|  primary|   cnt|
+-------+---------+------+
|      0|  BATTERY| 22835|
|      1|    THEFT| 22531|
|      2|    THEFT| 56578|
|      3|  BATTERY| 18667|
|      4|    THEFT| 28086|
|      5|    THEFT| 12353|
|      6|    THEFT| 17690|
|      7|    THEFT| 20776|
|      8|    THEFT|  8620|
|      9|  BATTERY| 16748|
|     10|  BATTERY|  2477|
|     11|    THEFT|  7767|
|     12|    THEFT| 25137|
|     13|NARCOTICS| 55267|
|     14|  BATTERY| 24412|
|     15|  BATTERY|  8614|
|     16|  BATTERY| 47529|
|     17|  BATTERY| 29594|
|     18|  BATTERY| 35580|
|     19|  BATTERY| 16725|
|     20|    THEFT| 13771|
|     21|  BATTERY| 19736|
|     22|  BATTERY| 27426|
|     23|    THEFT| 25078|
|     24|  BATTERY| 11138|
|     25|    THEFT| 15163|
|     26|    THEFT| 23104|
|     27|    THEFT|  7174|
|     28|    THEFT| 11175|
|     29|  BATTERY| 16018|
|     30|    THEFT| 13882|
|     31|NARCOTICS| 38490|
|     32|  BATTERY| 29885|
|     33|    THEFT| 16146|
|     34|    THEFT| 29625|
|     35|    THEFT| 28223|
|     36|    THEFT| 20618|
|     37|    THEFT| 88356|
|     38|    THEFT| 55050|
|     39|    THEFT| 26175|
|     40|  BATTERY| 27309|
|     41|NARCOTICS| 53688|
|     42|  BATTERY| 30499|
|     43|    THEFT| 21245|
|     44|    THEFT| 46284|
|     45|  BATTERY| 12703|
|     46|  BATTERY| 10087|
|     47|    THEFT| 13708|
|     48|NARCOTICS| 48416|
|     49|    THEFT| 18579|
|     50|    THEFT| 40637|
|     51|  BATTERY| 31798|
|     52|  BATTERY| 32676|
|     53|  BATTERY| 29120|
|     54|  BATTERY| 30182|
|     55|  BATTERY| 34006|
|     56|  BATTERY| 15985|
|     57|    THEFT| 24588|
|     58|    THEFT| 11062|
|     59|    THEFT|109134|
+-------+---------+------+

>>>
```

Most Occuring crimes in each cluster




0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
1940000
1920000
1900000
1880000
1860000
1840000
1820000
ycor
1100000
1120000
1140000
1160000
1180000
1200000
xcor

Decision Trees:

The decision tree is a data mining technique for solving classification and prediction problems. Decision trees are a simple recursive structure for expressing a sequential classification process in which a case, described by a set of attributes, is assigned to one of a disjoint set of classes. Decision trees consist of nodes and leaves. Each node in the tree involves testing a particular attribute and each leaf of the tree denotes a class. Usually, the test compares an attribute value with a constant. Leaf nodes give a classification that applies to all instances that reach the leaf, or a set of classifications, or a probability distribution over all possible classifications. To classify an unknown instance, it is routed down the tree according to the values of the attributes tested in successive nodes, and when a leaf is reached, the instance is classified according to the class assigned to the leaf. If the attribute that is tested at a node is a nominal one, the number of children is usually the number of possible values of the attribute. The tree complexity is measured by one of the following metrics: the total number of nodes, total number of leaves, tree depth and number of attributes used [18].

The dataset had almost 50% of the records which belong to either class Theft or Battery. To mitigate the class imbalance problem, we used SMOTE technique. Also, we only considered records of three types- Theft, Battery, Narcotics.

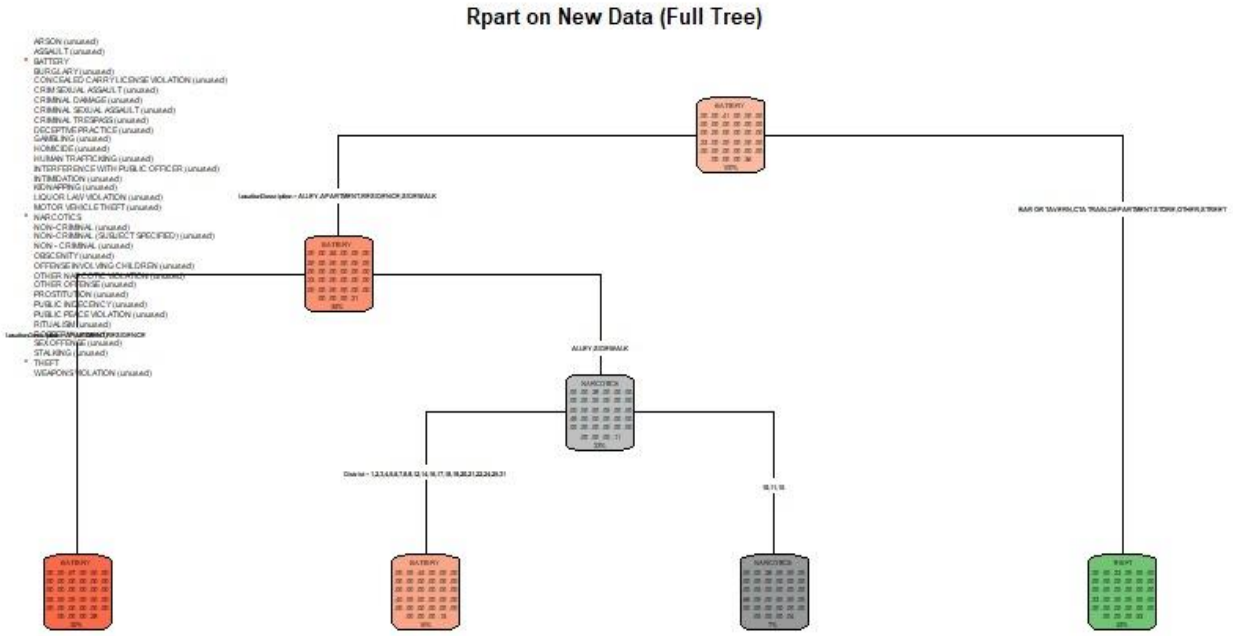
Predictors: Location Description, District, Time of the day

Only three predictors are included in the decision tree. Location description talks about the where the crime had taken place (e.g. streets, residence). There are 25 districts in Chicago (e.g. Central, Lincoln). According to the time at which the crime has occurred we labeled every crime in 5 categories viz. early morning, dawn, morning, afternoon, evening, night.

Target Variable: Primary Type

The aim is to predict the type of the crime given the location, the district and the time of the day.

