Buvaneshwaran T & Thinh Huynh

Analysis of Crime in Chicago

Using Amazon Web Services to Analyze and Fight Crime

1. Abstract

The basis of this project is to formulate interesting data science questions and to analyze them using data analysis tools (such as R, Python, and Hive/Pig) in AWS and data mining algorithms (such as Association, Rule-Based, K-nearest) to discover patterns/ trends in crimes committed in the Greater Chicago area during past years. The data-set we used to study is taken from the Chicago Police Department which updates crimes records daily and is hosted by the Chicago Data Portal. The questions we formulated to study the data-set includes, but not limited to:

1. Explore the crime distribution across the various areas – is there a trend, if any?
2. Likelihood of crime being committed during a certain time – are there any?
3. Weapons used/ and severity of crimes across the various boroughs?
4. Pinpoint general time index ex. Season, month etc.?
5. Are there any crimes that are related?

2. Introduction

(2.1) Motivation

In recent years, the notoriety of Chicago's high crime rates has been thrust into the spotlight with the political atmosphere in general. Chicago is notorious, even in the past, for being the crime capital of America due to the prevalence of the Italian Mafia in the city. The Italian Mafia is known for some of the more brutal acts of crime committed in the world.

We offer an alternative approach in this paper, namely the use of technology to fight crime, specifically the Hadoop infrastructure through Amazon Web Services. Traditional approaches, as Mr. Trump proposes, are too narrow-minded, in our opinion, in this technologically advanced 21st century. Hadoop is capable of processing huge data-sets quickly, hence, providing an excellent platform for scaling huge Chicago's crime data-set that is updated every day and which is hosted by the police department of the city. Hadoop through Amazon's Web Service can manage all aspects of "Big Data" such as Volume, Variety, and Velocity, and it is relatively easy to use.

(2.2) Benefits and Use

Our overall approach is to discover machine learning algorithms that are best able to learn from the data-set. We also tested the veracity of the results by spot-checking and defining a test harness and using experiments and heuristics to get the most out of each algorithm used. Upon completion of the project we found the results achieved/ discovered to be satisfactory because these findings can be used to prevent crime in all crime-ridden places, not just Chicago.

3. Problem Definition

The first step is to define our problem. Jason Brownlee, PhD., of MachineLearningMastery.com recommends the following framework when defining a new problem to address with machine learning. This, he proposes, helps us better understand what our goals and motivations are and whether machine learning is a suitable formulation.

(3.1) Informal Description

In here we describe the problem as though we were describing it to a friend.

I need a program that will tell me which crimes are likely to occur at a certain time, at a certain place, given a certain characteristic, to better able to prevent the crime from happening.

(3.2) Formal Description

Tom Mitchell defines a useful formalism for describing a machine learning problem:

A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

This formalism will be used to define the **T**, **P**, and **E** for the problem.

- ***Task (T):*** Classify a crime that has not occurred as likely to occur or not.
- ***Experience (E):*** A corpus of crime records for Chicago.
- ***Performance (P):*** Classification accuracy, the number of crimes predicted correctly of all crimes considered, as a percentage.

(3.3) Assumptions

- The ethnicity and gender of the committer do not matter to the model – although it probably does.
- The specific crimes committed matter to the model.
- The number of crimes in a category may matter to the model.

- The population of each district matters to the model.
- Older crimes may be less predictive due to lack of documentation.

4. Analyze Data

(4.1) Summarize Data

The original data-set is in CSV format, and each horizontal line represented a record, the characteristics of a specific crime, and each record has many attributes separated by comma.

The original data-set contains many attributes which is of little use for our purposes. Hence, we will only include the following attributes, cherry-picking the attributes that are known to have implications for our model and removing the rest. The attributes we included are as follows:

1. *Primary Type* – The primary offense description of the IUCR code.
2. *Arrest* – Indicates whether an arrest was made or not.
3. *Year* – Year the incident occurred.
4. *District* – Indicates the police district where the incident occurred.
5. *Latitude/ Longitude* – The latitude and longitude where the incident occurred.
6. *Location Description* – Description of the location where the incident occurred.

(4.2) Visualize Data

In this section, we'll create as many summaries of the data-set as possible. The motivation here is to create different perspectives or views on the data-set in order to elicit insights about the data. To preserve space, we'll restrict each of our output to 10 lines of data, at the most. In some cases, we may simply include only the diagrammatic output, instead of tables, as a visual representation may be easier to comprehend or may be more appropriate for that query.

The following visualizations are made through amazon's Elastic Map Reduce which uses Hadoop to quickly process our large crime data-set. We'll specifically implement Pig-Latin scripts to manipulate and process our data-set stored in Amazon's S3 service. Upon successful completion of EMR's cluster, the output is stored in Amazon's S3 storage service in the form of tables. These tables are then exported to Microsoft Excel, where appropriate visuals were applied to generate the graphs.

4.2.1 Total number of crimes occurring each year

Input: Crime data set

Output*: Total number of crime from the year 2012-2017*

Algorithm used in AWS EMR.

```
1.  crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
2.  crimeGroupDate = GROUP crime BY Date;
3.  crimeGroupDateCounted = FOREACH crimeGroupDate GENERATE
COUNT(crime) AS cnt;
4.  sorted = ORDER crimeGroupDateCounted BY cnt;
5.  store sorted INTO 's3://bt-cs-383/output';
```

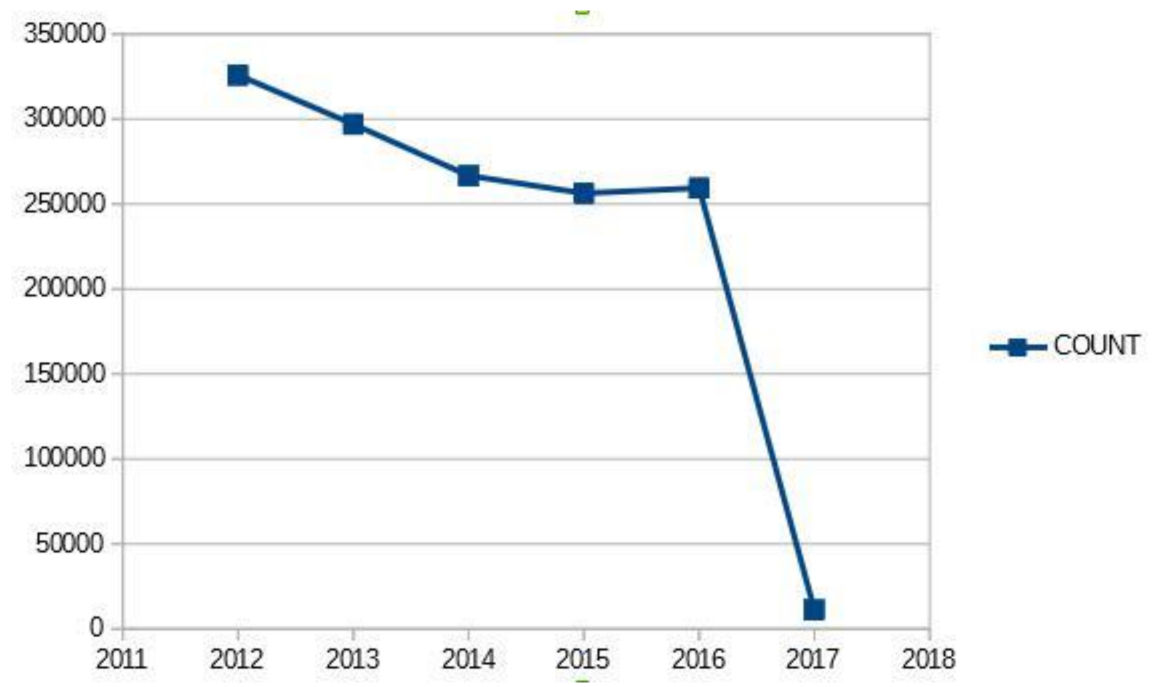


Illustration 2: Total Crimes Occurring by Year

From the above graph, it can be concluded that overall crime occurrences are dropping year after year. However, there is a sudden drop from year 2016 to year 2017, this is because the year 2017 is still running. The crime records have not been updated yet for the year 2017, which will probably be updated at the end of the year. Keep in mind that this data has been updated to the Kaggle website during the January of 2017. Thus, the crime count of 2017 only includes January. For this analysis, we'll only research the four years of 2012 through 2016, inclusive. Thus, it can be concluded that there is a considerable amount of reduction in crime since 2012. This can be taken as a sign that law enforcement is in the right track of targeting and eliminating crime occurrences.

4.2.2 Total Number of crimes occurring in each district.

Input: Crime data set

Output: Total number of crime occurring in each district.

Algorithm used in AWS EMR.

```
1.  crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
2.  crimeGroupDistrict = GROUP crime BY District;
3.  crimeGroupDistrictCounted = FOREACH crimeGroupDistrict GENERATE
COUNT(crime) AS cnt;
4.  sorted = ORDER crimeGroupDistrictCounted BY cnt;
5.  store sorted INTO 's3://bt-cs-383/output';
```

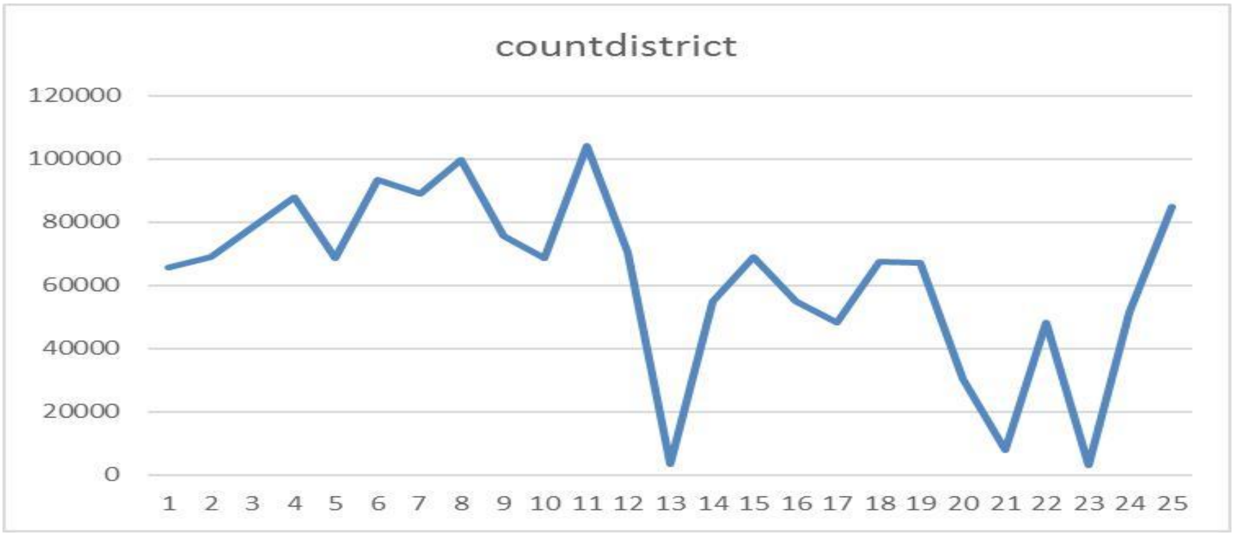


Illustration 3: Total Crimes by District

From the above chart, it can be clearly seen that there are less crime occurrences in certain districts and there's more crime happening in certain districts. This fact can be used while predicting the districts where crimes are more likely to occur. Also, this fact helps law enforcement to focus their attention on certain districts and be more laid back on the other districts where crime occurrences are low. Thus, if for a particular district the amount of crime is found to be high, more police forces must be targeted towards that district compared to others.

4.2.3 Total Number of Crime by Type

Input: Crime data set

Output: Total number of crime by type.

Algorithm used AWS EMR.

```
1.  crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING PigStorage(',') AS (ID, caseNumber,
Date, Block, IUCR, primaryType, Description, locationDescription, Arrest, Domestic, Beat,
District, Ward, communityArea, fbiCode, xCoordinate, yCoordinate, Year, updatedOn, Latitude,
Longitude, Location);
2.  crimeGroupType = GROUP crime BY primaryType;
3.  crimeGroupTypeCounted = FOREACH crimeGroupType GENERATE COUNT(crime) AS
cnt;
4.  sorted = ORDER crimeGroupTypeCounted BY cnt;
5.  store sorted INTO 's3://bt-cs-383/output';
```

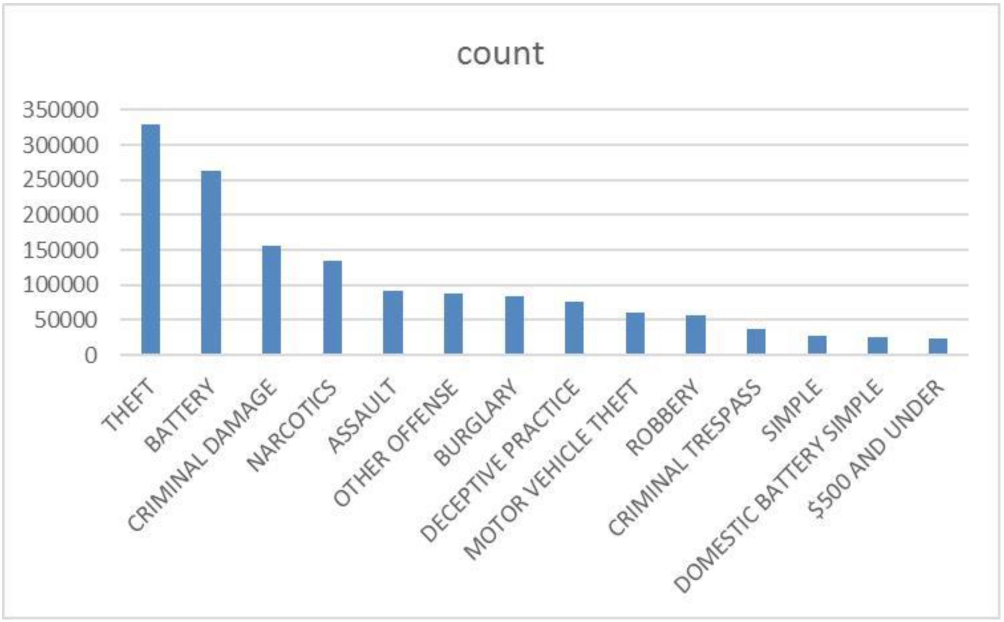


Illustration 4: Total Crimes by Type

From the above graph, it can be seen that Theft, Battery, criminal Damage, and Narcotics are rampant compared to the other crimes. This fact should be used by law enforcement to target these crimes and focus more on these crimes and get the tools necessary to handle these kinds of crimes than the crimes that occur in fewer amounts.

4.2.4 Highest Number of crimes and their dates

Input: Crime data set

Output: Total number of crime in a particular day.

Algorithm Used for AWS EMR:

```
1. crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
2. crimeGroupDate = GROUP crime BY Date;
3. crimeGroupDateCounted = FOREACH crimeGroupDate GENERATE
COUNT(crime) AS cnt;
4. sorted = ORDER crimeGroupDateCounted BY cnt;
5. store sorted INTO 's3://bt-cs-383/output';
```

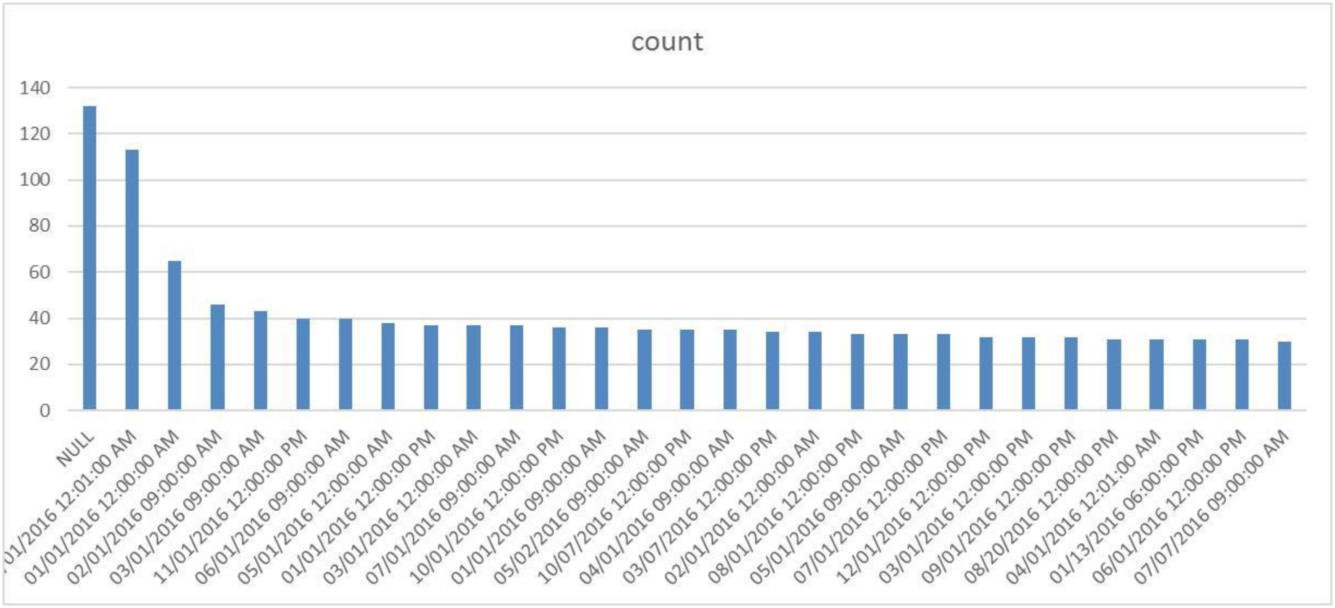


Illustration 5: Total Crimes by Date

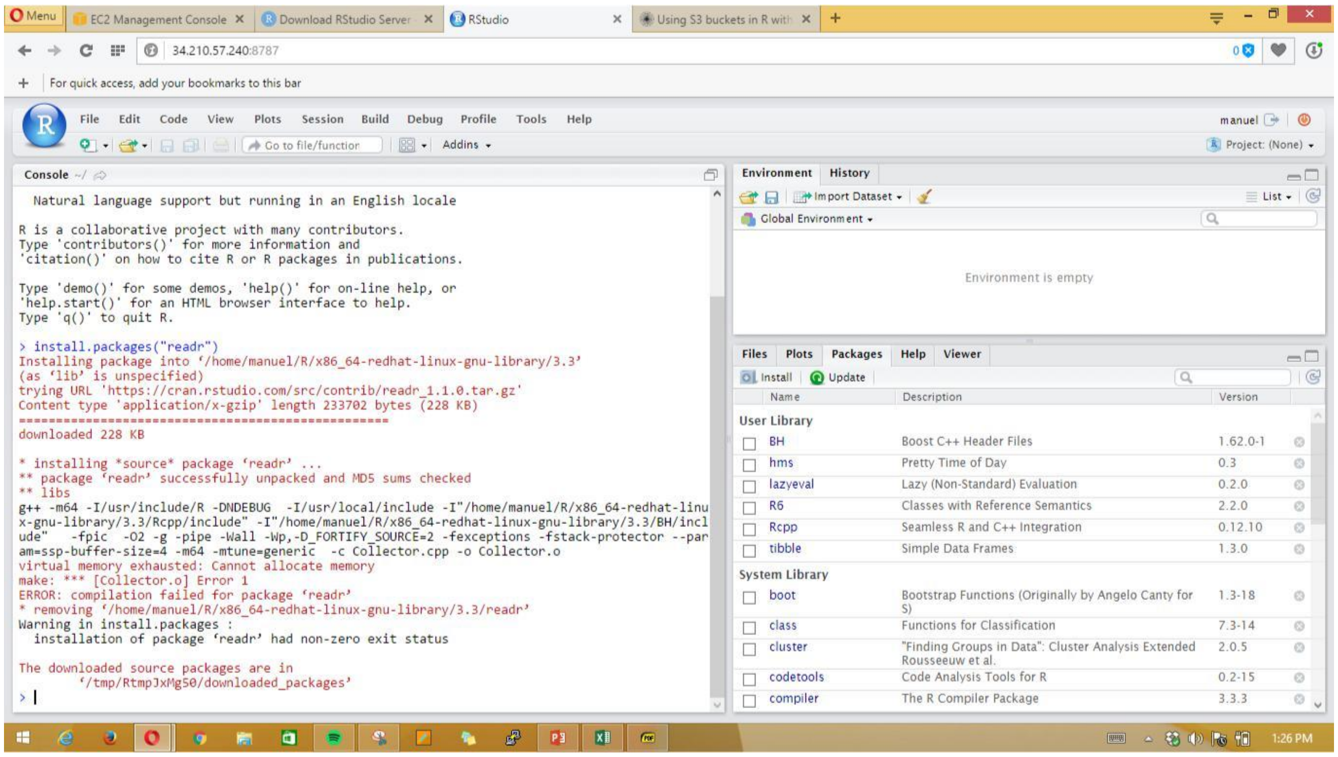
From this query, it can be taken that more crime tends to occur during the holidays, New Year, for instance. Also, another thing to take away from this query is that care needs to be taken to insert the time of occurrence of each crime, because they can be useful. From the image, it is clear that most of the crime occurrences are labeled with NULL for time. This is either because they are not sure when the crime actually occurred, or they neglect to record the time due to negligence. However, this is very useful in crime prediction, so it is very important to record the time.