```
Overall Statistics

               Accuracy : 0.5725
                 95% CI : (0.571, 0.5739)
    No Information Rate : 0.4101
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3183
```



Analysis and Results

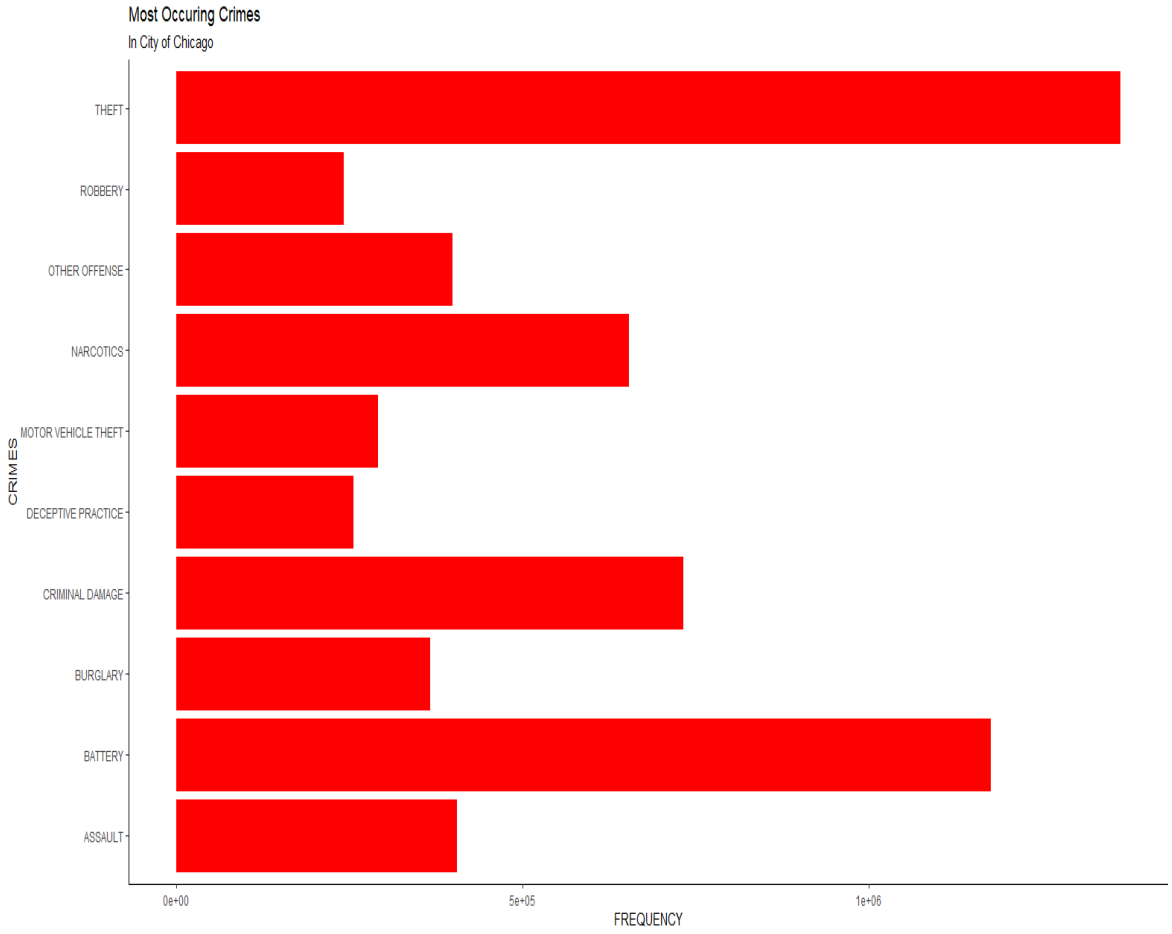
Analysis:

The aim of the study was to find the answers to the 9 research questions we had defined. Our study shows that theft was the most occurring crime in the city of Chicago and battery was the second most frequent crime in the city. We also found out almost 30% of the crimes were committed on the streets. It was interesting to see most of the crimes occurred between 3 PM and 12 AM. When we analyzed crime by season throughout the year, we realized that most of the crimes occurred in summer. Other seasons were relatively quiet as compared to summer. Further, we analyzed the crime battery in detail and came to the conclusion that most of these incidents happened in apartments. By using K-Means algorithm we divided the city into 60 parts and the results show theft is the dominant type of the crime in 36 parts of the city , battery is the dominant type of the crime in 19 parts of the city and narcotics is the most occurring crime in 5 parts of the city. The model created using decision trees had an overall accuracy of 62%. Also, theft, battery, and narcotics had per class balanced accuracy of 69%, 77%, 69% respectively.

```
Confusion Matrix and Statistics

          Reference
Prediction  BATTERY NARCOTICS THEFT
  BATTERY      4436       269  1918
  NARCOTICS    1202      1362   836
  THEFT         878       259  3020

Overall Statistics

               Accuracy : 0.6219
                 95% CI : (0.6138, 0.6299)
    No Information Rate : 0.4595
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4036

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: BATTERY Class: NARCOTICS Class: THEFT
Sensitivity                  0.6808          0.72063       0.5230
Specificity                  0.7146          0.83417       0.8647
Pos Pred Value               0.6698          0.40059       0.7265
Neg Pred Value               0.7248          0.95102       0.7252
Prevalence                   0.4595          0.13329       0.4072
Detection Rate               0.3128          0.09605       0.2130
Detection Prevalence         0.4671          0.23977       0.2932
Balanced Accuracy            0.6977          0.77740       0.6939
```

Visualization of the results:

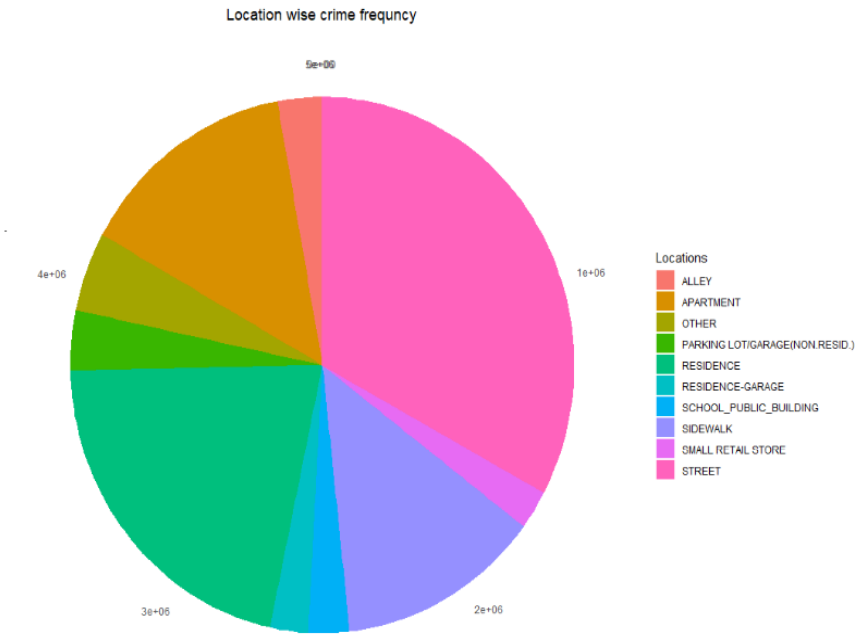
1.What are the most occurring crimes in the city?



Hive:

```
2020-04-28 21:29:22,049 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 1.47 sec
2020-04-28 21:29:23,071 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 1.47 sec
MapReduce Total cumulative CPU time: 1 seconds 470 msec
Ended Job = job_1588124911529_0017
MapReduce Jobs Launched:
Job 0: Map: 7  Reduce: 2   Cumulative CPU: 30.93 sec   HDFS Read: 1873114766 HDFS Write: 1554 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.47 sec   HDFS Read: 2157 HDFS Write: 210 SUCCESS
Total MapReduce CPU Time Spent: 32 seconds 400 msec
OK
"THEFT" 1364098
"BATTERY"       1176836
"CRIMINAL DAMAGE"       733021
"NARCOTICS"     654359
"ASSAULT"       405186
"OTHER OFFENSE" 399222
"BURGLARY"      367477
"MOTOR VEHICLE THEFT"   291224
"DECEPTIVE PRACTICE"    256500
"ROBBERY"       242666
Time taken: 106.05 seconds, Fetched: 10 row(s)
hive>
    >
```

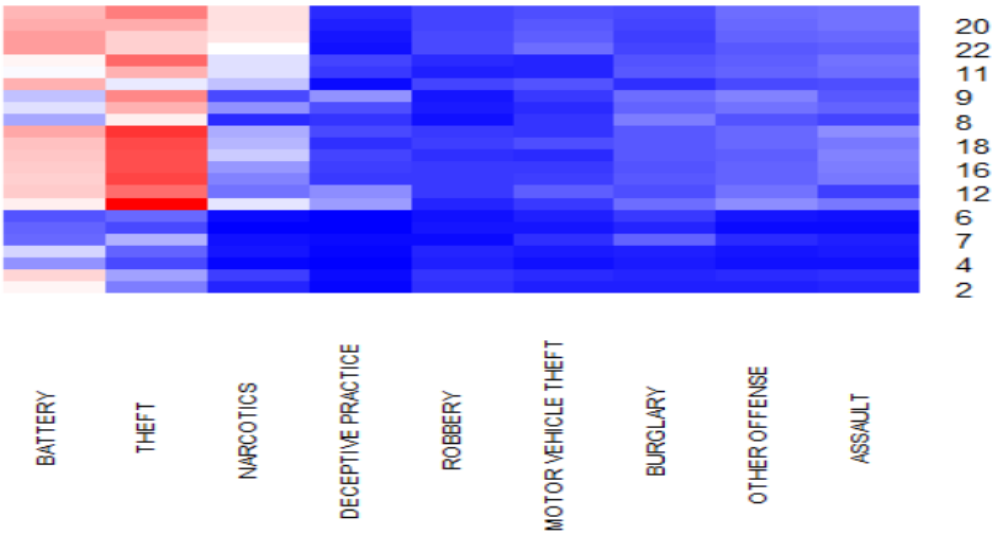
2.How many crimes are being committed at a specific location? (e.g. street, residence)



Hive:

```
Job 0: Map: 92  Reduce: 2   Cumulative CPU: 130.17 sec   HDFS Read: 1757860539 HDFS Write: 9046 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.3 sec   HDFS Read: 9649 HDFS Write: 232 SUCCESS
Total MapReduce CPU Time Spent: 2 minutes 11 seconds 470 msec
OK
"STREET"        1623790
"RESIDENCE"     1066747
"APARTMENT"     703397
"SIDEWALK"      650372
"OTHER" 236732
"PARKING LOT/GARAGE(NON.RESID.)"        178179
"ALLEY" 142835
"SCHOOL; PUBLIC; BUILDING"      131412
"RESIDENCE-GARAGE"      121920
"SMALL RETAIL STORE"    118276
Time taken: 548.776 seconds, Fetched: 10 row(s)
hive>
```

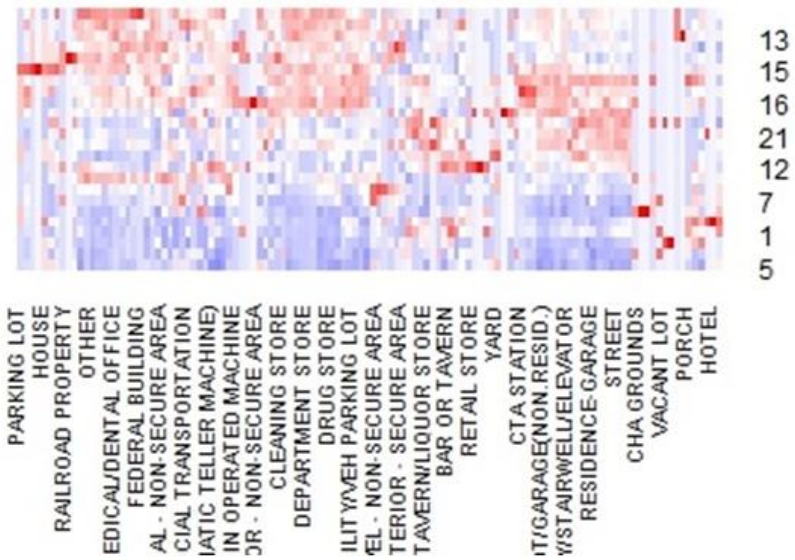
3.Which crimes are being committed at a specific time of the day?



Hive:

```
10      "OTHER OFFENSE" 22681
10      "THEFT" 63090
10      "ROBBERY"       6903
10      "BATTERY"       41952
10      "ASSAULT"       19722
10      "BURGLARY"      19896
10      "DECEPTIVE PRACTICE"    15797
10      "MOTOR VEHICLE THEFT"   8748
10      "NARCOTICS"     27900
11      "OTHER OFFENSE" 20334
11      "THEFT" 63169
11      "ROBBERY"       8121
11      "BATTERY"       47389
11      "ASSAULT"       21377
11      "BURGLARY"      18207
11      "DECEPTIVE PRACTICE"    11794
11      "MOTOR VEHICLE THEFT"   7402
11      "NARCOTICS"     42806
12      "DECEPTIVE PRACTICE"    27237
12      "BURGLARY"      16187
12      "THEFT" 75265
12      "ROBBERY"       12313
```

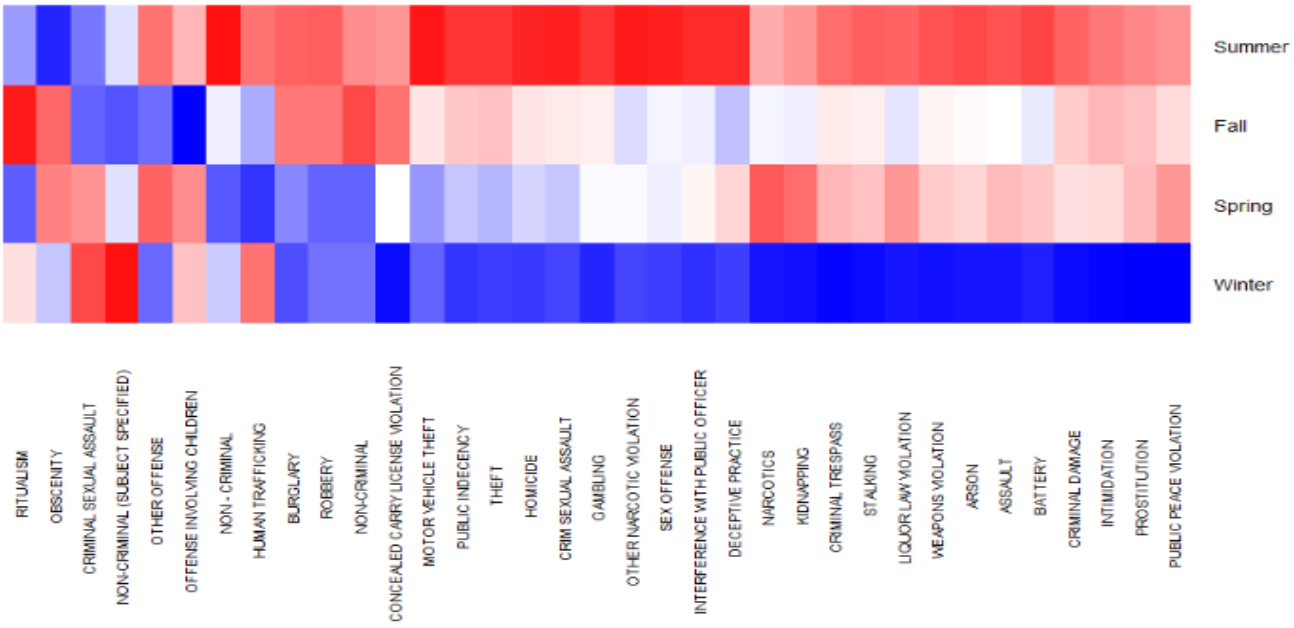
4.Analysis of crimes with respect to locations and time of the day



Hive:

```
MapReduce Jobs Launched:
Job 0: Map: 7  Reduce: 2   Cumulative CPU: 33.12 sec   HDFS Read: 1757467331 HDFS Write: 149639 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.62 sec   HDFS Read: 150240 HDFS Write: 283 SUCCESS
Total MapReduce CPU Time Spent: 34 seconds 740 msec
OK
22      "STREET"        120187
21      "STREET"        113437
20      "STREET"        110027
12      "STREET"        103078
19      "STREET"        101255
23      "STREET"        95985
18      "STREET"        91108
17      "STREET"        77130
24      "RESIDENCE"     75498
12      "RESIDENCE"     75035
15      "STREET"        72290
16      "STREET"        69872
9       "RESIDENCE"     68543
24      "STREET"        67960
14      "STREET"        65811
```

5.What are the most occurring crimes in different seasons?

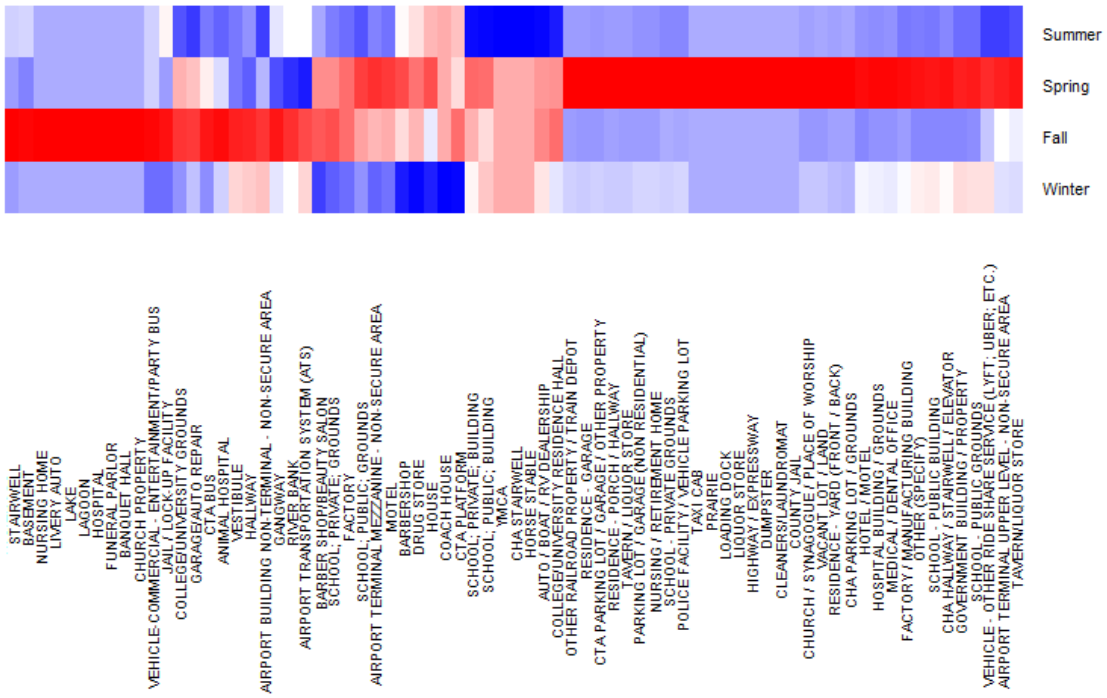


Pig:

```
grunt> Group_By_Season_Primary = GROUP Crime_Data BY (season, primary_type);
grunt> Count_Season = foreach Group_By_Season_Primary GENERATE group, COUNT(Crime_Data.id);
grunt> STORE Count_Season INTO '/user/PigResults/count_season' using PigStorage(',');
```