(4.3) Statistical Analysis

For this section, we'll be using the R programming language and the R studio interface through Amazon's Web Services. This allows us to import the data-set directly from S3 right onto the R Studio interface.

First, we need to start a server on AWS, also called an EC2 instance, and this is very simple to do. The next step is to choose your instance, and we chose the Amazon Linux AMI, which has a stable version of R in its repository. After which you are provided with several EC2 instance types to choose from, each capable of a certain memory-size and certain processing power. We chose the M4 instance type, since our data-set is huge, spanning nearly a decade of crime data.

Once completed, the rStdio would look like the image provided below.



```
  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages("readr")
Installing package into '/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/src/contrib/readr_1.1.0.tar.gz'
Content type 'application/x-gzip' length 233702 bytes (228 KB)
==================================================
downloaded 228 KB

* installing *source* package 'readr' ...
** package 'readr' successfully unpacked and MD5 sums checked
** libs
g++ -m64 -I/usr/include/R -DNDEBUG  -I/usr/local/include -I"/home/manuel/R/x86_64-redhat-linu
x-gnu-library/3.3/Rcpp/include" -I"/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3/BH/incl
ude"   -fpic  -O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector --par
am=ssp-buffer-size=4 -m64 -mtune=generic  -c Collector.cpp -o Collector.o
virtual memory exhausted: Cannot allocate memory
make: *** [Collector.o] Error 1
ERROR: compilation failed for package 'readr'
* removing '/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3/readr'
Warning in install.packages :
  installation of package 'readr' had non-zero exit status

The downloaded source packages are in
        '/tmp/RtmpJxMg50/downloaded_packages'
> 
```

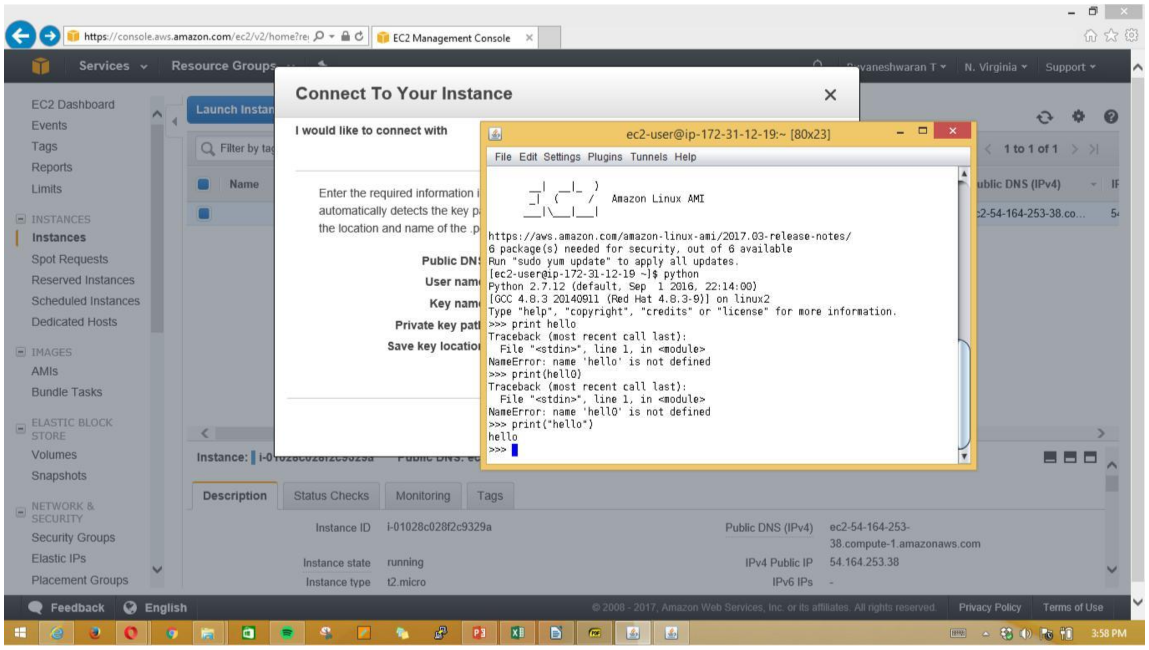
	Name	Description	Version
	User Library		
☐	BH	Boost C++ Header Files	1.62.0-1
☐	hms	Pretty Time of Day	0.3
☐	lazyeval	Lazy (Non-Standard) Evaluation	0.2.0
☐	R6	Classes with Reference Semantics	2.2.0
☐	Rcpp	Seamless R and C++ Integration	0.12.10
☐	tibble	Simple Data Frames	1.3.0
	System Library		
☐	boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-18
☐	class	Functions for Classification	7.3-14
☐	cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.0.5
☐	codetools	Code Analysis Tools for R	0.2-15
☐	compiler	The R Compiler Package	3.3.3

Illustration 6: Running RStudio via AWS's EC2 service

The advantages of using rStudio through AWS is that it is relatively fast, especially for machine learning purposes. However, it could get really expensive, so we tested our r-scripts on a local machine before testing it out on AWS.

We'll also be using python as it is one of the more popular tools for statistical analysis. The advantage of using python over AWS is the same as using R on AWS: it is comparatively faster in terms of processing speed, especially for larger and larger data-sets, depending on the type of instance or cloud machine chosen.

The steps to starting-up python on AWS is like that of starting R on AWS, with a few simple tweaks. Provided below is what the interface looks like.



```
     __|  __|_  )
     _|  (     /   Amazon Linux AMI
    ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
6 package(s) needed for security, out of 6 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-12-19 ~]$ python
Python 2.7.12 (default, Sep  1 2016, 22:14:00)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hello' is not defined
>>> print(hell0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hell0' is not defined
>>> print("hello")
hello
>>> 
```

Illustration 7: Running Python via AWS's EC2 service

(4.3.a) Correlation Matrix

Correlation is the statistical measure of the relationship strength between two assets. Correlation ranges between -1 and +1. Negative 1 is considered a perfectly negative correlation and thus the two attributes will move in a completely opposite direction. The positive one is considered a perfectly positive correlation. The correlation matrix helps you visualize the relationship between the various attributes in our data-set.

The below code basically takes in the data-set and determines how close with respect to time, week and month, each of the crimes occur. The results are outputted in the form of a correlation matrix.

```
library(dplyr)
library(ggplot2)
library(tidyr)
library(reshape2)
library(data.table)
library(DT)
library(d3heatmap)
library(lubridate)

library(xts)
library(dygraphs)

# Reading Input
d <- read.csv(choose.file(), header = T)
library(corrplot)
t <- d
t$Date <- as.POSIXct(t$Date)
t$month <- cut(t$Date, breaks= "month")
t$Week <- cut(t$Date, breaks= "week")
monthName <- function(x) strftime(x, '%b') # Jan, Feb etc.
t$month <- monthName(t$Date)
counts <- summarise(group_by(t,
Primary.Type,month),Counts=length(Primary.Type))
counts <- counts[order(counts$month),]
p <- dcast(counts,month ~ Primary.Type, value.var = "Counts" ) p[is.na(p)] <- 0
# Make month row names
row.names(p) <- p$month
# Remove first
p = p[,-1]
M <- cor(p)
corrplot(M, type = "upper", order = "hclust", tl.col = "black", tl.srt = 45,number.cex=0.75,tl.cex =

0.46)
```

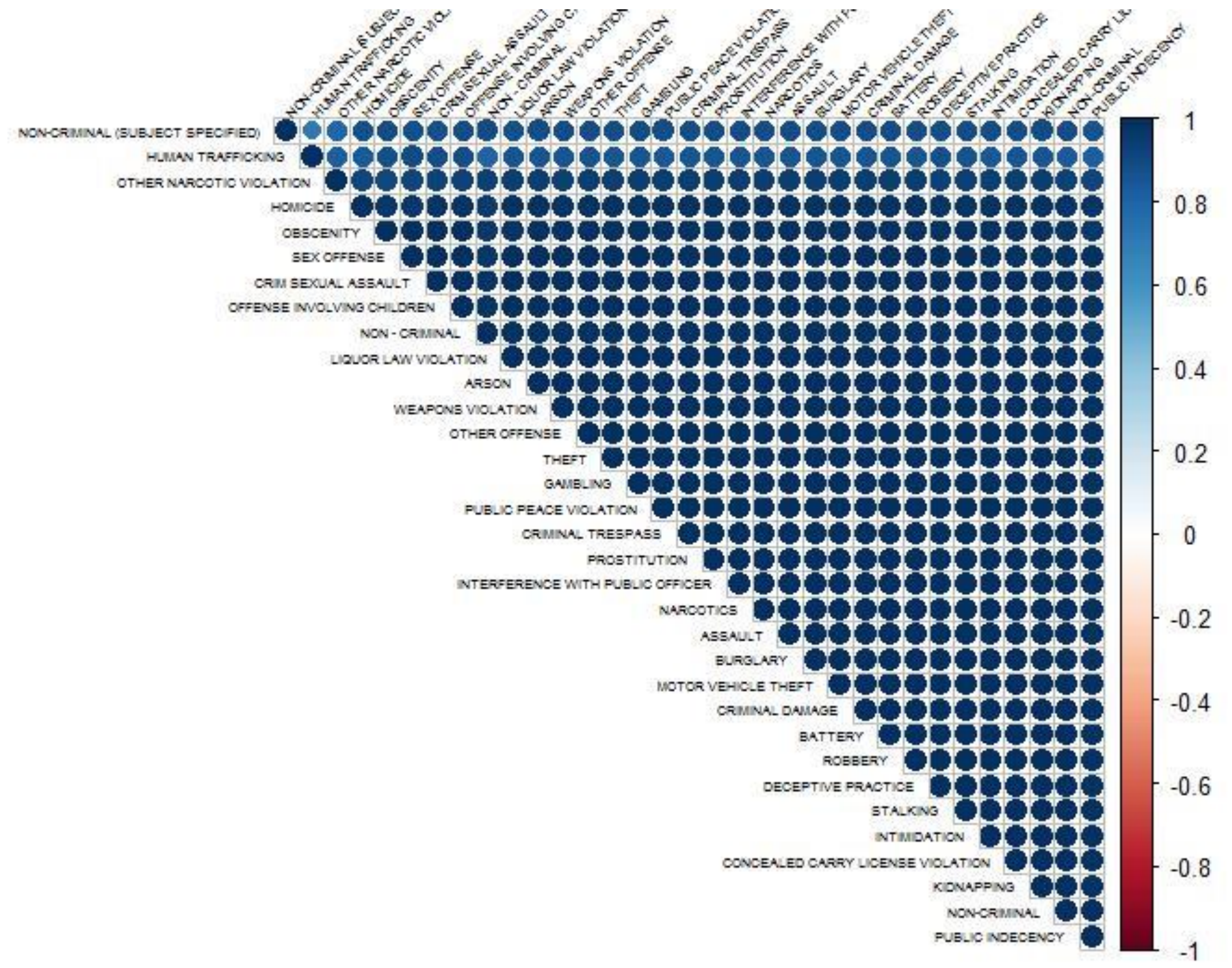


Illustration 8: Correlation Matrix for the Crime Types

From the output above, you can tell that most of the crimes tend to have high correlation. Which means an increase in crime for one category, means an increase in crime for another category. This correlation matrix shows the correlation of the various crimes with respect to time, i.e., Month and Week. What this means for our purposes is that an increase in a particular kind of crime means any other crime is also likely to occur.

(4.3.b) Coefficient of Variation

The coefficient of variation just compares the standard deviation to the mean. It is calculated as S.D./ Mean sometimes it is multiplied 100 to get a percentage. It is difficult to compare standard deviations of variables where the units are not the same. For example, if one variable had units in millions of dollars and another in cents how do you compare their variability? One way to compare them and get an idea of their variability is to use the CV.

The coefficient of variation is a measure of spread that describes the amount of variability relative to the mean. Because the coefficient of variation is unit-less, you can use it instead of the standard deviation to compare the spread of variables that have different units or different means. We will be using this to see how our various attributes vary with respect to other attributes like time, year, month, week, and so on. So first we need to import the important packages.

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

# Store start time to check script's run-time scriptStartTime =
time.time()
# Read file
df = pd.read_csv("../input/train.csv")

df["Dates"] = pd.to_datetime(df["Dates"])
```

For this project, we'll be using the SciPy ecosystem. SciPy is a package of Python libraries for mathematics, science and engineering. It is an add-on to Python that we'll use for machine learning and statistical analysis. The SciPy ecosystem is comprised of the following core modules relevant to machine learning:

Numpy: A foundation for SciPy that allows you to efficiently work with data in arrays.

Matplotlib: Allows you to create 2D charts and plots from data.

Pandas: Tools and data structures to organize and analyze your data. The package comes with more

stuff, but these three will suffice for our

project.

Below, we print out the amount of crimes per ‘Primary Type’.

```
# Amount of crimes per category
groups = df.groupby("Category")["Category"].count()
groups = groups.sort_values(ascending=0)
plt.figure()
groups.plot(kind='bar', title="Category Count")
print(groups)
```

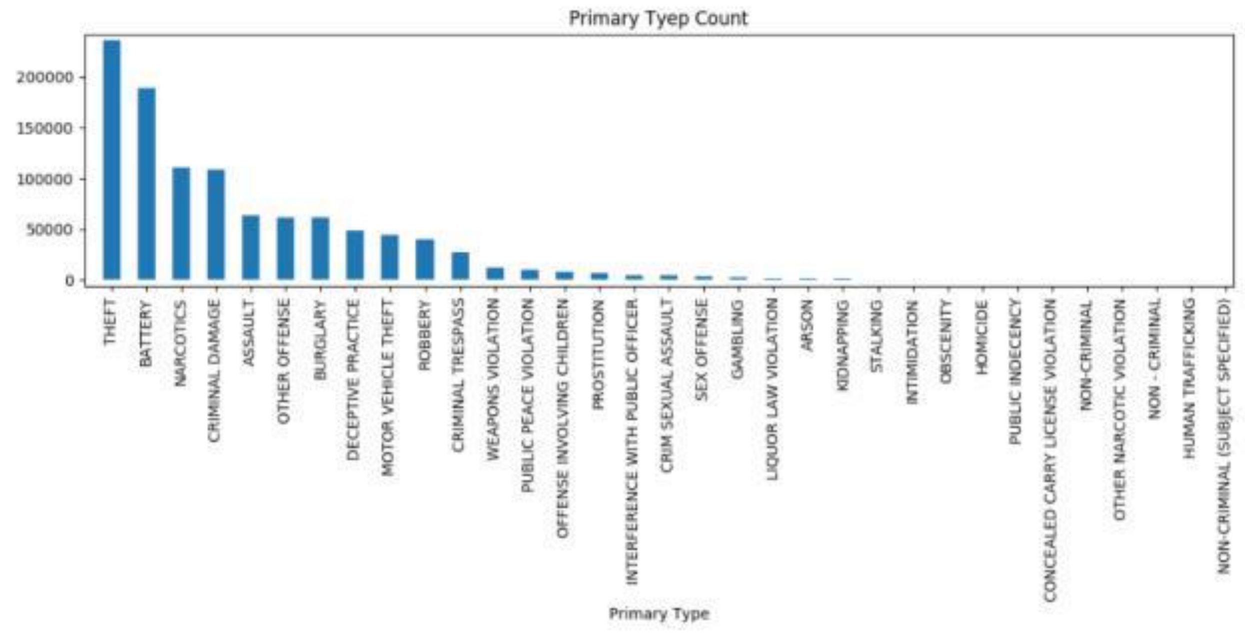


Illustration 9: Primary Type Count

Below is the code to compute the crime groups with highest per-day-CV.