```
2020-04-29 19:21:53,924 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2020-04-29 19:21:53,924 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
((1,"MOTOR VEHICLE THEFT"),1130)
((2,"MOTOR VEHICLE THEFT"),822)
((3,"MOTOR VEHICLE THEFT"),714)
((4,"MOTOR VEHICLE THEFT"),595)
((5,"MOTOR VEHICLE THEFT"),682)
((6,"MOTOR VEHICLE THEFT"),935)
((7,"MOTOR VEHICLE THEFT"),1070)
((8,"MOTOR VEHICLE THEFT"),1332)
((9,"MOTOR VEHICLE THEFT"),1389)
((10,"MOTOR VEHICLE THEFT"),1187)
```

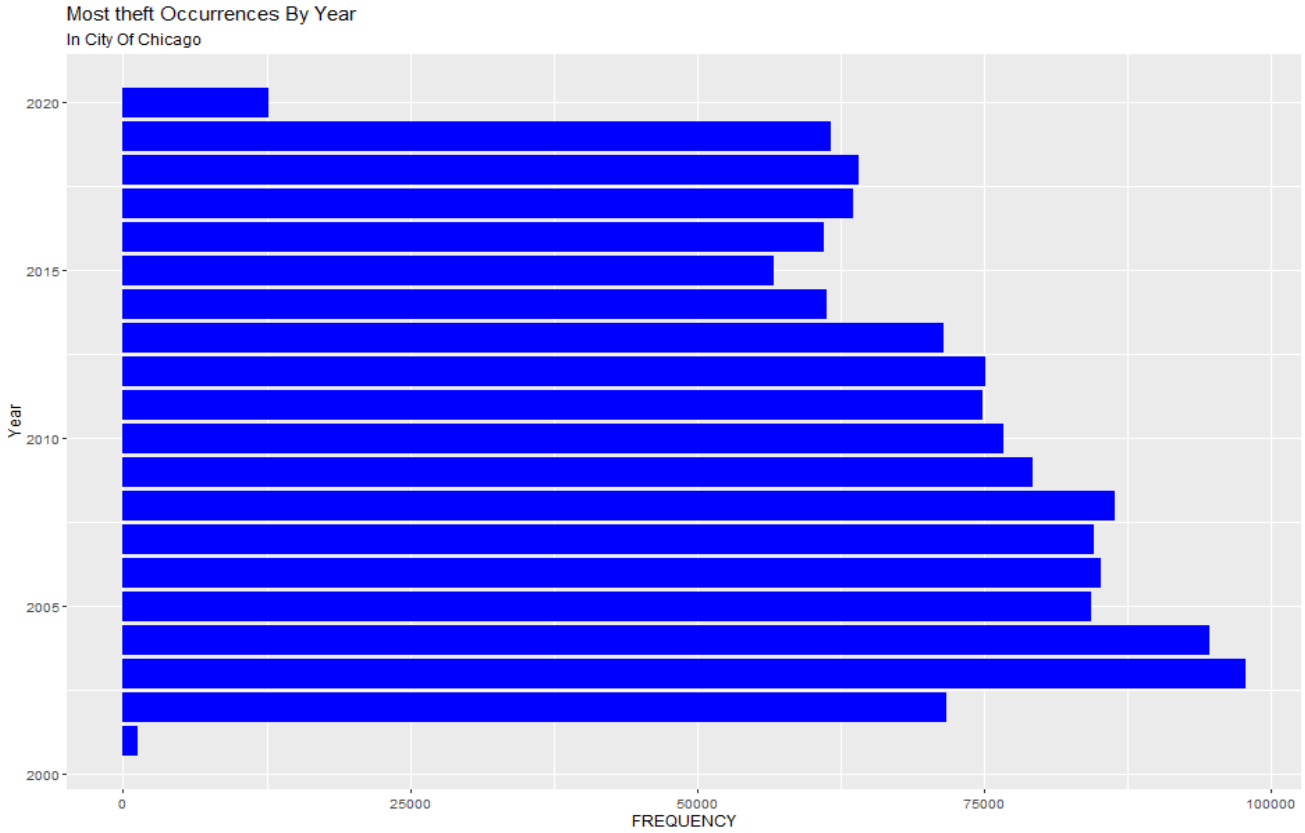
6. Analysis of crimes with respect to locations and seasons



Pig:

```
grunt> Group_By_Season_Location = GROUP Crime_Data BY (season,location_description);
grunt> Count_location = foreach Group_By_Season_Location GENERATE group, COUNT(Crime_Data.id);
grunt> STORE Count_location INTO '/user/PigResults/count_location1' using PigStorage(',');
2020-04-29 19:25:20,513 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2020-04-29 19:25:20,540 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY
2020-04-29 19:25:20,584 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2020-04-29 19:25:20,584 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2020-04-29 19:25:20,584 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator,
GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatte
n, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2020-04-29 19:25:20,587 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
2020-04-29 19:25:20,587 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing to move algebraic foreach to combiner
2020-04-29 19:25:20,588 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2020-04-29 19:25:20,588 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
2020-04-29 19:25:20,607 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2020-04-29 19:25:20,608 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2020-04-29 19:25:20,609 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.MRScriptState - Pig script settings are added to the job
2020-04-29 19:25:20,609 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not
 set, set to default 0.3
2020-04-29 19:25:20,609 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Reduce phase detected, estimating # of required r
educers.
2020-04-29 19:25:20,609 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Using reducer estimator: org.apache.pig.backend.h
adoop.executionengine.mapReduceLayer.InputSizeReducerEstimator
```

7.Analysis of a particular crime type over the years (Crime: Theft)



Pig:

```
grunt> all_theft = FILTER Crime_Data BY (primary_type == '"THEFT"');
grunt> grouped_theft = GROUP all_theft BY year;
grunt> count_crimes_per_year = foreach grouped_theft GENERATE group, COUNT(all_theft.id);
grunt> STORE count_crimes_per_year INTO '/u01/BDT_project/PigResults/Pig' using PigStorage(',');
grunt>
```

```
2020-04-29 18:52:30,043 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(2001,1232)
(2002,71727)
(2003,97796)
(2004,94635)
(2005,84298)
(2006,85226)
(2007,84557)
(2008,86339)
(2009,79248)
(2010,76680)
grunt>
```

Conclusion

This study started with looking for an interesting dataset. We explored the field of exploratory data analysis and we were able to visualize the results we got from our analysis. We also implemented the data mining techniques to real-world problems and 62% accuracy is acceptable in the industry. Along the way, we learned and used four big data technologies and now we feel very comfortable using these technologies. Further, this study made us more confident about learning new technologies and taught us to be more proactive while handling multiple big data technologies. We wish to continue this study as we feel there are certain insights yet to be found from this dataset which will help the city police department immensely.

References

[1] Crime rate in Chicago, Illinois (IL): murders, rapes, robberies, assaults, burglaries, thefts, auto thefts, arson, law enforcement employees, police officers, crime map. (n.d.). Retrieved from http://www.city-data.com/crime/crime-Chicago-Illinois.html.

[2] Chicago Driving Uptick in Murders; National Crime Rate Stays Near 'Historic Lows'. (n.d.). Retrieved from https://www.usnews.com/news/articles/2016-09-19/chicago-drives-uptick-in-murders-national-crime-rate-stays-near-historic-lows.

[3] Sanburn, Josh. "Chicago Responsible for Nearly Half of U.S. Homicide Spike." *Time*, Time, 19 Sept. 2016, http://time.com/4497814/chicago-murder-rate-u-s-crime/.

[4] HDFS Architecture Guide. (n.d.). Retrieved from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[5] Machine Learning Library (MLlib) Guide. (n.d.). Retrieved from https://spark.apache.org/docs/latest/ml-guide.html.

[6] Ingilevich, V., & Ivanov, S. (2018). Crime rate prediction in the urban environment using social factors. In *Procedia Computer Science* (Vol. 136, pp. 472–478). Elsevier B.V. https://doi.org/10.1016/j.procs.2018.08.261

[7] Toppireddy, H. K. R., Saini, B., & Mahajan, G. (2018). Crime Prediction & Monitoring Framework Based on Spatial Analysis. In *Procedia Computer Science* (Vol. 132, pp. 696–705). Elsevier B.V. https://doi.org/10.1016/j.procs.2018.05.075

[8] Alexandros Belesiotis, George Papadakis, and Dimitrios Skoutas. 2018. Analyzing and Predicting Spatial Crime Distribution Using Crowdsourced and Open Data. ACM Trans. Spatial Algorithms Syst. 3, 4, Article 12 (April 2018), 31 pages. DOI: https://doi.org/10.1145/3190345

[9] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, 2010, pp. 1-10. doi: 10.1109/MSST.2010.5496972
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5496972&isnumber=5496967

[10] Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N. Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2014. Major technical advancements in apache hive. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14). ACM, New York, NY, USA, 1235-1246. DOI: https://doi.org/10.1145/2588555.2595630

[11] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. 2010. Hive - a petabyte scale data warehouse using Hadoop. In ICDE

[12] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig latin: a not-so-foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08). ACM, New York, NY, USA, 1099-1110. DOI: https://doi.org/10.1145/1376616.1376726

[13] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI'04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10

[14] V. Kalavri and V. Vlassov, "MapReduce: Limitations, Optimizations and Open Issues," 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, 2013, pp. 1031-1038. doi: 10.1109/TrustCom.2013.126

[15] 2015 data science salary survey. https: //www.oreilly.com/ideas/2015-data-science-salary-survey.

[16] Venkataraman, Shivaram & Stoica, Ion & Zaharia, Matei & Yang, Zongheng & Liu, Davies & Liang, Eric & Falaki, Hossein & Meng, Xiangrui & Xin, Reynold & Ghodsi, Ali & Franklin, Michael. (2016). SparkR: Scaling R Programs with Spark. 1099-1104. 10.1145/2882903.2903740.

[17] S. Na, L. Xumin and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, Jinggangshan, 2010, pp. 63-67. doi: 10.1109/IITSI.2010.74 URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5453745&isnumber=5453559

[18] Mesaric, Josip & Šebalj, Dario. (2016). Decision trees for predicting the academic success of students. Croatian Operational Research Review. 7. 367-388. 10.17535/crorr.2016.0025.

Dataset

[DS] Crimes - 2001 to present | City of Chicago | Data Portal. url: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

Appendix:

Clean and prepare dataset