```
#   Find the crime group with highest per-day-Coefficient Of Variation dayOfWeekVars =
pd.DataFrame(columns=["Primary Type", "CoefficientOfVariation"]) rows = []
for c in df["Primary Type"].unique(): dfSubset = df[df["Primary
        Type"] == c]
        dfSubsetGrouped = dfSubset.groupby("DayOfWeek")["Primary Type"].count() std =
        dfSubsetGrouped.std()
        mean = dfSubsetGrouped.mean() cv = std

        / mean

        #   Only consider category, if there are enough samples if (len(dfSubset) >
        300):
            rows.append({'Primary Type': c, 'CoefficientOfVariation': cv})
```

```
categoryDayCV = pd.DataFrame(rows).sort_values(by="CoefficientOfVariation", ascending=0)
#plt.figure()
categoryDayCV.plot(x="Primary Type", kind="bar", title="Category Day Coefficient Of Variation")
plt.show()
```

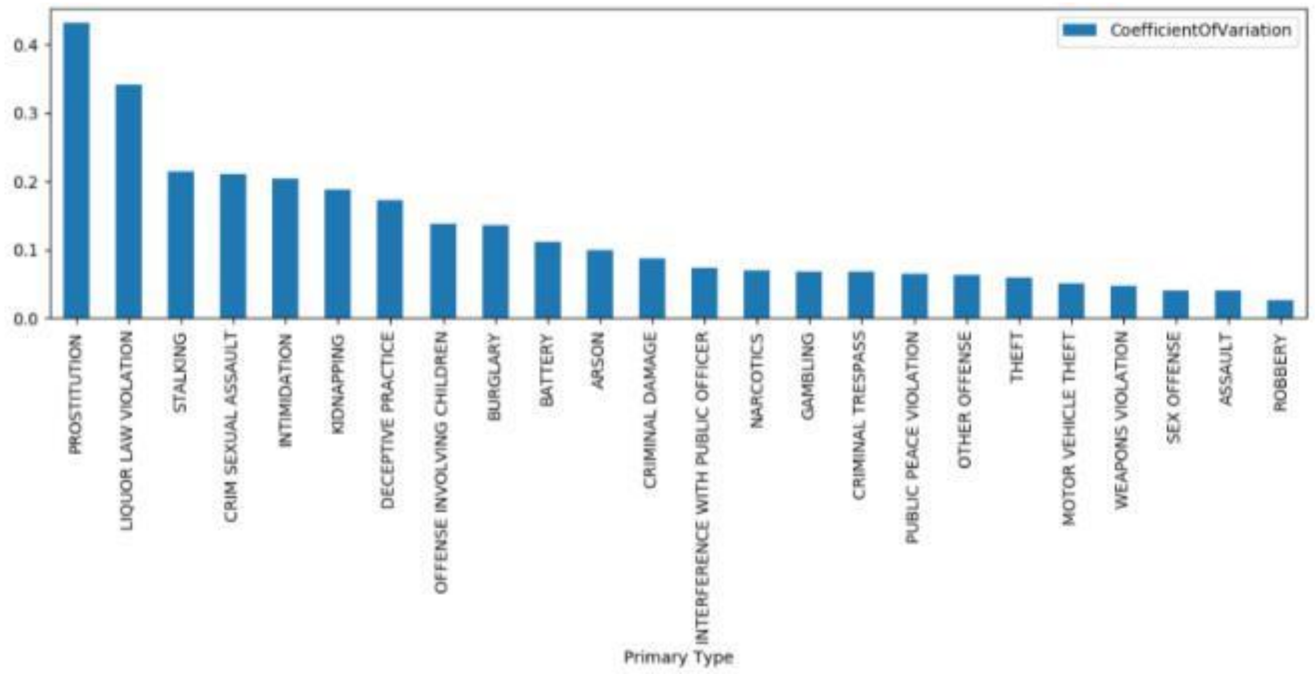


Illustration 10: Primary Type Coefficient of Variation by Day

Some of the categories appear to have a higher variation per day than others. To see the differences, plot the top five most varying categories per day.

```
for category in categoryDayCV["Primary Type"][:5]:
        dfCategory = df[df["Primary Type"] == category]
        groups = dfCategory.groupby("DayOfWeek")["Primary Type"].count()
        weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        groups = groups[weekdays]
        plt.figure()
        groups.plot(kind="bar", title=category + " count by day")
```

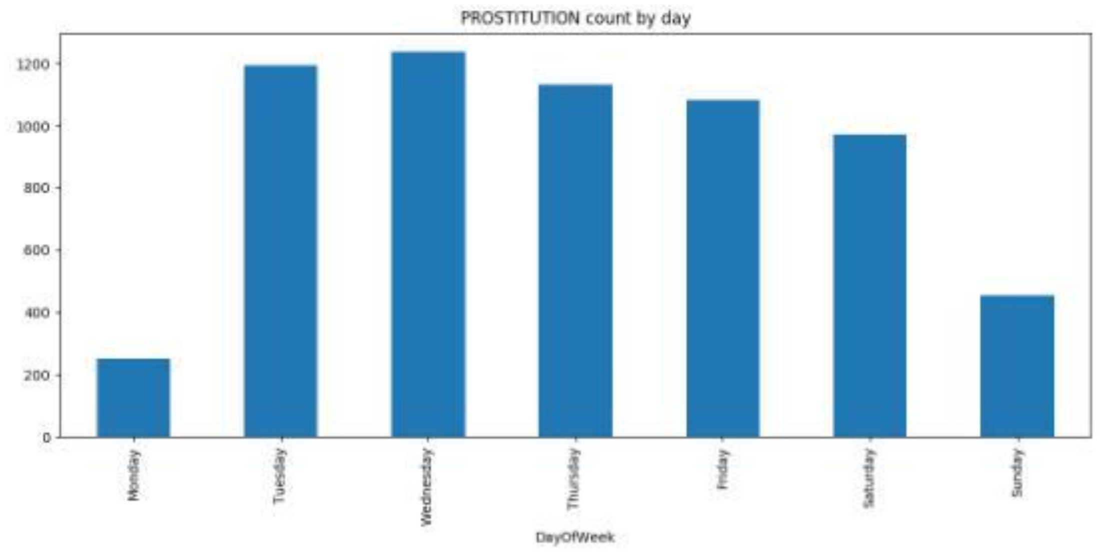


Illustration 11: Prostitution Count by Day

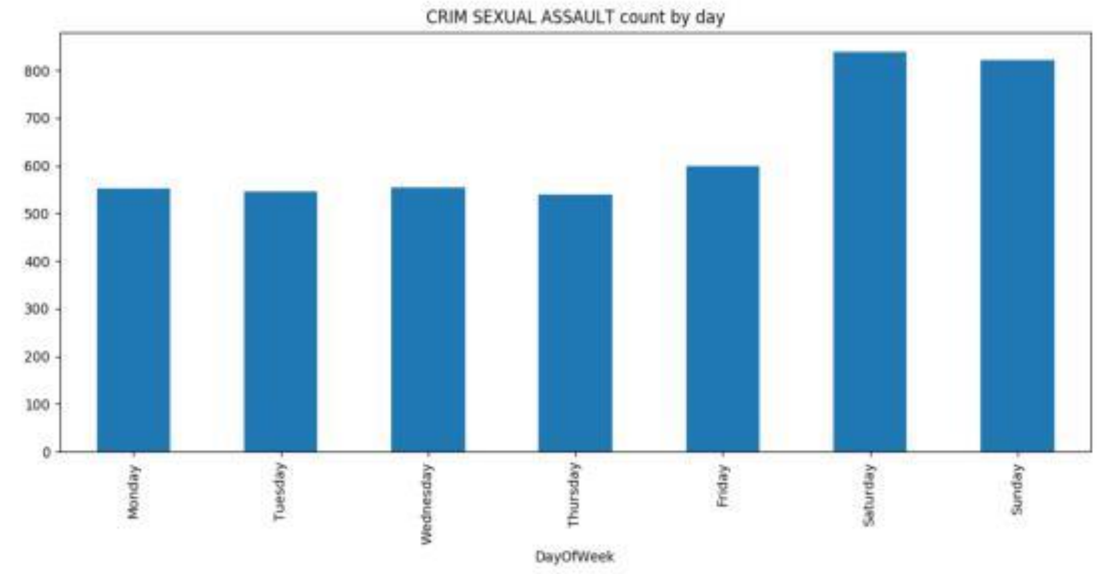


Illustration 12: Sexual Assault Count by Day

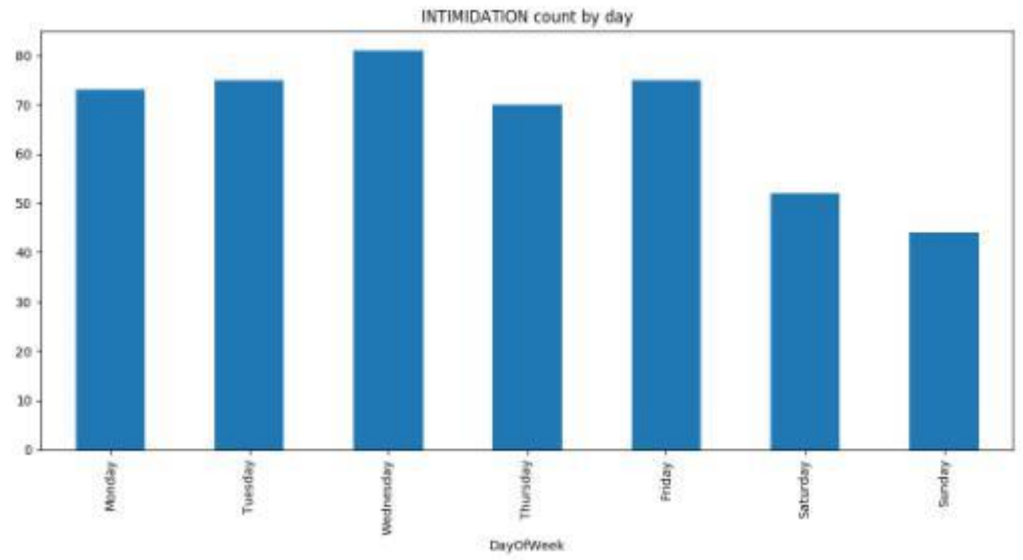


Illustration 13: Intimidation Count by Day

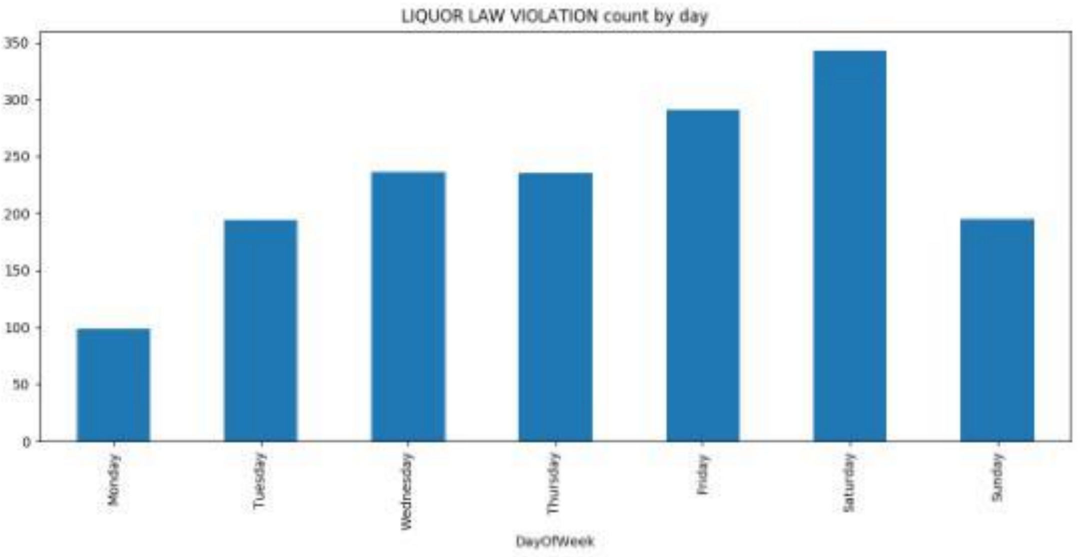


Illustration 14: Alcohol Law Violation Count by Day

All of the above crime tends to follow a trend, they tend to follow the normal distribution to some degree. We are only able to find out which crime followed this distribution through the concept of Coefficient of Variation.

Prostitution tends to occur towards the weekdays; Sexual assault tends to occur towards the weekends. Intimidation tends to occur on the weekdays; whereas, Liquor law tends to occur towards the weekends. These trends are very useful for law enforcement to effectively target the crimes and prevent them.

We are not limited to computing the CV of crime categories with respect to only the day of week, however. We can compute the CV with respect to Hour, Block, District, and so on. All of which can help uncover trends that Cops could use to keep public safe.

We'll now plot the CV of crime with respect to time. Provided below is the code we wrote.

```
dayOfWeekVars = pd.DataFrame(columns=["Primary Type", "CoefficientOfVariation"]) rows = []
for c in df["Primary Type"].unique():
        dfSubset = df[df["Primary Type"] == c]
        dfSubsetGrouped = dfSubset.groupby("HOUR")["Primary Type"].count()
        std = dfSubsetGrouped.std()
        mean = dfSubsetGrouped.mean()
        cv = std / mean

        #  Only consider category, if there are enough samples if (len(dfSubset) >
        300):
                rows.append({'Primary Type': c, 'CoefficientOfVariation': cv})

categoryDayCV = pd.DataFrame(rows).sort_values(by="CoefficientOfVariation", ascending=0)
#plt.figure()
categoryDayCV.plot(x="Primary Type", kind="bar", title="Category Hour Coefficient Of Variation")
plt.show()
```

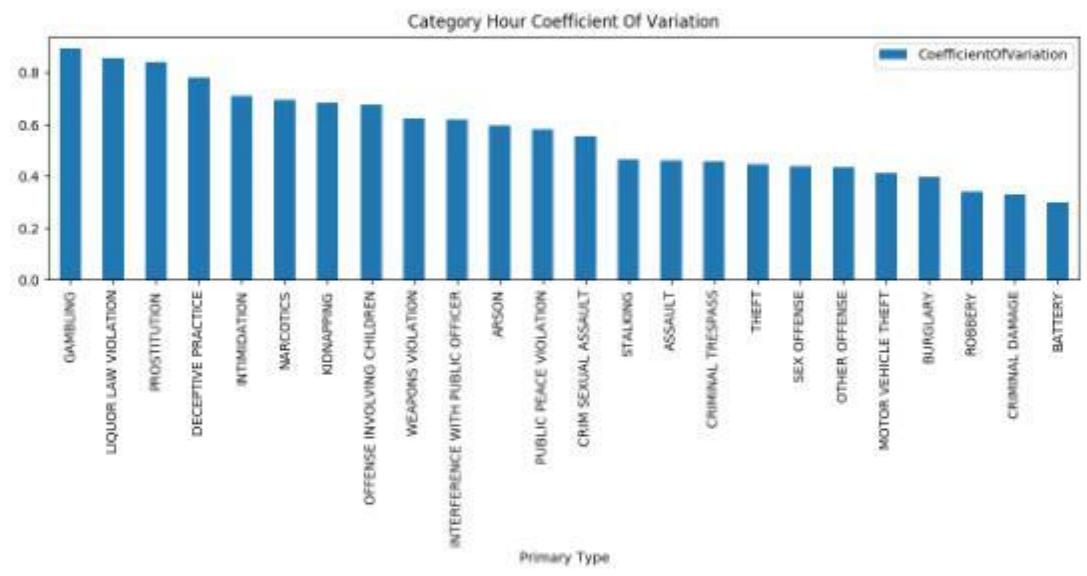


Illustration 15: Primary Type VS. CV by Hour

Now, let's plot the crime of gambling over the various hours of the day and see how they are distributed.

```
for category in categoryDayCV["Primary Type"][:1]:
        dfCategory = df[df["Primary Type"] == category]
        groups = dfCategory.groupby("HOUR")["Primary Type"].count()
        hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,15, 16, 17, 18, 19, 20, 21, 22, 23,0]
        groups = groups[hours]
        plt.figure()
                groups.plot(kind="bar", title=category + " count by Hour")
```

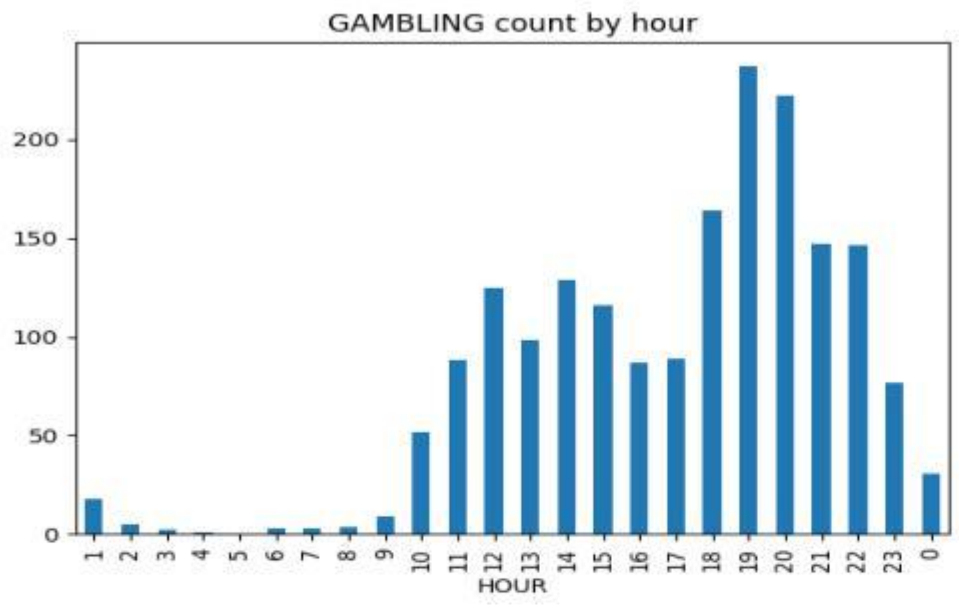


Illustration 16: Gambling Count by Hour

As can be seen, gambling tends to occur more towards the first half of the night and reduces gradually thereafter. So, a cop can look at this and be prepared to tackle gambling towards the night than they are towards the morning hours.

We'll do this one more time and call it quits for this section. Let's find the CV of crime with respect to district and block.

```
dayOfWeekVars = pd.DataFrame(columns=["Primary Type", "CoefficientOfVariation"]) rows = []
for c in df["Primary Type"].unique():
        dfSubset = df[df["Primary Type"] == c]
        dfSubsetGrouped = dfSubset.groupby("District")["Primary Type"].count()
        std = dfSubsetGrouped.std()
        mean = dfSubsetGrouped.mean()
        cv = std / mean

        #   Only consider category, if there are enough samples if (len(dfSubset) >
        300):
                rows.append({'Primary Type': c, 'CoefficientOfVariation': cv})

categoryDayCV = pd.DataFrame(rows).sort_values(by="CoefficientOfVariation", ascending=0)
#plt.figure()
categoryDayCV.plot(x="Primary Type", kind="bar", title="Category District Coefficient Of Variation")
            plt.show()
```

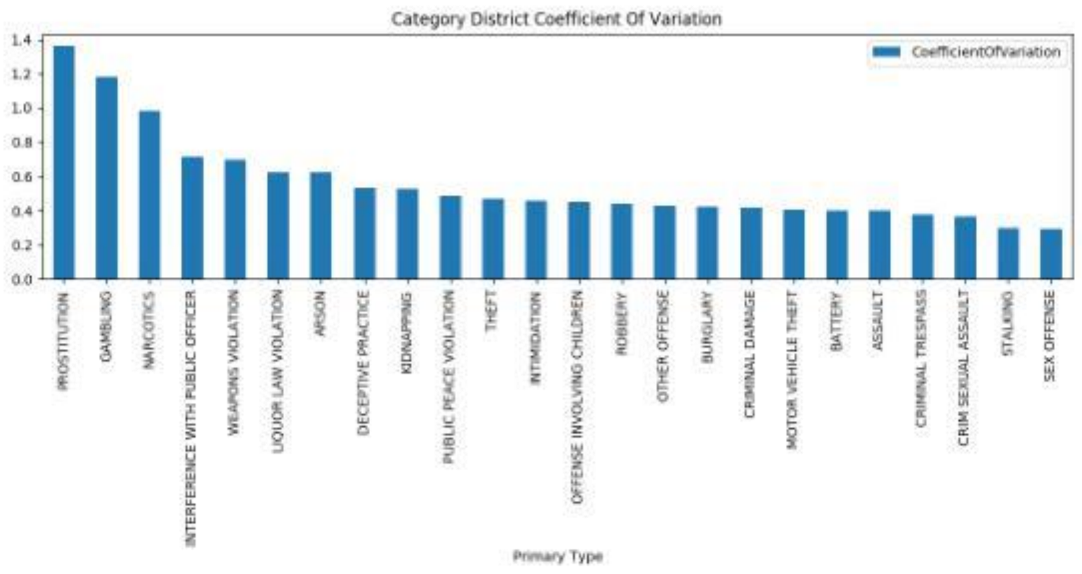


Illustration 17: Primary Type Vs. CV by District

As can be seen prostitution tends to differ with respect to district by a lot. Let's see how prostitution is actually distributed over the various districts.

```
for category in categoryDayCV["Primary Type"][:1]:
        dfCategory = df[df["Primary Type"] == category]
        groups = dfCategory.groupby("District")["Primary Type"].count()
        district = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,15, 16, 17, 18, 19, 20, 21, 22, 23, 25]
        groups = groups[district]
        plt.figure()
        groups.plot(kind="bar", title=category + " count by District")
```

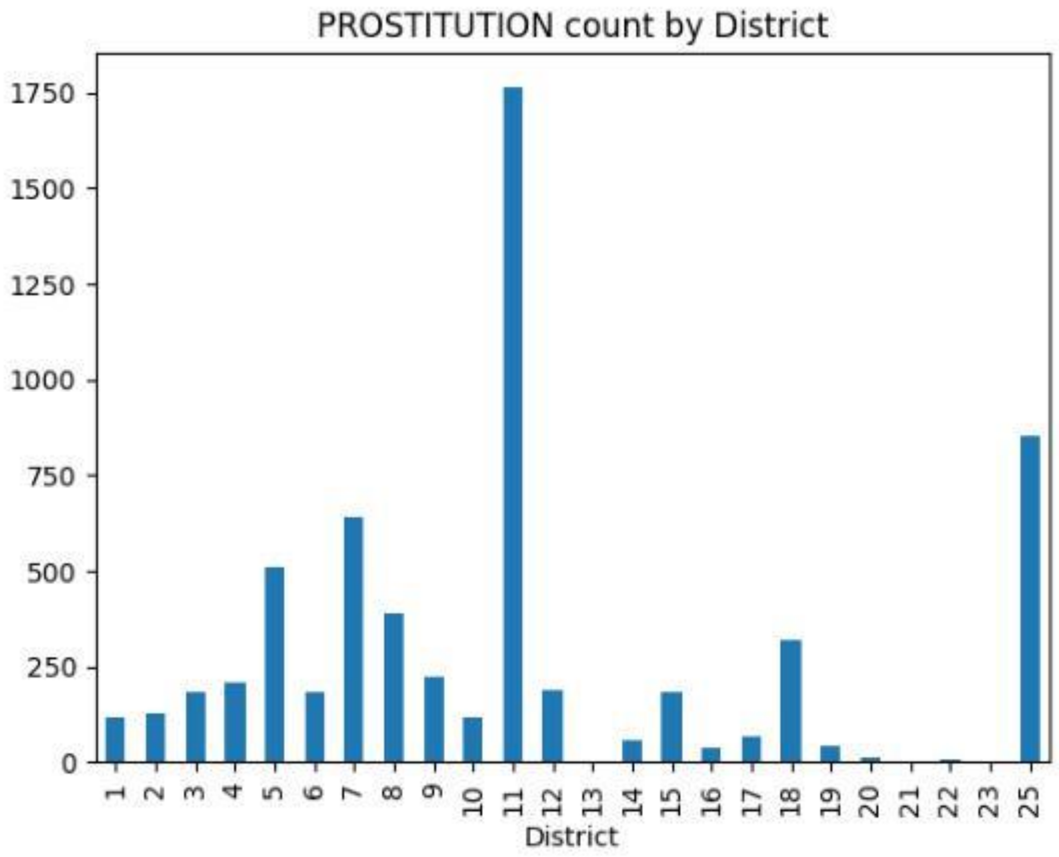


Illustration 18: Prostitution Count by District

So, let's plot the distribution of prostitution over the various blocks and districts using PowerBI. This will give a visual of the crime ridden areas that cops can use to target these prostitutes.

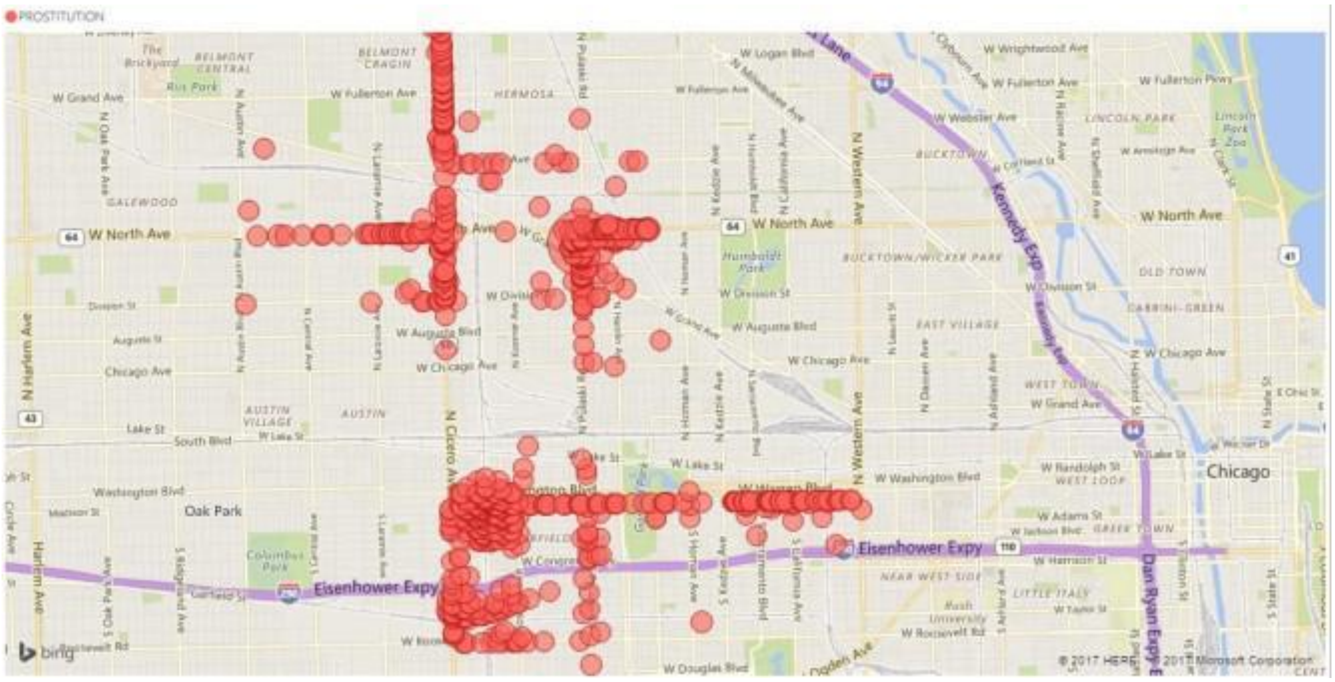


Illustration 19: Distribution of Prostitution

As you can see, prostitutes are rampant in the district 11 (the lower cluster) and district 25(the upper cluster).

Now, let's try and see how the various crimes differ by the months of year.

```
dayOfWeekVars = pd.DataFrame(columns=["Primary Type", "CoefficientOfVariation"]) rows = []
for c in df["Primary Type"].unique():
        dfSubset = df[df["Primary Type"] == c]
        dfSubsetGrouped = dfSubset.groupby("Months")["Primary Type"].count()
        std = dfSubsetGrouped.std()
        mean = dfSubsetGrouped.mean()
        cv = std / mean

        #   Only consider category, if there are enough samples if (len(dfSubset) >
        300):
                rows.append({'Primary Type': c, 'CoefficientOfVariation': cv})

categoryDayCV = pd.DataFrame(rows).sort_values(by="CoefficientOfVariation", ascending=0)
#plt.figure()
categoryDayCV.plot(x="Primary Type", kind="bar", title="Category District Coefficient Of Variation")
            plt.show()
```

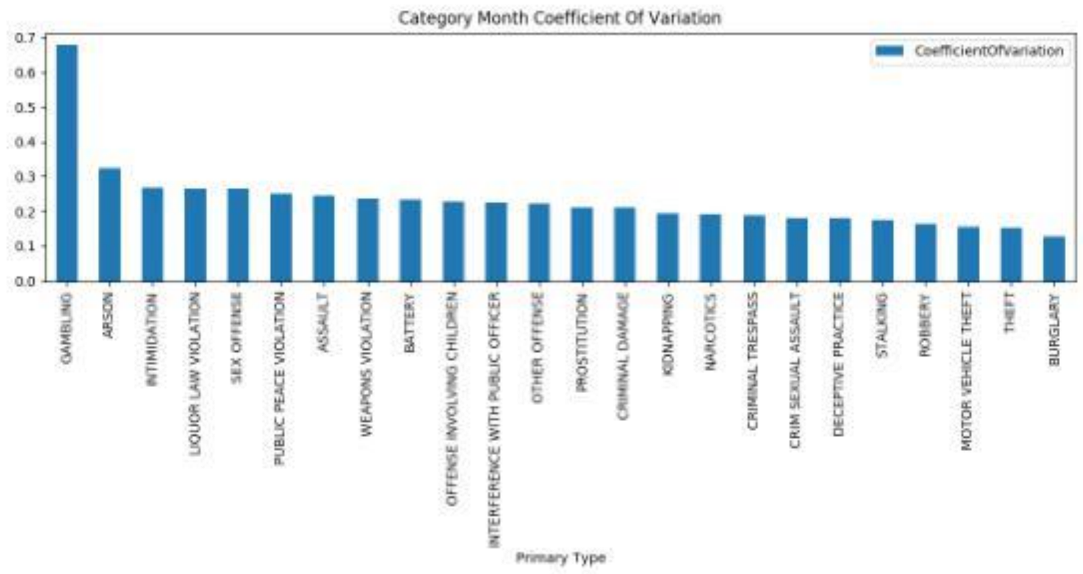


Illustration 20: Primary Type VS. CV by Month

Now, let's see how the Gambling crimes vary with the various months of the year.

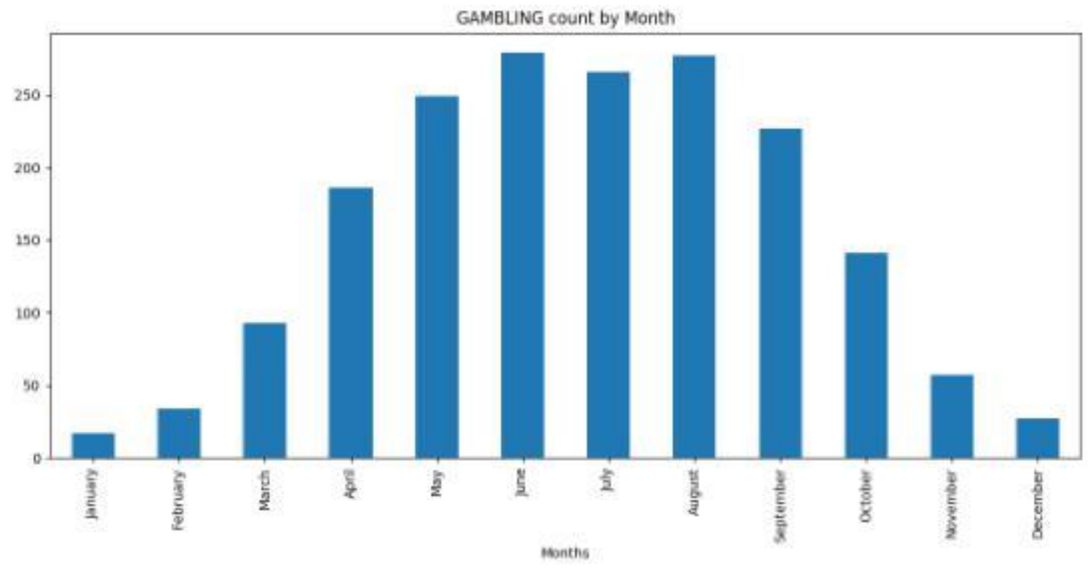


Illustration 21: Distribution of Gambling by Month

WOW!! This is almost a perfect normal distribution!! This would a treasure for law enforcement to effectively recognize the times of the month were these kinds of crimes are likely to occur.

All the visuals so far intuitively make up a rule, which will show up explicitly when using machine learning algorithms, which is our next section.

5. Evaluate Algorithms

We'll be using the AWS instance where we installed R to train and test our data-set on machine learning algorithms. This is particularly useful for this section, because machine learning algorithms takes days if not weeks to run on a personal computer with data-sets as large as this. By increasing our instances' cores to a size that is capable of handling such tasks, machine learning algorithms can be evaluated on large data-sets within minutes.

There are two considerations when evaluating algorithms:

1. Define your Test Harness
2. Spot-Checking Algorithms

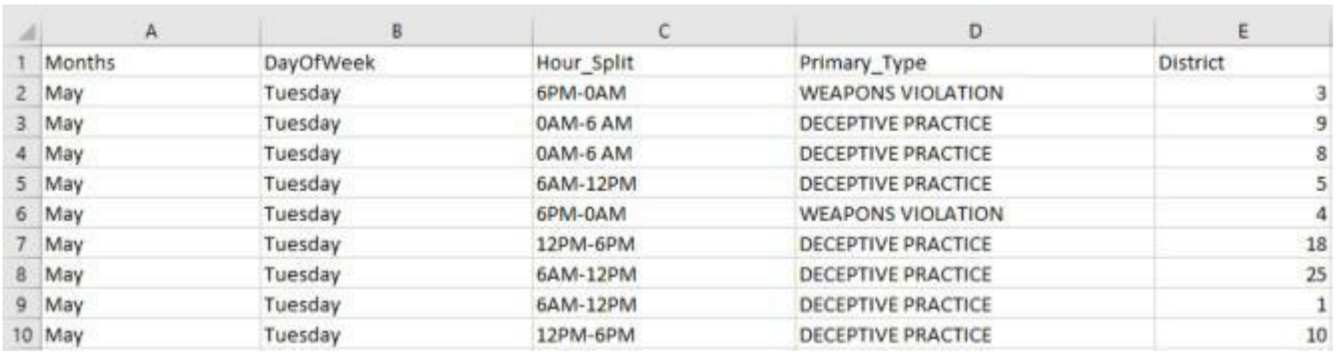
(5.1) Define your Test Harness

(5.1.a) Test Harness

The test harness is the data you will train and test an algorithm against and the measure(s) you will use to assess its performance. The goal of the test harness is to be able to quickly and consistently test algorithms.

The goal of this phase is to come up with different structures of the original data-set.

We made a few calculations to our original data-set. First, we extracted the month and made a nominal attribute and assigned it a column. We did the same thing for the day of week. We then split the time into four intervals, each spanning six hours. This keeps the complexity of the rules as simple as possible. We also extracted out only the most variable crime types, as computed in the statistical analysis section. This district column is left as is. The rest of the attributes present in the original data-set have been removed.



	A	B	C	D	E
1	Months	DayOfWeek	Hour_Split	Primary_Type	District
2	May	Tuesday	6PM-0AM	WEAPONS VIOLATION	3
3	May	Tuesday	0AM-6 AM	DECEPTIVE PRACTICE	9
4	May	Tuesday	0AM-6 AM	DECEPTIVE PRACTICE	8
5	May	Tuesday	6AM-12PM	DECEPTIVE PRACTICE	5
6	May	Tuesday	6PM-0AM	WEAPONS VIOLATION	4
7	May	Tuesday	12PM-6PM	DECEPTIVE PRACTICE	18
8	May	Tuesday	6AM-12PM	DECEPTIVE PRACTICE	25
9	May	Tuesday	6AM-12PM	DECEPTIVE PRACTICE	1
10	May	Tuesday	12PM-6PM	DECEPTIVE PRACTICE	10

Illustration 22: Altered Dataset

(5.1.b) Test Options

For our project, we'll use the same data-set to train and test the algorithm's performance. This option is chosen because it is comparatively faster than other options, especially considering the size of this data-set which spans hundreds-of-thousands of records. Moreover, a split test on a large data-set has been shown to produce accurate estimate of the actual performance of the algorithm.

First, the data is split into two parts: a training set, wherein you train the data-set on that algorithm, generating a decision tree, and a validation set, where you take the results and apply it on another data-set to validate.

Also, once it is validated, you can prune your tree and refine your tree and apply it on the test data again.

If you have, say, thousand data points, I might take 600 for building that tree or I might take 300 for validation and after refining I'll apply it on 100, and I want to see how well this tree functions. Often you see Data Scientists don't separate the validation and test data. They will just have one data-set that does both validation and test. This will be our option for this study.

However, the drawback is that if we split the training data-set into a different split proportion than we may get a different result from our algorithm. This is called model variance, which we'll keep an eye on in our project.

(5.2) Spot-Check Algorithms

Now that we've defined our test harness, our aim is to discover algorithms that are best able to learn from this data-set, and we provided each algorithm a fair chance to perform well. For this project, we'll primarily use association-rule based algorithm, Apriori, and classification rules-based algorithm, CART etc.

(5.2.a) Apriori

Many a people refer to association rules as affinity rules.

Association rules take on this format: It is very simple. If somebody has preference for "A", then "C". Accordingly, "A" refers to antecedent, something that comes before, and "C" refers to consequent, something that comes after.
For example, if I buy bread, then I'm likely to buy cheese. Or if I buy this particular book, I'm likely to buy another book, so on and so forth.

This is often called market basket analysis. So, if you buy a whole bunch of things in a basket, the question is what do you buy together, and the understanding of that can be used for pricing, placement, and even inventory management.

Now, let's look at the basics of association rules. One of the parameters is called the confidence, how much confidence you have in the rule "if A Then C". Basically, Confidence is nothing but probability of C given A. If somebody buys A, what is the likelihood of them buying C. Given a database with large number of transactions that has bought A and lot of transactions that have bought A and C, then, the confidence for that rule is number of records that has both A and C divided by number of transactions that has A.

Now a lot of rules that you get may not be very meaningful. But you want to look at interesting ones. What are the interesting ones? For example, if I buy say IPhone, what is the likelihood of me buying and IPad. The likelihood of buying may be 10 times more or 1.5 times, so on and so forth.

How do we know if something is interesting? We need to know the probability of C, the consequent, which is called the benchmark confidence. So basically, the number of transactions that have C, divided by number of transactions in the database.

For the question of how relevant it is? We come up with a whole notion called lift ratio. What does this lift ration mean? I know there's this probability of somebody buying C, what happens to the likelihood of buying C if they already bought A, and how much more likely. This is called the lift ratio. Mathematically written as p(C/A)/P(C). This is nothing but your confidence over benchmark confidence. Anytime this number is 1 or more, we know the likelihood is actually going up so it is useful for us.