```r
library(cluster)
library(factoextra)
library(dplyr)
library(datasets)
set.seed(20)

chicago.df <- read.csv("C:\\Project\\Data\\Crimes_-_2001_to_present.csv", header = T, stringsAsFactors = F)
#str(chicago.df)

sapply(chicago.df, function(x) sum(x=="" | is.na(x)))


sapply(chicago.df, function(x) sum(x=="" | is.na(x)))*100/nrow(chicago.df)

missing <- apply(chicago.df, 1, function(x) sum(x=="" | is.na(x)))/ncol(chicago.df)

head(missing[order(-missing)])

chicago.df <- chicago.df[missing == 0,]

jpeg('C:\\Project\\Results\\Plots\\rplot1.jpg')
boxplot(chicago.df$X.Coordinate)
dev.off()
jpeg('C:\\Project\\Results\\Plots\\rplot2.jpg')
boxplot(chicago.df$Y.Coordinate)
dev.off()

chicago.df <- filter(chicago.df, X.Coordinate != 0, Y.Coordinate != 0)

jpeg('C:\\Project\\Results\\Plots\\rplot1.jpg')
boxplot(chicago.df$X.Coordinate)
dev.off()
jpeg('C:\\Users\\Project\\Results\\Plots\\rplot2.jpg')
boxplot(chicago.df$Y.Coordinate)
dev.off()

write.csv(chicago.df,"C:\\Project\\Data\\clean_data.csv")
```

```r
# Importing required libraries.
library(lubridate)
library(stringr)
# install.packages("dplyr")
library(dplyr)
library(data.table)
library(ggplot2)

#Read cleaned data
Dst <- read.csv("C:\\Project\\Data\\clean_data.csv", header = T, stringsAsFactors = F)

# Splitting Date and Time in two different Columns.
m<-str_split_fixed(Dst$Date, " ", 3)
Dst$Date <-m[,1]
Dst$Time <-m[,2]
Dst$AMPM <-m[,3]


#Splits the time as HH MM
# split <- strsplit(Dst$Time, ":")

t<-str_split_fixed(Dst$Time, ":", 3)
Dst$Hour <-t[,1]
Dst$Minute <-t[,2]
# Dst$AMPM <-m[,3]


Dst$Hour <- as.numeric(Dst$Hour)
#Converts time in 24 hour format
condition <- Dst$AMPM == "PM"
Dst$Hour[condition] <- Dst$Hour[condition] + 12

v<-str_split_fixed(Dst$Date, "/", 3)
Dst$Month <-v[,1]

season <- c(rep("Winter", nrow(Dst)))
```

```r
#Assigns the correct season according to the month
for(n in 1:nrow(Dst))
{
  if(Dst$Month[n] >= 3 && Dst$Month[n] < 6)
  {
    season[n] = "Spring"
  }
  else if (Dst$Month[n] >= 6 && Dst$Month[n] < 9)
  {
    season[n] = "Summer"
  }
  else if (Dst$Month[n] >= 9 && Dst$Month[n] < 12)
  {
    season[n] = "Fall"
  }
}

Dst <- cbind(Dst,season)

drops <- c("AMPM","Minute","")
Dst <- Dst[ , !(names(Dst) %in% drops)]

write.csv(Dst,"C:\\Project\\Data\\new_clean_data_final.csv")
```

Hive queries

```
CREATE DATABASE IF NOT EXISTS MyDb;

CREATE TABLE IF NOT EXISTS MyDb.CrimeData
(no INT,
id INT,
case_number VARCHAR(10000),
date_field STRING,
block VARCHAR(10000),
IUCR VARCHAR(10000),
primary_type VARCHAR(10000),
description VARCHAR(10000),
location_description VARCHAR(10000),
arrest VARCHAR(10000),
domestic VARCHAR(10000),
beat INT,
district INT,
ward INT,
community_area INT,
fbi_code VARCHAR(10000),
x_coordinate INT,
y_coordinate INT,
year INT,
updated_on STRING,
latitude DOUBLE,
longitude DOUBLE,
hour_of_the_day INT,
location STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS textfile LOCATION
'/user/maria_dev/data.csv'
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA LOCAL INPATH './cleaned_data.csv' OVERWRITE INTO TABLE MyDb.CrimeData;

SELECT  primary_type, COUNT(id) as cnt from MyDb.CrimeData GROUP BY primary_type ORDER BY cnt DESC LIMIT 3;

SELECT location_description, COUNT(id) as cnt from MyDb.CrimeData GROUP BY location_description ORDER BY cnt DESC LIMIT 10;

SELECT hour_of_the_day, primary_type, COUNT(id) as cnt from MyDb.CrimeData GROUP BY hour_of_the_day,
primary_type ORDER BY hour_of_the_day;

SELECT hour_of_the_day, location_description, COUNT(id) as cnt from MyDb.CrimeData WHERE hour_of_the_day > 0 AND
hour_of_the_day < 25 GROUP BY hour_of_the_day, location_description ORDER BY cnt DESC LIMIT 15;
```

Pig Latin script

```
Crime_Data = LOAD '/user/maria_dev/new_clean_data_final.csv' USING PigStorage(',')
AS (
no:int, id:int, case_number:chararray, date_field:chararray,
block:chararray, IUCR:chararray, primary_type:chararray,
description:chararray, location_description:chararray,
arrest:chararray, domestic:chararray, beat:int,
district:int, ward:int, community_area:int,
fbi_code:chararray, x_coordinate:int, y_coordinate:int,
year:int, updated_on:chararray, latitude:double,
longitude:double, location:chararray, hour_of_the_day:int, season:chararray);
DESCRIBE Crime_Data;

all_theft = FILTER Crime_Data BY (primary_type MATCHES 'THEFT');
grouped_theft = GROUP all_theft BY year;
count_crimes_per_year = foreach grouped_theft GENERATE group, COUNT(all_theft.id);
STORE count_crimes_per_year INTO 'crime_per_year' using PigStorage(',');