```
library(arules)
library(arulesViz)

tt<-Book1

tt$District<-factor(tt$District)
tt$Months<-factor(tt$Months)
tt$DayOfWeek<-factor(tt$DayOfWeek)
tt$Hour_Split<-factor(tt$Hour_Split)
tt$Primary_Type<-factor(tt$Primary_Type)

rules = apriori(tt, parameter = list(supp=0.1, conf=0.8, minlen=4))

rules = sort (rules, by="lift")
```

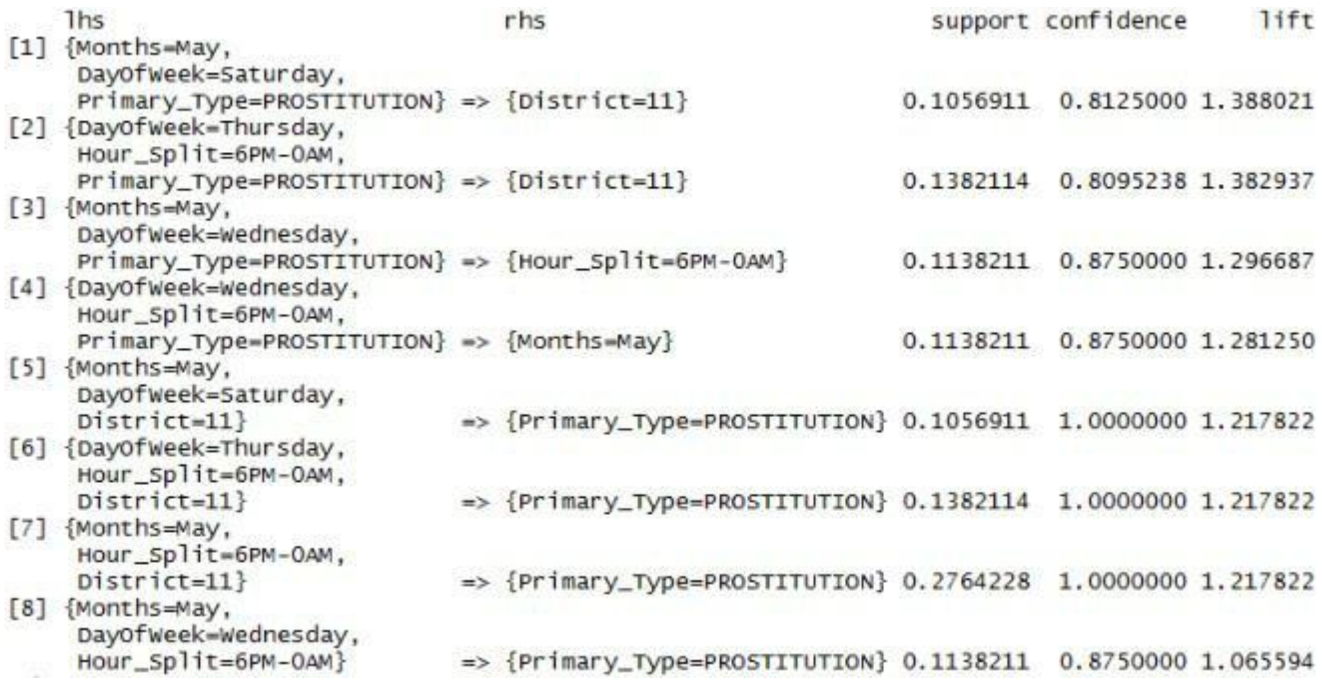
inspect(rules)

Accordingly, we got that file now, and the only thing you need to know is, what is the support level, which means what fraction of the transaction should support this rule. So, you can say 10 percent. Less than that means we'll get more rules.

Then you need to know some confidence; confidence, let's say 50 percent minimum for our purposes. Confidence basically means how much confidence do u have in that rule or how accurate that rule is for the transactions that it supports. In other words, out of all the records that support this rule, what percentage of it is accurate.

If somebody buys A what is the likelihood of buying C, in other words.

Here's the output.



```
    lhs                                  rhs                              support confidence     lift
[1] {Months=May,
     DayOfWeek=Saturday,
     Primary_Type=PROSTITUTION} => {District=11}                   0.1056911  0.8125000 1.388021
[2] {DayOfWeek=Thursday,
     Hour_Split=6PM-0AM,
     Primary_Type=PROSTITUTION} => {District=11}                   0.1382114  0.8095238 1.382937
[3] {Months=May,
     DayOfWeek=Wednesday,
     Primary_Type=PROSTITUTION} => {Hour_Split=6PM-0AM}            0.1138211  0.8750000 1.296687
[4] {DayOfWeek=Wednesday,
     Hour_Split=6PM-0AM,
     Primary_Type=PROSTITUTION} => {Months=May}                    0.1138211  0.8750000 1.281250
[5] {Months=May,
     DayOfWeek=Saturday,
     District=11}               => {Primary_Type=PROSTITUTION} 0.1056911  1.0000000 1.217822
[6] {DayOfWeek=Thursday,
     Hour_Split=6PM-0AM,
     District=11}               => {Primary_Type=PROSTITUTION} 0.1382114  1.0000000 1.217822
[7] {Months=May,
     Hour_Split=6PM-0AM,
     District=11}               => {Primary_Type=PROSTITUTION} 0.2764228  1.0000000 1.217822
[8] {Months=May,
     DayOfWeek=Wednesday,
     Hour_Split=6PM-0AM}        => {Primary_Type=PROSTITUTION} 0.1138211  0.8750000 1.065594
```

Illustration 23: Apriori Rules for the Altered Dataset

The results were outputted very quickly, considering the size of the data-set. This is the power of using AWS for machine learning. What would normally take hours, is now made possible in a fraction of the time, several minutes in our case.

(5.2.b) CART

In this section, we'll discuss classification tree. Classification tree is part of a much larger problem called classification, and the classification and regression tree is often called CART. CART is a very popular method not only because it is easily comprehensible, but also because they are in the form of a tree which can be converted into long rules.

Hence, the question is how do you come up with theses classification trees, and the answer is that the underlying classification is basically regression.

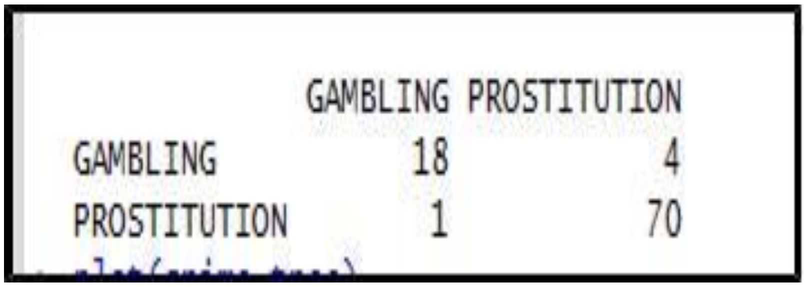
So how do you know whether you are doing a good job. So basically, you have a data-set, and you have a class attribute and the attribute has class-values, like, "yes" and "no", and you know in the training set a rule is developed, and you are applying that tree that you generated to validate or to test how well it does.

Let's now implement CART for our problem. This will allow us to do what's called a conditional inference tree, which provides an implementation of Brieman's random forests.

What We need to do is develop a classification tree where the leaf node or the nodes at the end of the tree represents the primary type. So, we want to make the decision based on various attributes of this data-set. As we discussed earlier, we want to classify this data-set for training this classification tree, the rest we're using for testing. If you recollect we have three types of sets, training, validation, and testing. But in this case, we'll be using just training and then build the classification tree and test it on the rest of the data, 30 percent of the data. In order to do this, you randomly select records from this data-set. So, you set the seed, I'll just give some arbitrary seed value. Why we do that is that we can replicate the same desert as long as we have the same seed.

Here is the code for our implementation.

```
Book1 <- read_csv("C:/Users/bt/Downloads/Book1.csv")
set.seed(1234)
tt<-Book1
tt$District<-factor(tt$District)
tt$Months<-factor(tt$Months)
tt$DayOfWeek<-factor(tt$DayOfWeek)
tt$Hour_Split<-factor(tt$Hour_Split)
tt$Primary_Type<-factor(tt$Primary_Type)
ind <- sample(2, nrow(tt), replace = TRUE, prob = c(0.7,0.3)) #make training data
and testing data
train.data<-tt[ind==1,]
test.data<-tt[ind==2,]
myf <- Primary_Type ~ Months+DayOfWeek+Hour_Split+District #pick set of attributes that has implication on
your classification crime_tree <- ctree(myf, data = train.data)
table(predict(crime_tree), train.data$Primary_Type) #make the matrix
plot(crime_tree)
```

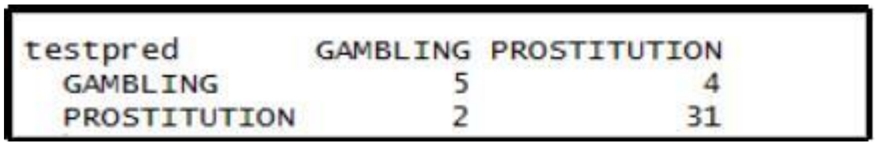


```
              GAMBLING PROSTITUTION
GAMBLING            18            4
PROSTITUTION         1           70
```

So, we basically created a table based on the tree that we have and the actual data that we have in the training data type. So, it gives out actual vs predicted. The higher the number in the diagonal from left-upper corner to the right-lower bottom, we have a good classifier. For example, in the first row, GAMBLING, out of 22 items that it has picked randomly, 18 is accurate, 4 is mis-classified. For the second row, PROSTITUTION, you have 1 mis-classified. So, you have about 5 data points that are in error compared to the actual. Not bad, maybe if we add more variables we might get more accurate data.

Now, let’s use this tree that we developed to predict the primary type from the test data, the 30% set aside for testing earlier.

```
Testpred ← predict(crime_tree, newdata=test.data)
table(testpred, test.data$Primary_Type)
```



```
testpred        GAMBLING PROSTITUTION
  GAMBLING             5            4
  PROSTITUTION         2           31
```

There are 9 data-points for GAMBLING, 4 is mis-classified. For the other stuff, 33 data-points, 2 mis-classified. There are 42 data-points, out of which 7 is in error. So again, we have 7/42 error rate. Which is not bad in any sense. If you can give any arbitrary content of the crime, you should be able to predict what type of crime it is.

Below is a tree diagram for the rule.

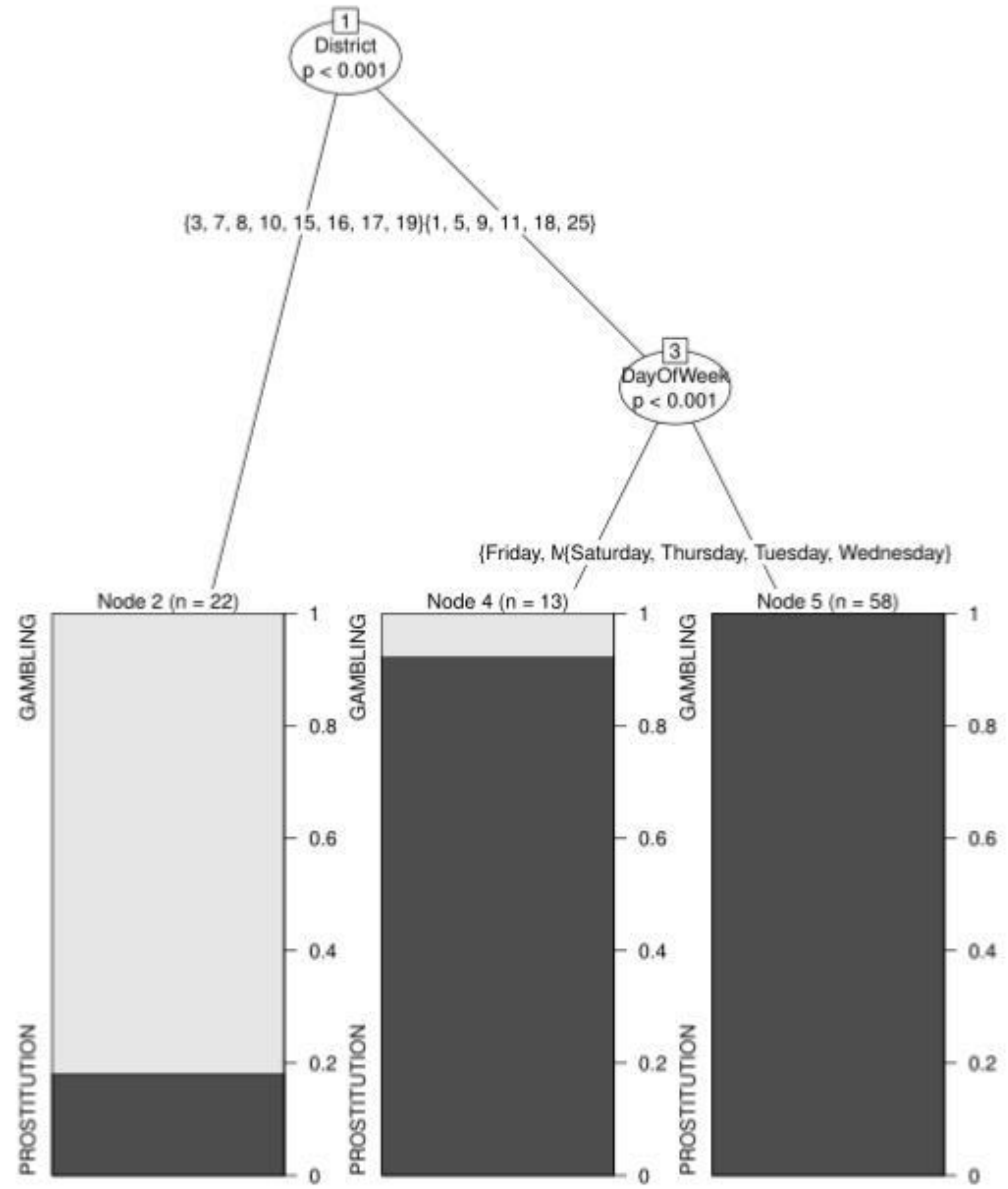
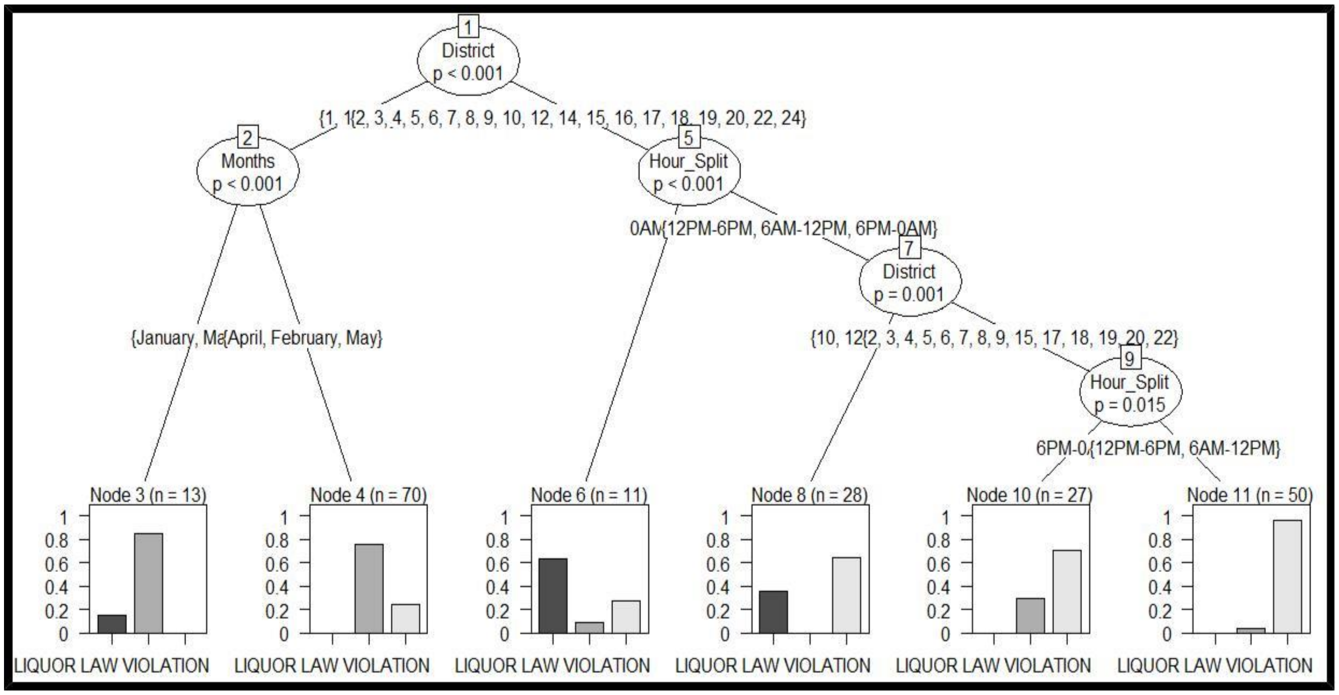
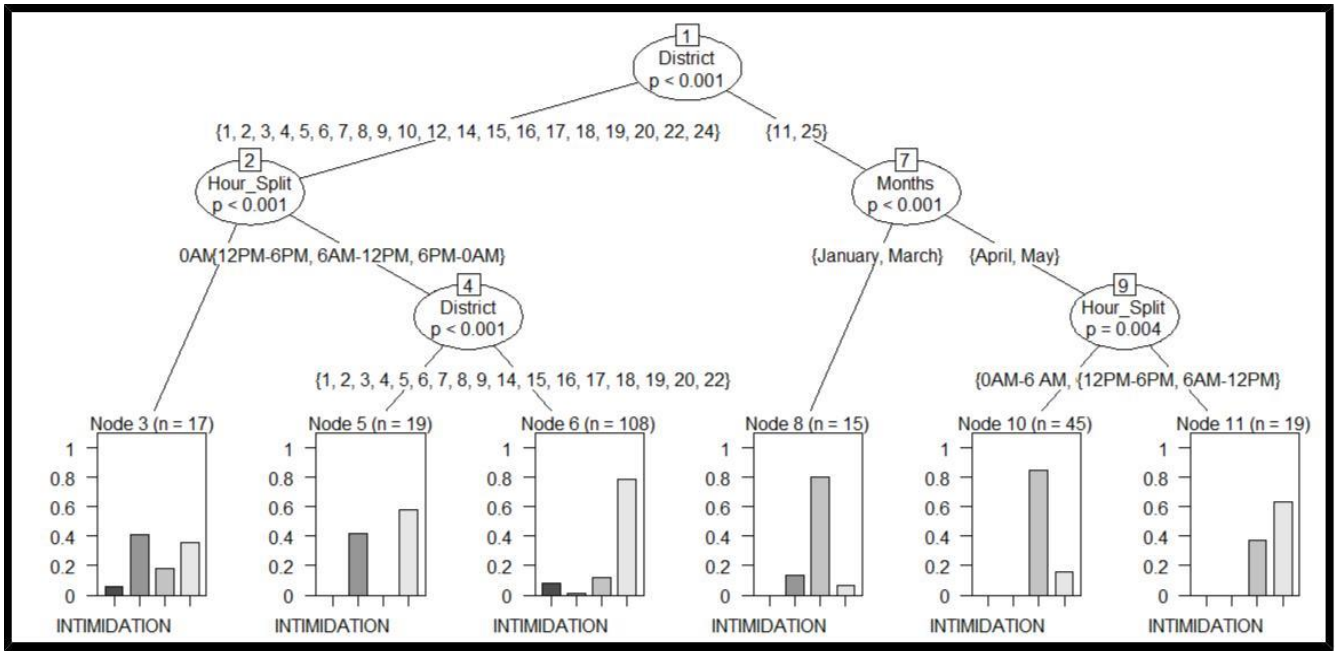


Illustration 24: Tree for the Altered Data-set

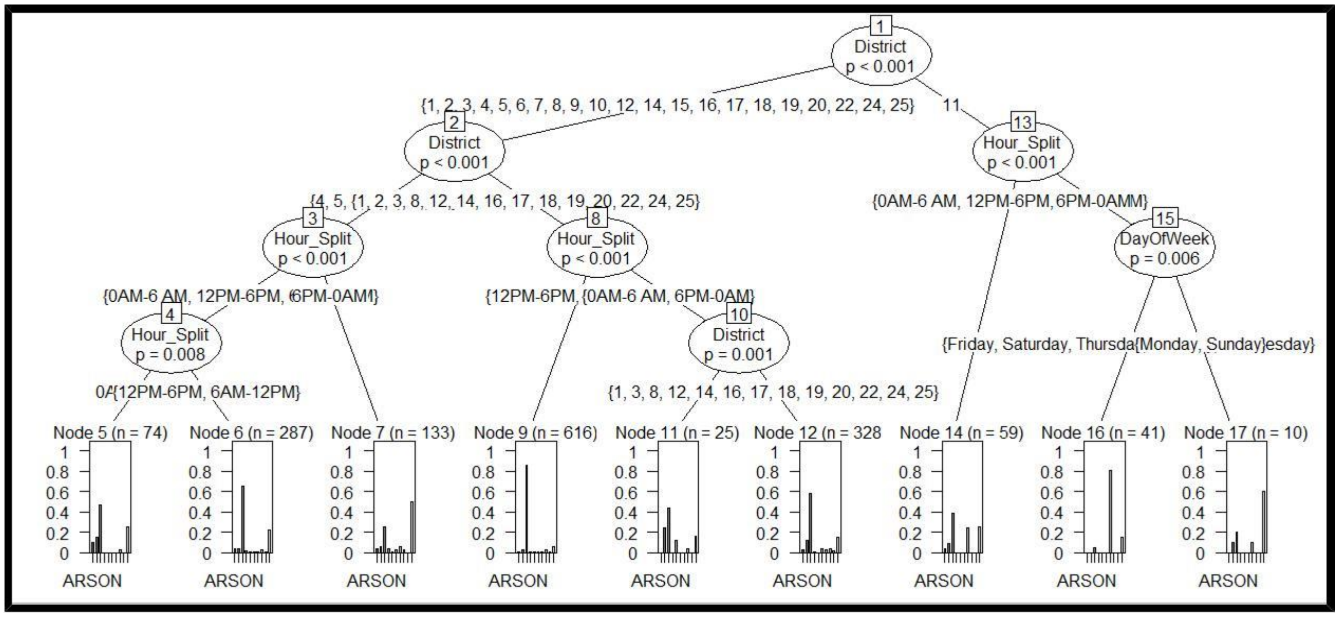
Below are various tress that we get from the various transformations of the data-set.



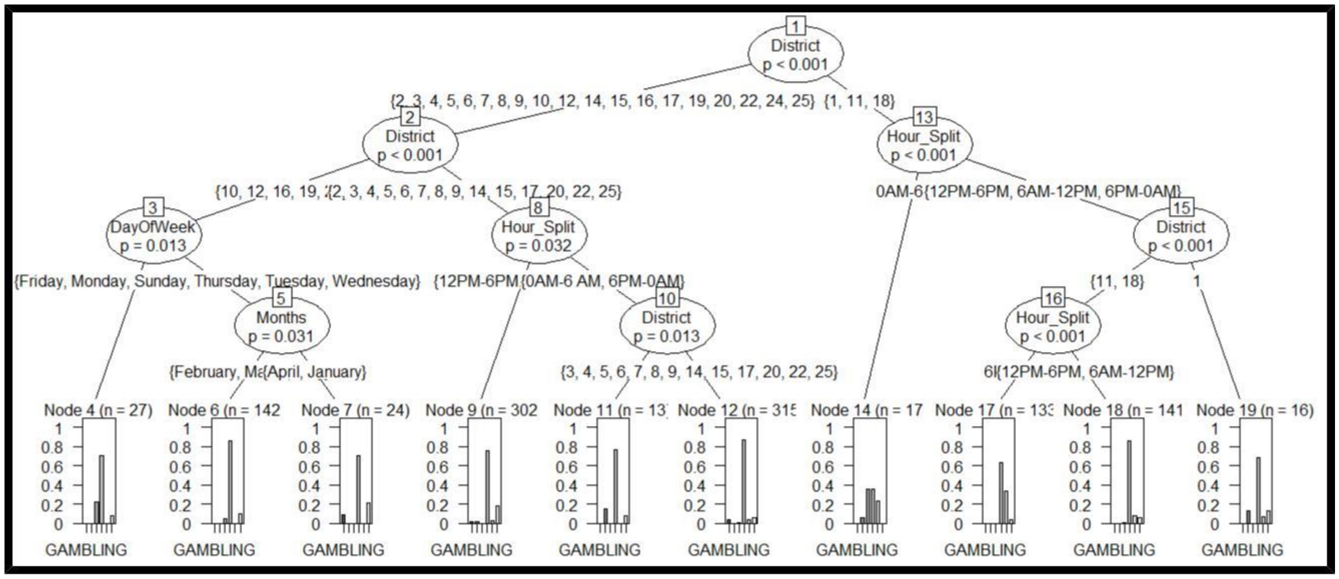
1 District p < 0.001
{1, 1{2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 22, 24}
2 Months p < 0.001
5 Hour_Split p < 0.001
{January, Ma{April, February, May}
0AM{12PM-6PM, 6AM-12PM, 6PM-0AM}
7 District p = 0.001
{10, 12{2, 3, 4, 5, 6, 7, 8, 9, 15, 17, 18, 19, 20, 22}
9 Hour_Split p = 0.015
6PM-0{12PM-6PM, 6AM-12PM}
Node 3 (n = 13)
Node 4 (n = 70)
Node 6 (n = 11)
Node 8 (n = 28)
Node 10 (n = 27)
Node 11 (n = 50)
LIQUOR LAW VIOLATION



1 District p < 0.001
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 22, 24}
{11, 25}
2 Hour_Split p < 0.001
7 Months p < 0.001
0AM{12PM-6PM, 6AM-12PM, 6PM-0AM}
4 District p < 0.001
{January, March}
{April, May}
9 Hour_Split p = 0.004
{1, 2, 3, 4, 5, 6, 7, 8, 9, 14, 15, 16, 17, 18, 19, 20, 22}
{0AM-6 AM, {12PM-6PM, 6AM-12PM}
Node 3 (n = 17)
Node 5 (n = 19)
Node 6 (n = 108)
Node 8 (n = 15)
Node 10 (n = 45)
Node 11 (n = 19)
INTIMIDATION



District p < 0.001
Hour_Split p < 0.001
DayOfWeek p = 0.006
Hour_Split p = 0.008
District p = 0.001
Node 5 (n = 74)
Node 6 (n = 287)
Node 7 (n = 133)
Node 9 (n = 616)
Node 11 (n = 25)
Node 12 (n = 328
Node 14 (n = 59)
Node 16 (n = 41)
Node 17 (n = 10)
ARSON



District p < 0.001
Hour_Split p < 0.001
DayOfWeek p = 0.013
Hour_Split p = 0.032
Months p = 0.031
District p = 0.013
Hour_Split p < 0.001
Node 4 (n = 27)
Node 6 (n = 142
Node 7 (n = 24)
Node 9 (n = 302
Node 14 (n = 17
Node 18 (n = 141
Node 19 (n = 16)
GAMBLING

(5.3) Interpret Results

From all of our observations, it became increasingly clear that association rules ought to work best for our particular situation. It is very hard to generalize the situation and come up with rules that predict the type of crimes for every occurrence; however, it is feasible to discover relations between variables in general in large databases. Thus, the Apriori algorithm is the best approach for this project.

6. Future Work

In the future we would like to experiment with other machine learning algorithms. Also, would like to spend more time fine-tuning each algorithm to make sure our algorithm is given a chance to perform its best. In addition, we would like to create transformations of the data-set in order to reveal the full structure of the data. Due to time constraint, we were not able to complete as much analysis as possible. Moreover, we would like to work on the data with even more years included in them. This analysis can be further carried out on a fully-distributed cluster mode, that which runs on multiple machines. Also, similar analysis can be carried out in different sectors.

7. References

1. https://aws.amazon.com/blogs/big-data/running-r-on-amazon-athena/
2. https://aws.amazon.com/blogs/big-data/running-r-on-aws/
3. https://aws.amazon.com/blogs/big-data/statistical-analysis-with-open-source-r-and-rstudio-on-amazon-emr/
4. http://machinelearningmastery.com

5. MachineLearningMaster.com

6. ~~https://www.youtube.com/watch?v=zJuFpqB01u4~~

7. ~~https://www.youtube.com/watch?v=M2Wc8JIS-p8~~

8. ~~https://www.youtube.com/watch?v=VncfC5GeqGs~~

9. https://en.wikipedia.org/wiki/Machine_learning

10. https://www.youtube.com/watch?v=Vm8ARd5SKLE

8. Acknowledgment & Division of Labor

Buvaneshwaran T:

First, big thanks to myself. The project would not be functional without my guidance and leadership. I showed tremendous patience when things were not going as planned and I believed in the idea right from the get-go. In addition, I was responsible for writing R/Python codes for statistical analysis and the queries, Hive and PIG. And I am responsible for setting up the AWS environment to run these queries, and hence, I am responsible for all the intellectually tiresome division of the project. I also mapped out the design for the project and allocated or distributed the jobs for the other team member.

Thinh Huynh:

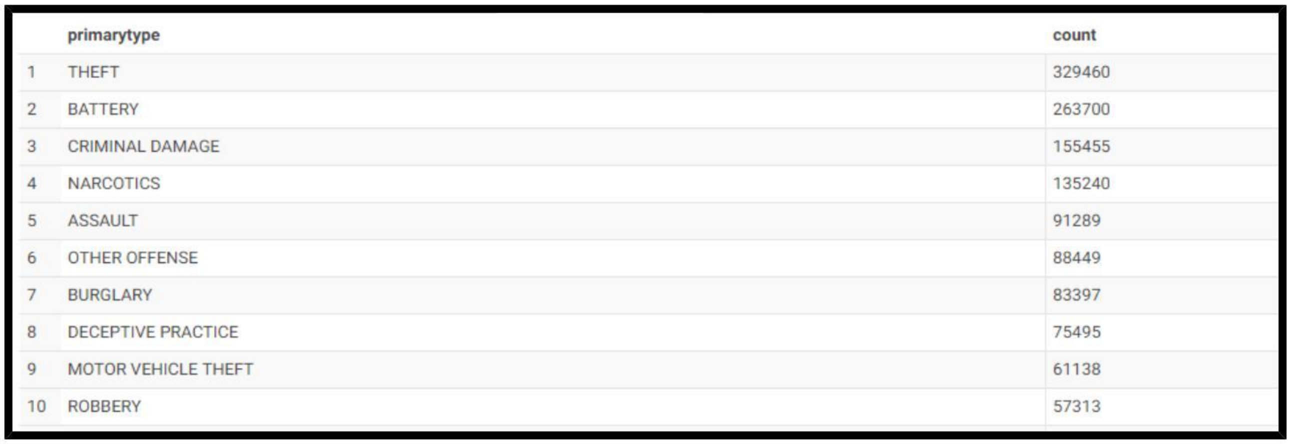
Deep thanks to Thinh Huynh, who said “yes” to every task assigned to him, and who made me very happy that he did. He endured long nights working to integrate my logic to his, and who did all of it while working full time as a Test Technician/Analyst at EMC/Dell and all while serving as a national guard at the Marine Corps. Thinh utilized his skills in the back-end of things; he worked on the Hive queries to generate interesting and eye-catching visuals. Thinh also played a managerial role in that he booked the daily, weekly, and sometimes nightly meetings to make sure the project is on track for completion.

9. Appendix

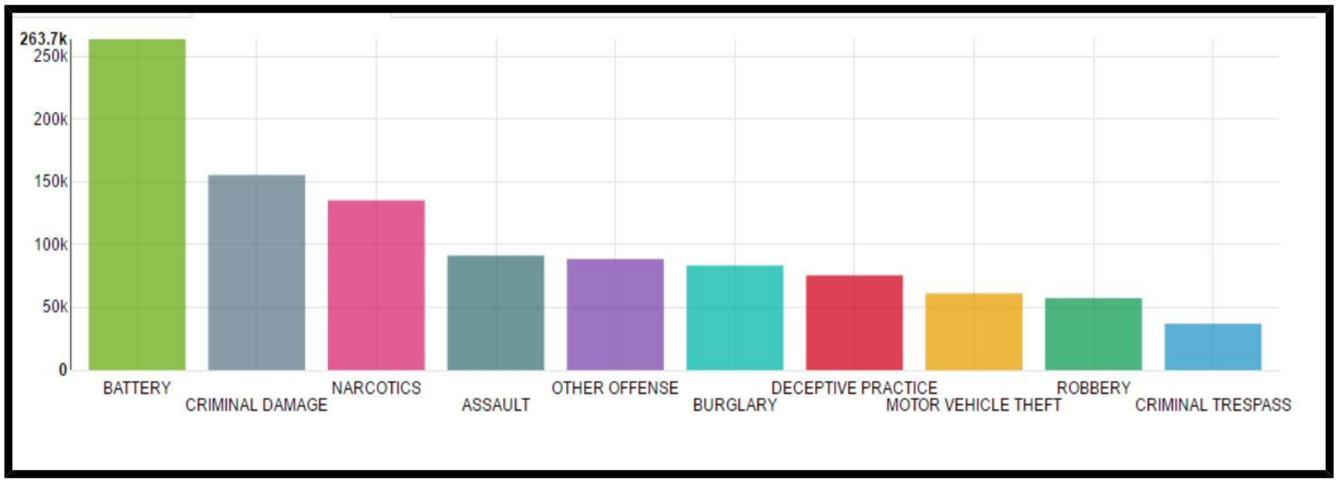
(9.1) Hive

1. **The most frequently** occurring **primary type.**

SELECT primaryType, COUNT(*) AS count FROM chicagoCrime2016 GROUP BY primaryType ORDER BY count DESC

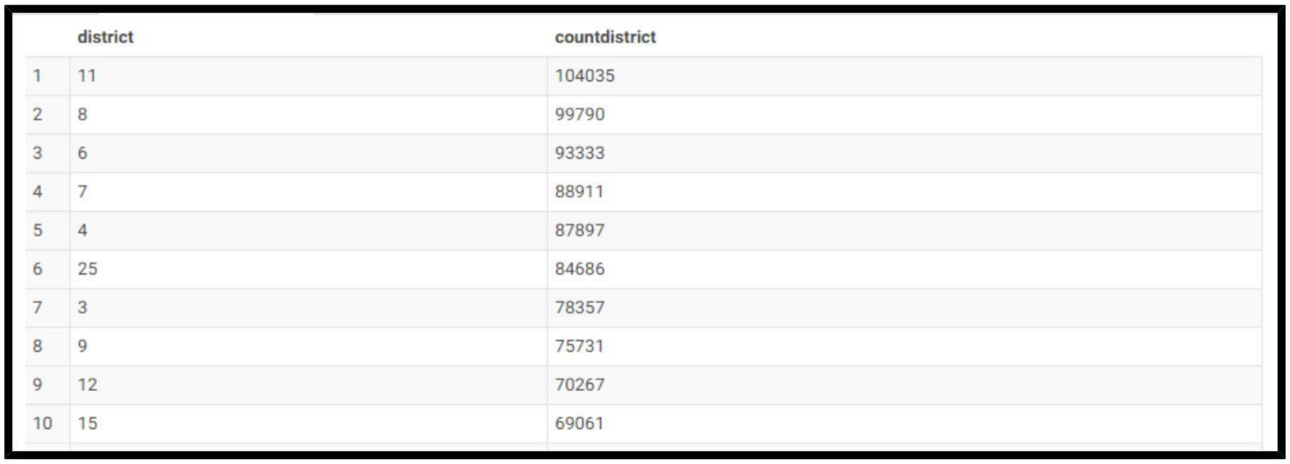


	primarytype	count
1	THEFT	329460
2	BATTERY	263700
3	CRIMINAL DAMAGE	155455
4	NARCOTICS	135240
5	ASSAULT	91289
6	OTHER OFFENSE	88449
7	BURGLARY	83397
8	DECEPTIVE PRACTICE	75495
9	MOTOR VEHICLE THEFT	61138
10	ROBBERY	57313

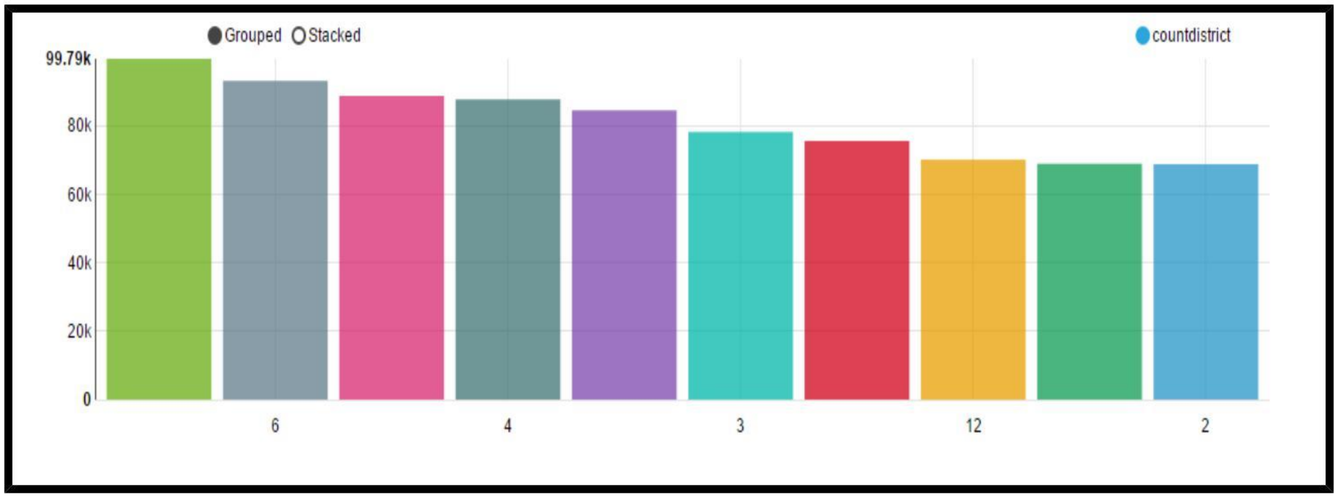


2. Districts with the most reported incidents.

SELECT District, COUNT(*) AS countDistrict FROM chicagoCrime2016 GROUP BY District ORDER BY countDistrict DESC

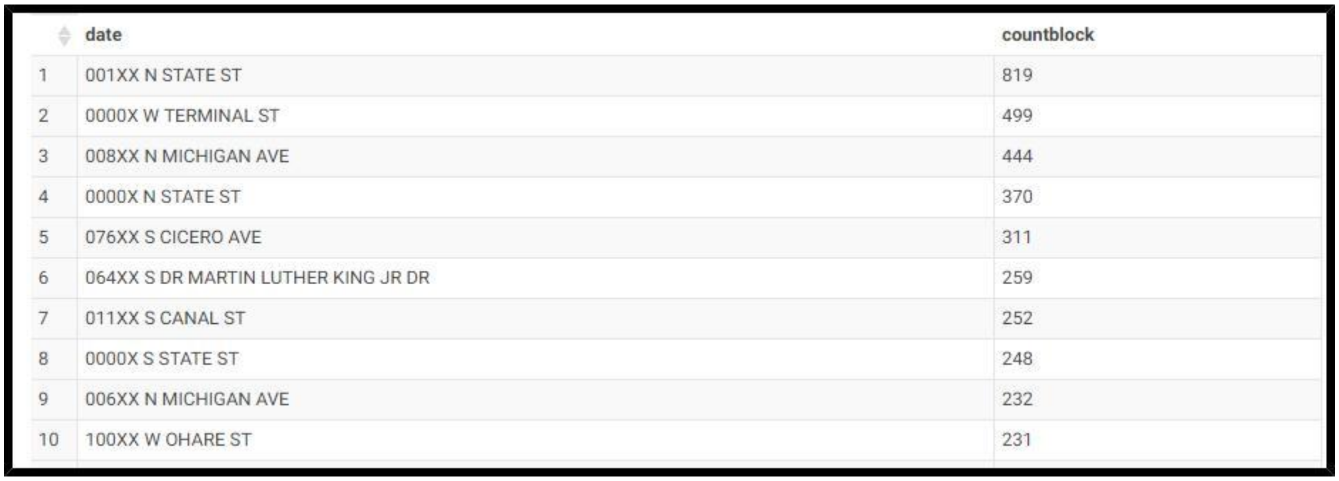


	district	countdistrict
1	11	104035
2	8	99790
3	6	93333
4	7	88911
5	4	87897
6	25	84686
7	3	78357
8	9	75731
9	12	70267
10	15	69061

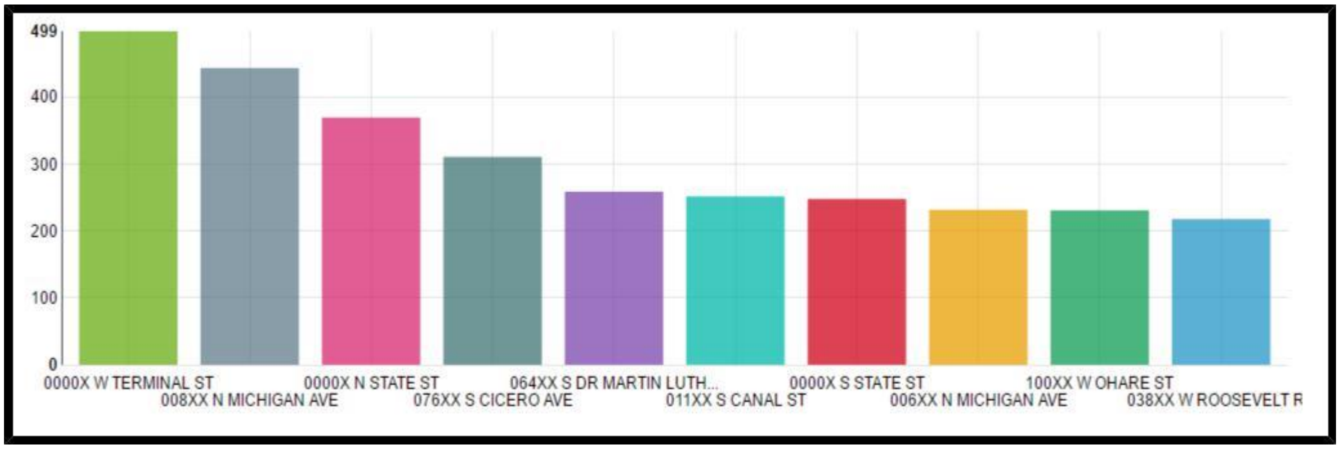


3. Blocks with the most reported incidents.