Group_By_Season_Primary = GROUP Crime_Data BY (season, primary_type);
Count_Season = foreach Group_By_Season_Primary GENERATE group, COUNT(Crime_Data.id);
STORE Count_Season INTO 'count_season.csv' using PigStorage(',');

Group_By_Season_Location = GROUP Crime_Data BY (season,location_description);
Count_location = foreach Group_By_Season_Location GENERATE group, COUNT(Crime_Data.id);
STORE Count_location INTO 'count_location' using PigStorage(',');
```

K-Means clustering

```
## Important essential libararies
import numpy as np
import pandas as pd
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.clustering import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

#Read the data and store it in a dataframe
chicagoDF = pd.read_csv("/u01/BDT_project/CrimeData/clean_data.csv")

# rename dataframe columns to remove '.' from there names
chicagoDF.rename(columns={'X.Coordinate':'xcor','Y.Coordinate':'ycor','Primary.Type':'primary'},inplace=True)

# create dataframe only with essential columns
mapDF = chicagoDF[['xcor','ycor','primary']]

## create spark dataframe
SparkMapdf = spark.createDataFrame(mapDF)

## prepare the dataframe for k means training
features=('xcor','ycor')
assembler = VectorAssembler(inputCols=features,outputCol="features")
mapDataset=assembler.transform(SparkMapdf)

#mapDataset.select("features").show(truncate=False)

## Train a kemeans model
kmeans = KMeans().setK(60).setSeed(1)
model = kmeans.fit(mapDataset)

#model.summary

# Make predictions
predictions = model.transform(mapDataset)

```

```
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Evaluate clustering.
cost = model.computeCost(mapDataset)
print("Within Set Sum of Squared Errors = " + str(cost))

# print results
print("Cluster Centers: ")
ctr=[]
centers = model.clusterCenters()
for center in centers:
    ctr.append(center)
    print(center)


# Create dataframe from predictions
pandasDF=predictions.toPandas()
pandasDF.to_csv('/home/hadoop/pandasDF.csv')

label=pd.DataFrame(list(range(0,60)))
frames=(centers,label)
centerLabel=pd.concat(frames,axis=1)

## extract predictions from dataframe
y_means=pandasDF["prediction"]

## Visualize clustering results
plot=sns.lmplot(data=pandasDF,x='xcor',y='ycor',hue='cluster',fit_reg=False,legend=False,legend_out=False,height=7)
plot.fig.legend(ncol=2)
plot.savefig('/home/hadoop/KMeans.png')

##y_kmeans = model.predict(mapDataset)


#Extract most occurring crime for every cluster
from pyspark.sql.types import StructField, StructType, StringType, IntegerType
CrimeDataSchema = StructType([
StructField("id", IntegerType(), True),
StructField("xcor", IntegerType(), True),
StructField("ycor", IntegerType(), True),
StructField("primary", StringType(), True),
StructField("cluster", IntegerType(), True),
])
CrimeData = spark.read.format("csv")\
.schema(CrimeDataSchema)\
.load("/project/pandasDF.csv")

CrimeData.createOrReplaceTempView("CrimeDataView")

t=spark.sql('select primary,cluster,count(*) as cnt from CrimeDataView group by primary,cluster')
t.createOrReplaceTempView('tt')

y=spark.sql('select cluster,primary,cnt from tt where (cluster,cnt) in (select cluster,max(cnt) from tt group by cluster) order by cluster')
y.show(60)

## Create most occuring crime cluster.csv file
```

Decision trees

```
library(SparkR)
library(cluster)
library(factoextra)
library(dplyr)
library(datasets)
library(psych)
library(rpart)
library(rpart.plot)
library(DMwR)

set.seed(20)

#Reads the dataset into a dataframe
chicago.df <- read.csv("clean_data.csv", header = T, stringsAsFactors = F)

#Splits the dataset into train and test datasets in 80-20 proportion
indx <- sample(1:nrow(chicago.df), 0.8*nrow(chicago.df))
train.sample.chicago.df <- chicago.df[indx, ]
test.sample.chicago.df <- chicago.df[-indx, ]

#Makes sure the predictors and target variable are factors
train.new.primary.type <- as.factor(train.sample.chicago.df$Primary.Type)
train.new.location.description <- as.factor(train.sample.chicago.df$Location.Description)
train.new.district <- as.factor(train.sample.chicago.df$District)

#Splits the the dates as date and time
time1 <- strsplit(train.sample.chicago.df$Date, " ")
times <- c()
for(n in 1:nrow(train.sample.chicago.df))
{
  times <- c(times, as.character(time1[[n]][2]))
}

#Splits the time as HH MM
split <- strsplit(times, ":")
hour <- c()
```

```r
for(n in 1:nrow(train.sample.chicago.df))
{
  hour <- c(hour,as.numeric(split[[n]][1]))
}

#Extracts the hour
time2 <- c()
for(n in 1:nrow(train.sample.chicago.df))
{
  time2 <- c(time2, as.character(time1[[n]][3]))
}

time.of.the.day <- c(rep("Night", nrow(train.sample.chicago.df)))
time.df <- data.frame(hour,time2,time.of.the.day,stringsAsFactors = F)

#Adds 12 hours if is PM i.e. past noon
condition <- time.df$time2 == "PM"
time.df$hour[condition] <- time.df$hour[condition] + 12

#Categories the hours in 5 times of the day
for(n in 1:nrow(train.sample.chicago.df))
{
  if(time.df[n,1] >= 1 && time.df[n,1] < 4)
  {
    time.df[n,3] = "Early Morning"
  }
  else if (time.df[n,1] >= 4 && time.df[n,1] < 6)
  {
    time.df[n,3] = "Dawn"
  }
  else if (time.df[n,1] >= 6 && time.df[n,1] < 12)
  {
    time.df[n,3] = "Morning"
  }
```

```
  else if (time.df[n,1] >= 12 && time.df[n,1] < 16)
  {
    time.df[n,3] = "Afternoon"
  }
  else if(time.df[n,1] >= 16 && time.df[n,1] < 21)
  {
    time.df[n,3] = "Evening"
  }
}

TOTD <- as.factor(time.df$time.of.the.day)


new.for.corelation <- data.frame(train.new.primary.type, train.new.location.description, train.new.district, TOTD)
new.df <- filter(new.for.corelation, train.new.location.description == "STREET" | train.new.location.description == "RESIDENCE" | train.new
new.df <- filter(new.df, train.new.primary.type == "BATTERY" | train.new.primary.type == "NARCOTICS" |  train.new.primary.type == "THEFT")

new.df$train.new.primary.type <- factor(new.df$train.new.primary.type)
new.df$train.new.location.description <- factor(new.df$train.new.location.description)
new.df$train.new.district <- factor(new.df$train.new.district)

#Smote technique is used to handle class imbalance
new.df.smote <- SMOTE(train.new.primary.type ~ train.new.location.description + train.new.district + TOTD, new.df, perc.over = 100, perc.un

#Makes sure the predictors and target variable are factors
test.new.primary.type <- as.factor(test.sample.chicago.df$Primary.Type)
test.new.location.description <- as.factor(test.sample.chicago.df$Location.Description)
test.new.district <- as.factor(test.sample.chicago.df$District)