```
SELECT Block, COUNT(*) AS countBlock FROM chicagoCrime2016 GROUP BY Block

ORDER BY countBlock DESC
```

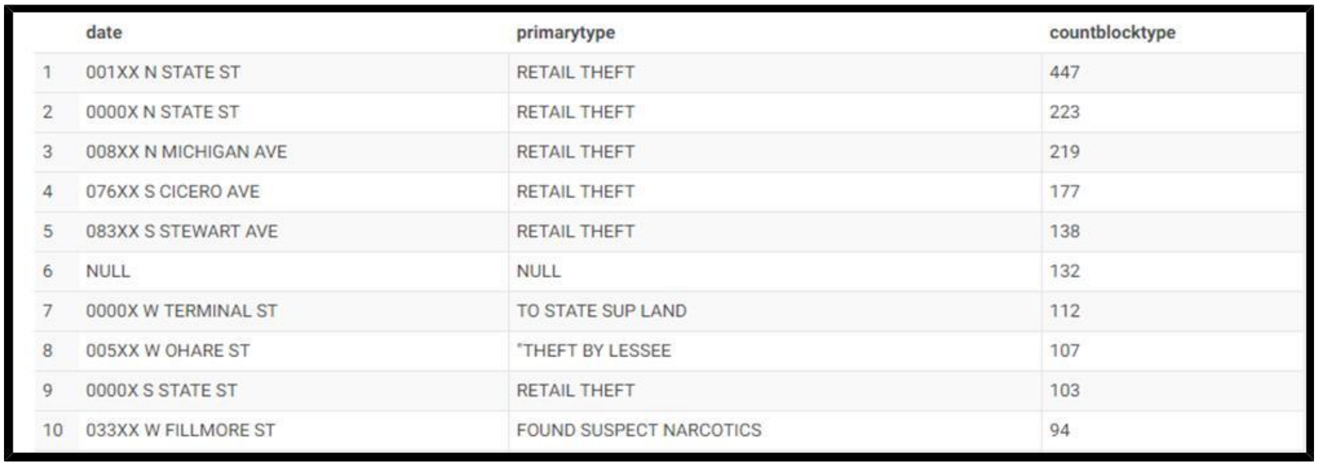


	date	countblock
1	001XX N STATE ST	819
2	0000X W TERMINAL ST	499
3	008XX N MICHIGAN AVE	444
4	0000X N STATE ST	370
5	076XX S CICERO AVE	311
6	064XX S DR MARTIN LUTHER KING JR DR	259
7	011XX S CANAL ST	252
8	0000X S STATE ST	248
9	006XX N MICHIGAN AVE	232
10	100XX W OHARE ST	231

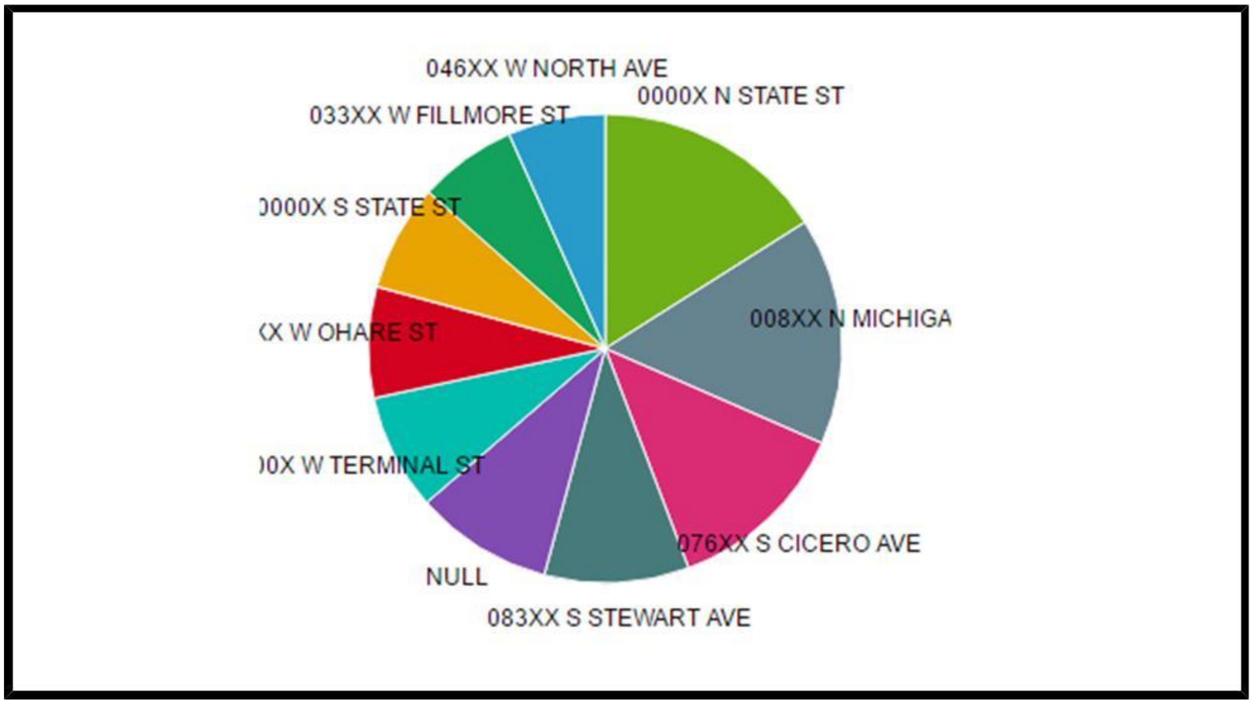


4. **Blocks with the most reported incidents, grouped by primary type.**

```
SELECT Block, primaryType, COUNT(*) AS countBlockType FROM chicagoCrime2016 GROUP BY

Block, primaryType ORDER BY countBlockType DESC
```

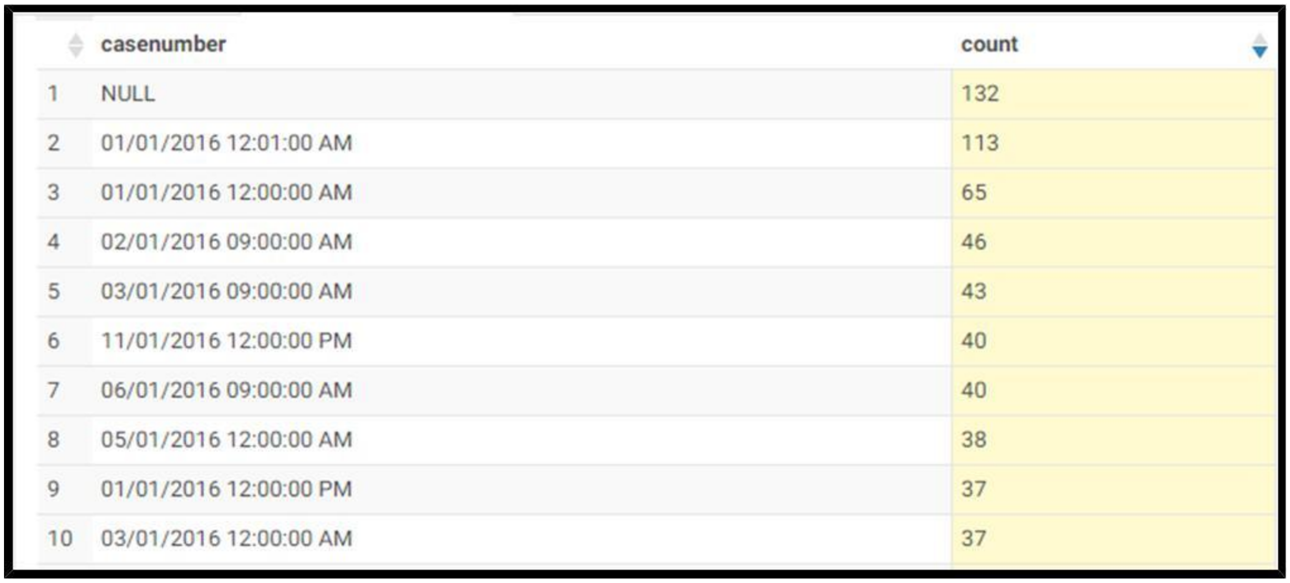


	date	primarytype	countblocktype
1	001XX N STATE ST	RETAIL THEFT	447
2	0000X N STATE ST	RETAIL THEFT	223
3	008XX N MICHIGAN AVE	RETAIL THEFT	219
4	076XX S CICERO AVE	RETAIL THEFT	177
5	083XX S STEWART AVE	RETAIL THEFT	138
6	NULL	NULL	132
7	0000X W TERMINAL ST	TO STATE SUP LAND	112
8	005XX W OHARE ST	"THEFT BY LESSEE	107
9	0000X S STATE ST	RETAIL THEFT	103
10	033XX W FILLMORE ST	FOUND SUSPECT NARCOTICS	94

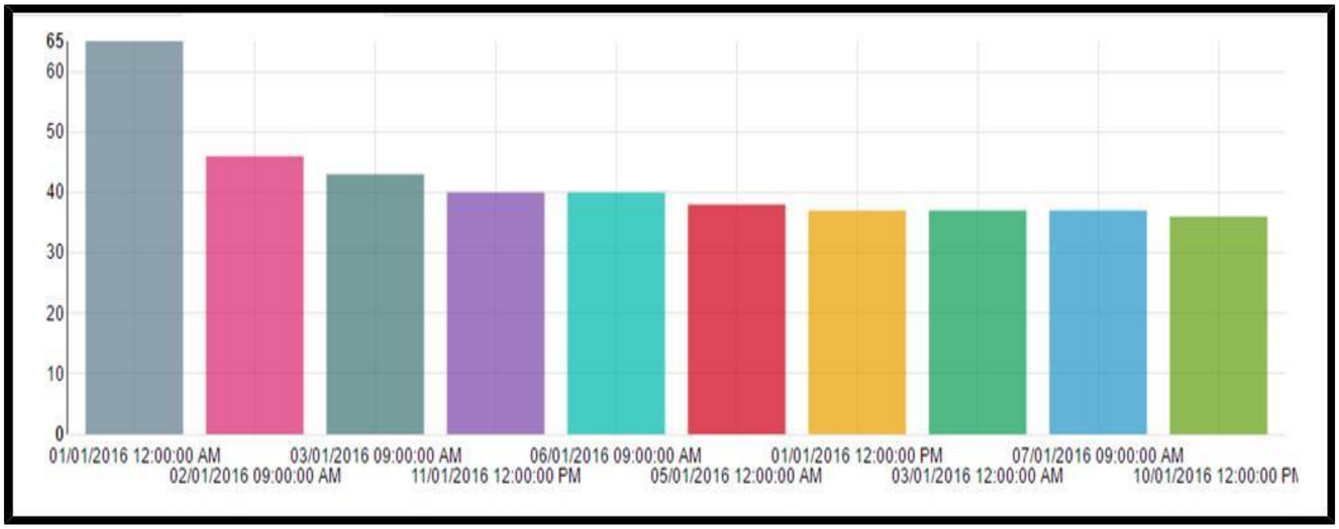


5. A look at the date and time when the highest number of incidents were reported.

SELECT date, COUNT(*) AS count FROM chicagoCrime2016 GROUP BY date ORDER BY count DESC



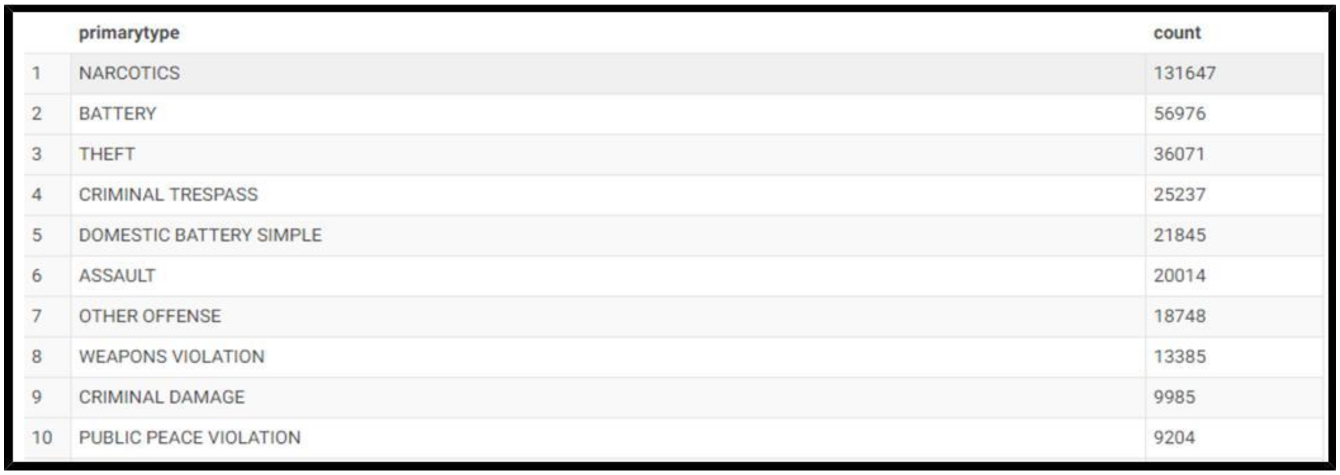
	casenumber	count
1	NULL	132
2	01/01/2016 12:01:00 AM	113
3	01/01/2016 12:00:00 AM	65
4	02/01/2016 09:00:00 AM	46
5	03/01/2016 09:00:00 AM	43
6	11/01/2016 12:00:00 PM	40
7	06/01/2016 09:00:00 AM	40
8	05/01/2016 12:00:00 AM	38
9	01/01/2016 12:00:00 PM	37
10	03/01/2016 12:00:00 AM	37



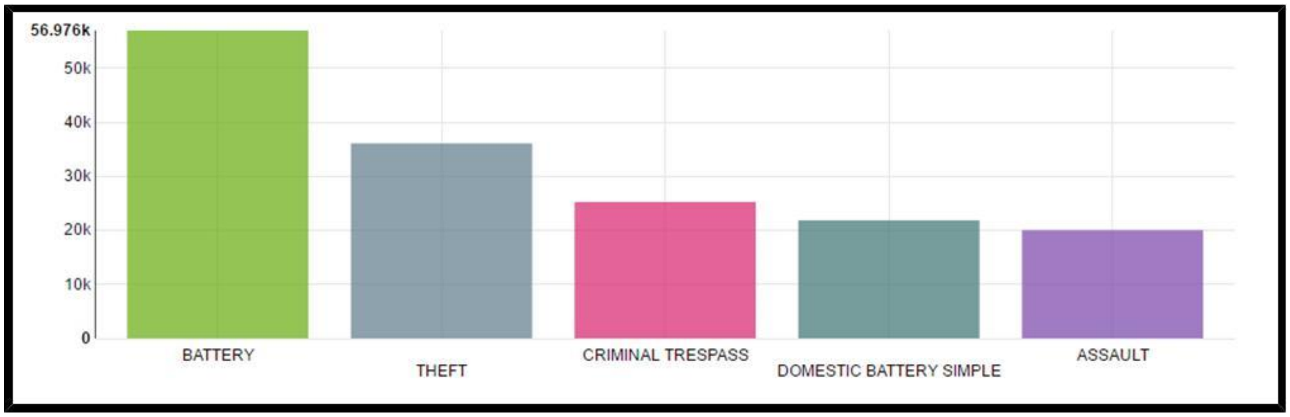
6. Arrests by primary type.

SELECT primaryType, COUNT(*) AS count FROM chicagoCrime2016 WHERE arrest = True

GROUP BY primaryType ORDER BY count DESC

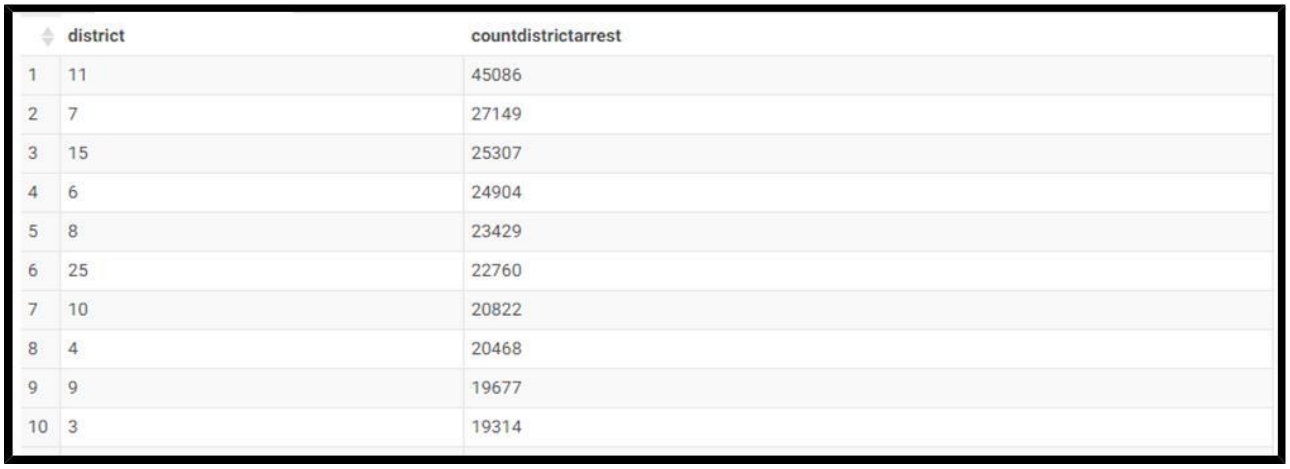


	primarytype	count
1	NARCOTICS	131647
2	BATTERY	56976
3	THEFT	36071
4	CRIMINAL TRESPASS	25237
5	DOMESTIC BATTERY SIMPLE	21845
6	ASSAULT	20014
7	OTHER OFFENSE	18748
8	WEAPONS VIOLATION	13385
9	CRIMINAL DAMAGE	9985
10	PUBLIC PEACE VIOLATION	9204

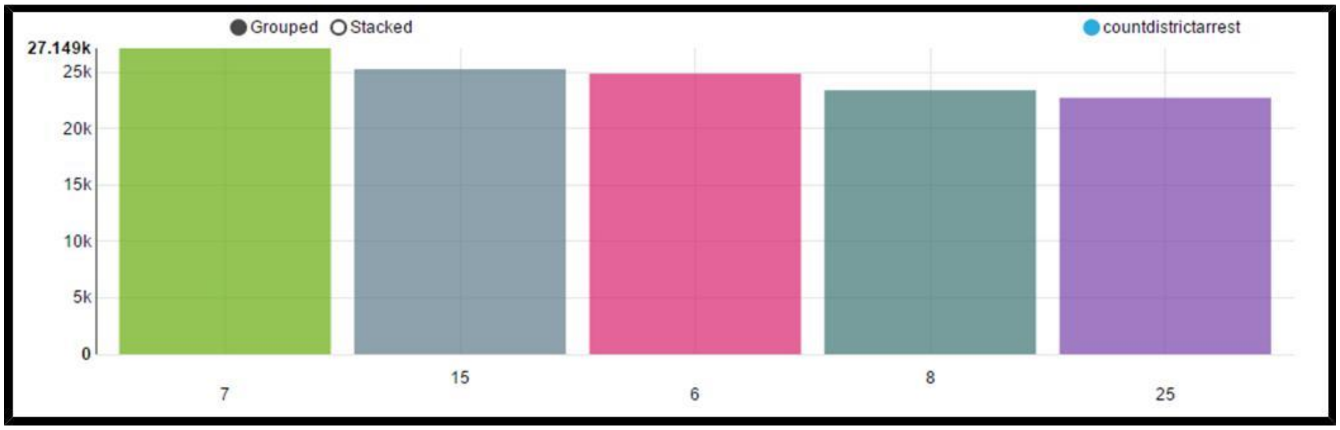


7. Arrests by district.