#Splits the the dates as date and time
test.time1 <- strsplit(test.sample.chicago.df$Date, " ")
test.times <- c()
for(n in 1:nrow(test.sample.chicago.df))
{
  test.times <- c(test.times, as.character(test.time1[[n]][2]))
}
```

```
test.split <- strsplit(test.times, ":")
test.hour <- c()
for(n in 1:nrow(test.sample.chicago.df))
{
  test.hour <- c(test.hour,as.numeric(test.split[[n]][1]))
}

test.time2 <- c()
for(n in 1:nrow(test.sample.chicago.df))
{
  test.time2 <- c(test.time2, as.character(test.time1[[n]][3]))
}


test.time.of.the.day <- c(rep("Night", nrow(test.sample.chicago.df)))
test.time.df <- data.frame(test.hour,test.time2,test.time.of.the.day,stringsAsFactors = F)

test.condition <- test.time.df$test.time2 == "PM"
test.time.df$test.hour[test.condition] <- test.time.df$test.hour[test.condition] + 12


for(n in 1:nrow(test.sample.chicago.df))
{
  if(test.time.df[n,1] >= 1 && test.time.df[n,1] < 4)
  {
    test.time.df[n,3] = "Early Morning"
  }
  else if (test.time.df[n,1] >= 4 && test.time.df[n,1] < 6)
  {
    test.time.df[n,3] = "Dawn"
  }
  else if (test.time.df[n,1] >= 6 && test.time.df[n,1] < 12)
  {
    test.time.df[n,3] = "Morning"
```

```
  }
  else if (test.time.df[n,1] >= 12 && test.time.df[n,1] < 16)
  {
    test.time.df[n,3] = "Afternoon"
  }
  else if(test.time.df[n,1] >= 16 && test.time.df[n,1] < 21)
  {
    test.time.df[n,3] = "Evening"
  }
}

test.TOTD <- as.factor(test.time.df$test.time.of.the.day)


test.new.for.corelation <- data.frame(test.new.primary.type, test.new.location.description, test.new.district, test.TOTD)
test.new.df <- filter(test.new.for.corelation, test.new.location.description == "STREET" | test.new.location.description == "RESIDENCE" | t
test.new.df <- filter(test.new.df, test.new.primary.type == "BATTERY" | test.new.primary.type == "NARCOTICS" | test.new.primary.type == "TH
test.new.df<-test.new.df[test.new.df$test.new.district != 31,]
colnames(test.new.df) <- c("train.new.primary.type", "train.new.location.description", "train.new.district", "TOTD")


test.new.df$train.new.primary.type <- factor(test.new.df$train.new.primary.type)
test.new.df$train.new.location.description <- factor(test.new.df$train.new.location.description)
test.new.df$train.new.district <- factor(test.new.df$train.new.district)

#Trais the model
model <- rpart(train.new.primary.type ~ train.new.location.description + train.new.district + TOTD, method="class", data=new.df.smote)

#Creates the tree for visualization
rpart.plot(model, extra=104, fallen.leaves = T, type=4, main="Rpart on New Data (Full Tree)")

#Makes the prediction on the test data
pred <- predict(model, test.new.df, type="class")

#Confusion matrix gives us a good idea about accuracy as well as class-wise sensitivity,specificity and balanced accuracy
confusionMatrix(pred, test.new.df$train.new.primary.type)
```

Visualization of the results

```
library(ggplot2)
library(gplots)

crime.freq.df <- read.csv("hive_que1.csv", header = F, stringsAsFactors = F)

ggplot(crime.freq.df, aes(x=crimes, y= crimes.frequency))+
  xlab("CRIMES")+
  ylab("FREQUENCY") +
geom_bar(stat = "identity", fill="red") +
  coord_flip() +
  labs(title="Most Occuring Crimes",
       subtitle="In City of Chicago") +
  theme(axis.text.x = element_text(vjust=0.6))


theme_set(theme_classic())

location.freq.df  <- read.csv("hive_que2.csv", header = F, stringsAsFactors = F)
colnames(location.freq.df) <- c("Locations", "Frequency")
pie <- ggplot(location.freq.df, aes(x = "", y=Frequency, fill = factor(Locations))) +
  geom_bar(width = 1, stat = "identity") +
  theme(axis.line = element_blank(),
        plot.title = element_text(hjust=0.5)) +
  labs(fill="Locations",
       x=NULL,
       y=NULL,
       title="Location wise crime frequncy")

pie + coord_polar(theta = "y", start=0)

hours.crimes.df <- read.csv("hive_que3.csv", header = F, stringsAsFactors = F)
hours.crimes.df[1,1] = 1
colnames(hours.crimes.df) <- c("AtHour", "CrimeType", "Freq")
hours.crimes.df$AtHour <- as.numeric(hours.crimes.df$AtHour)
hours.crimes.df$CrimeType <- as.factor(hours.crimes.df$CrimeType)
hours.crimes.df <- hours.crimes.df[order(hours.crimes.df$AtHour),]
test.res <- xtabs(hours.crimes.df$Freq~hours.crimes.df$AtHour+hours.crimes.df$CrimeType, data=hours.crimes.df)

heatmap.2(test.res, scale = "none", col = bluered(100),
          trace = "none", density.info = "none", key = T)
```

```r
library(ggplot2)
library(gplots)

crimes.by.season <- read.csv("pig1_results.csv", header = F, stringsAsFactors = F)
colnames(crimes.by.season) <- c("Season", "Crime", "Freq")
crimes.by.season$Season <- as.factor(crimes.by.season$Season)
crimes.by.season$Crime <- as.factor(crimes.by.season$Crime)
crimes.by.season <- crimes.by.season[order(crimes.by.season$Season),]
test.res <- scale(xtabs(crimes.by.season$Freq~crimes.by.season$Season+crimes.by.season$Crime, data=crimes.by.season))
heatmap.2(test.res, scale = "none", col = bluered(100),
          trace = "none", density.info = "none", key = T, dendrogram="none",srtCol=25)


locations.by.season <- read.csv("pig2_results.csv", header = F, stringsAsFactors = F)
colnames(locations.by.season) <- c("Season", "Location", "Freq")
locations.by.season$Season <- as.factor(locations.by.season$Season)
locations.by.season$Location <- as.factor(locations.by.season$Location)
locations.by.season <- locations.by.season[order(locations.by.season$Season),]
test.res <- scale(xtabs(locations.by.season$Freq~locations.by.season$Season+locations.by.season$Location, data=locations.by.season))
heatmap.2(test.res, scale = "none", col = bluered(100),
          trace = "none", density.info = "none", key = T, dendrogram="none", Rowv = FALSE, Colv=FALSE)


battery.all <- read.csv("pig3_results.csv", header = F, stringsAsFactors = F)
colnames(battery.all) <- c("Location","Freq")
ggplot(battery.all, aes(x=battery.all$Location, y= battery.all$Freq))+
  xlab("LOCATION")+
  ylab("FREQUENCY") +
  geom_bar(stat = "identity", fill="blue") +
  coord_flip() +
  labs(title="Most Battery Occurrences By Location",
       subtitle="In City Of Chicago") +
  theme(axis.text.x = element_text( vjust=0.6))
```