```
SELECT District, COUNT(*) AS countDistrictArrest FROM chicagoCrime2016 WHERE Arrest =

True GROUP BY District ORDER BY countDistrictArrest DESC
```



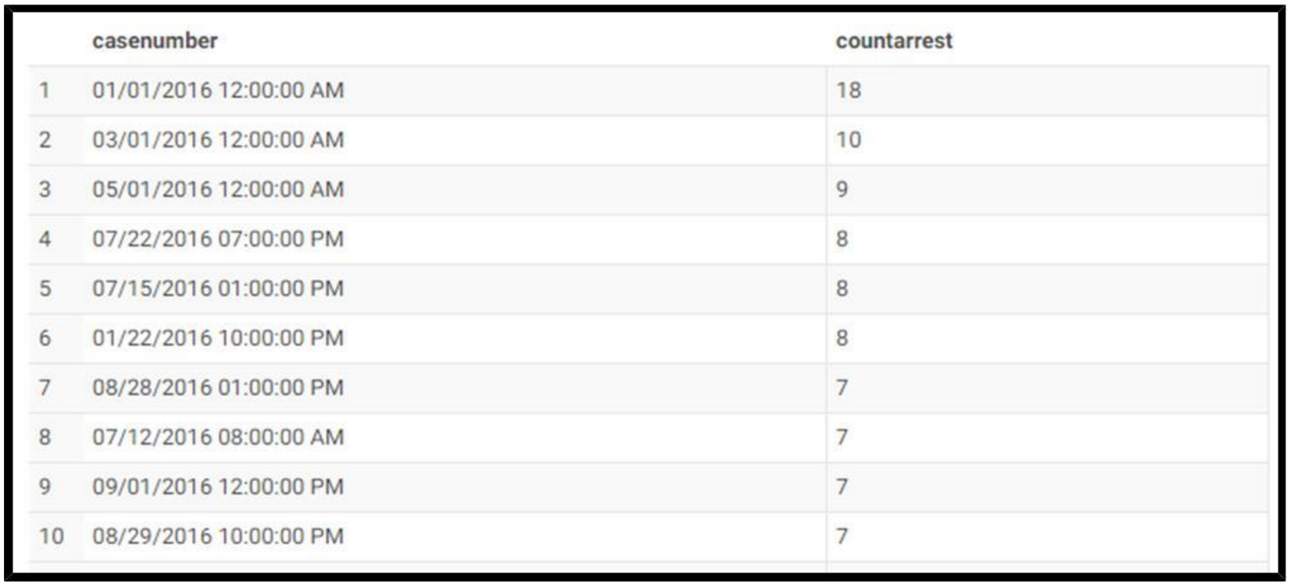
	district	countdistrictarrest
1	11	45086
2	7	27149
3	15	25307
4	6	24904
5	8	23429
6	25	22760
7	10	20822
8	4	20468
9	9	19677
10	3	19314



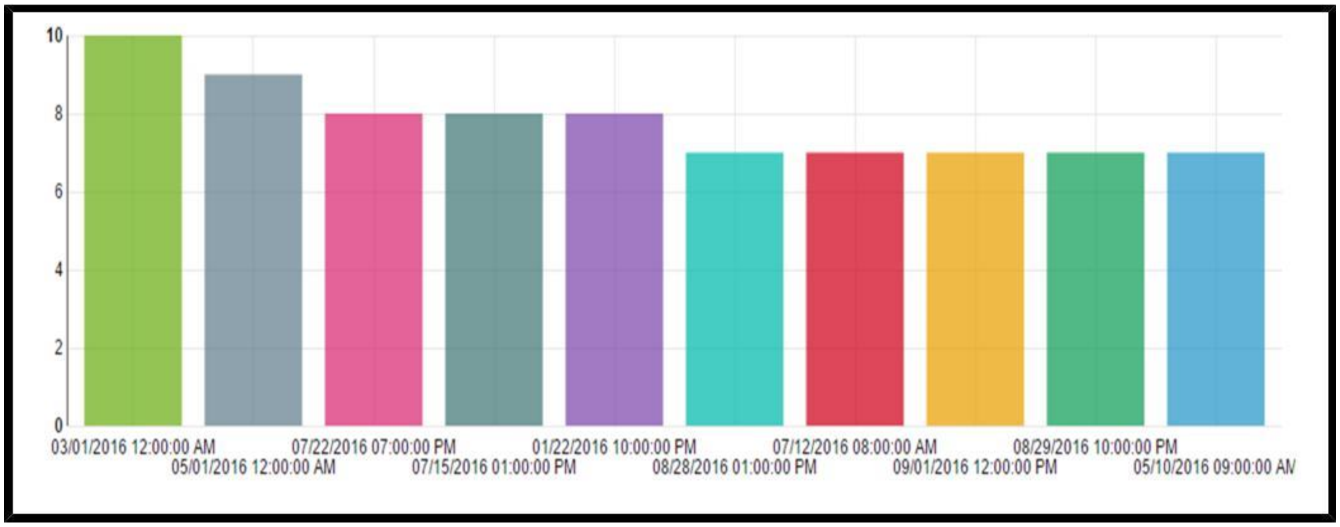
8. A look at the date and time when the highest number of arrests took place.

SELECT date, COUNT(*) AS countArrest FROM chicagoCrime2016 WHERE Arrest = True

GROUP BY date ORDER BY countArrest DESC

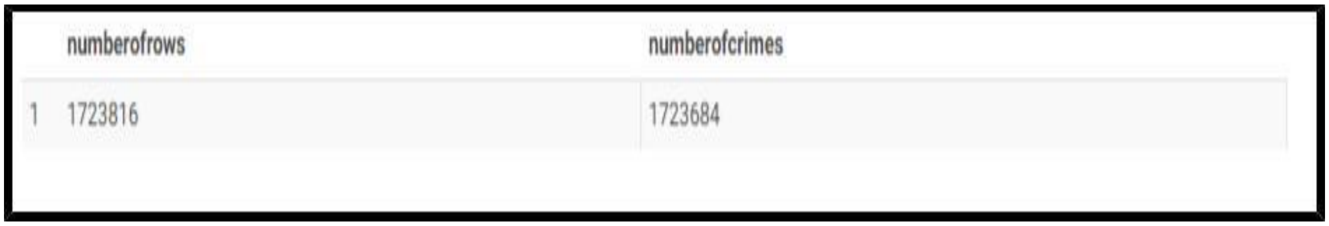


	casenumber	countarrest
1	01/01/2016 12:00:00 AM	18
2	03/01/2016 12:00:00 AM	10
3	05/01/2016 12:00:00 AM	9
4	07/22/2016 07:00:00 PM	8
5	07/15/2016 01:00:00 PM	8
6	01/22/2016 10:00:00 PM	8
7	08/28/2016 01:00:00 PM	7
8	07/12/2016 08:00:00 AM	7
9	09/01/2016 12:00:00 PM	7
10	08/29/2016 10:00:00 PM	7



9. Total number of crimes.

SELECT COUNT(*) as numberOfRows, count(casenumber) as numberOfCrimes From chicagoCrime2016;



	numberofrows	numberofcrimes
1	1723816	1723684

(9.2) PIG

Please refer to the 'HIVE' section for output results.

1. The most frequently occurring primary type.

1. crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description, locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode, xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
2. crimeGroupType = GROUP crime BY primaryType;

```
crimeGroupTypeCounted = FOREACH crimeGroupType GENERATE COUNT(crime) AS
cnt;
 sorted = ORDER crimeGroupTypeCounted BY cnt;
store sorted INTO 's3://bt-cs-383/output';
```

2. Districts with the most reported incidents.

```
crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
 crimeGroupDistrict = GROUP crime BY District;
crimeGroupDistrictCounted = FOREACH crimeGroupDistrict GENERATE
COUNT(crime) AS cnt;
 sorted = ORDER crimeGroupDistrictCounted BY cnt;
store sorted INTO 's3://bt-cs-383/output';
```

3. Blocks with the most reported incidents.

```
crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
crimeGroupBlock = GROUP crime BY Block;
crimeGroupBlockCounted = FOREACH crimeGroupBlock GENERATE
COUNT(crime) AS cnt;
sorted = ORDER crimeGroupBlockCounted BY cnt;
store sorted INTO 's3://bt-cs-383/output';
```

4. Blocks with the most reported incidents, grouped by primary type.

```
crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
crimeCoGroupBlockPrimaryType = COGROUP crime BY (block, primarytype);
crimeCoGroupBlockPrimaryTypeCounted = FOREACH
crimeCoGroupBlockPrimaryType GENERATE COUNT(crime) AS cnt;
sorted = ORDER crimeCoGroupBlockPrimaryTypeCounted BY cnt;
store sorted INTO 's3://bt-cs-383/output';
```

5. A look at the date and time when the highest number of incidents where reported.

```
crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
crimeGroupDate = GROUP crime BY Date;
crimeGroupDateCounted = FOREACH crimeGroupDate GENERATE
COUNT(crime) AS cnt;
sorted = ORDER crimeGroupDateCounted BY cnt;
store sorted INTO 's3://bt-cs-383/output';
```

6. Arrests by primary type.

```
crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
crimeFilter = FILTER crime BY ( UPPER (arrest) matches '.*TRUE.*' );
```

```
3.  crimeGroupType = GROUP crimeFilter BY primaryType;
4.  crimeGroupTypeCounted= FOREACH crimeGroupType GENERATE
COUNT(crimeFilter) AS cnt;
5.  sorted = ORDER crimeGroupTypeCounted BY cnt;
6.  store sorted INTO 's3://bt-cs-383/output';
```

7. Arrests by district.

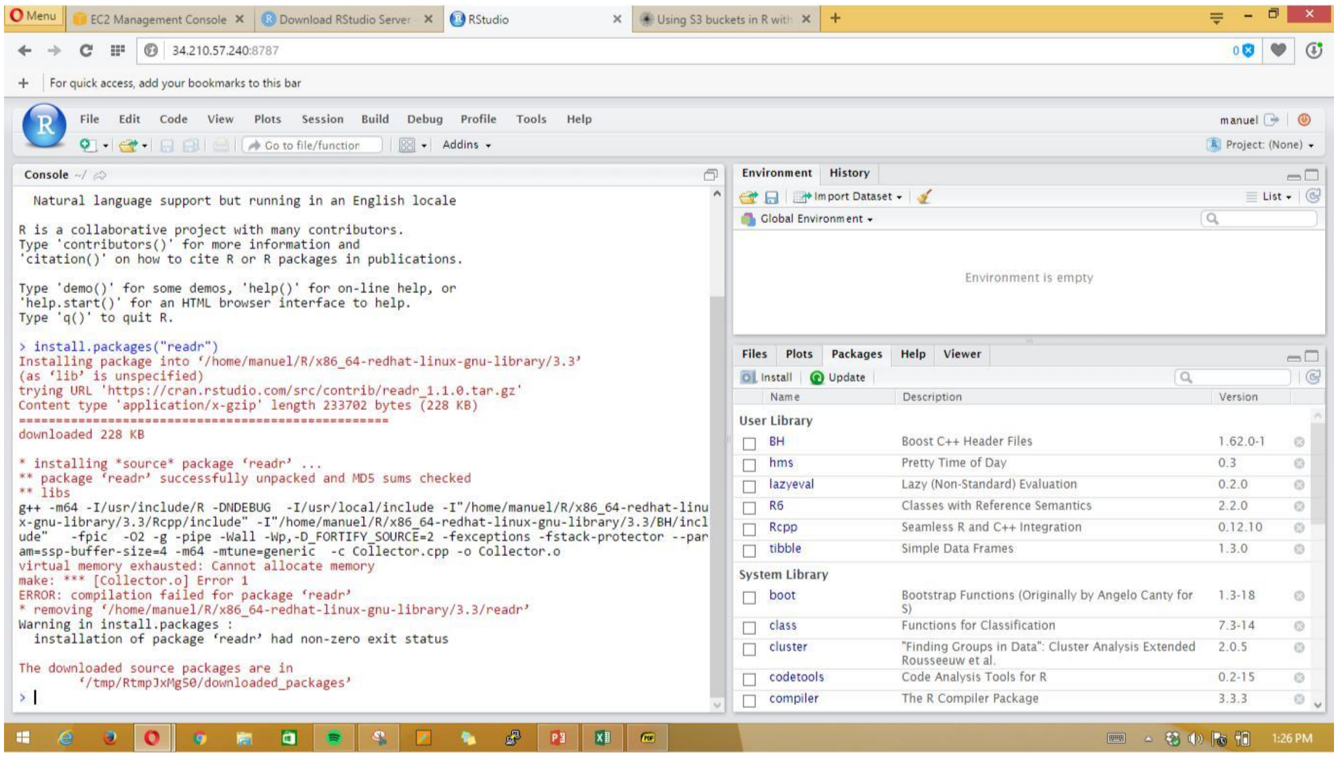
```
1.  crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType,
Description, locationDescription, Arrest, Domestic, Beat, District,
Ward, communityArea, fbiCode, xCoordinate, yCoordinate, Year, updatedOn, Latitude,
Longitude, Location); crime_filter = FILTER crime BY ( UPPER (arrest) matches '.*TRUE.*' );
2.  crimeGroupDistrict = GROUP crime BY district;
3.  crimeGroupDistrictCounted= FOREACH crimeGroupDistrict GENERATE
COUNT(crime) AS cnt;
4.  sorted = ORDER crimeGroupDistrictCounted BY cnt;
5.  store sorted INTO 's3://bt-cs-383/output';
```

8. A look at the date and time when the highest number of arrests took place.

```
1.  crime = LOAD 's3://bt-cs-383/chicago2016.csv' USING
PigStorage(',') AS (ID, caseNumber, Date, Block, IUCR, primaryType, Description,
locationDescription, Arrest, Domestic, Beat, District, Ward, communityArea, fbiCode,
xCoordinate, yCoordinate, Year, updatedOn, Latitude, Longitude, Location);
2.  crime_filter = FILTER crime BY ( UPPER (arrest) matches '.*TRUE.*' );
2.  crimeGroupDate = GROUP crime BY date;
3.  crimeGroupDateCounted = FOREACH crimeGroupDate GENERATE COUNT(crime) AS
cnt;
4.  sorted = ORDER crimeGroupDateCounted BY cnt;
```

5. store sorted INTO 's3://bt-cs-383/output';

(9.3) Running rStudio on Amazon AWS

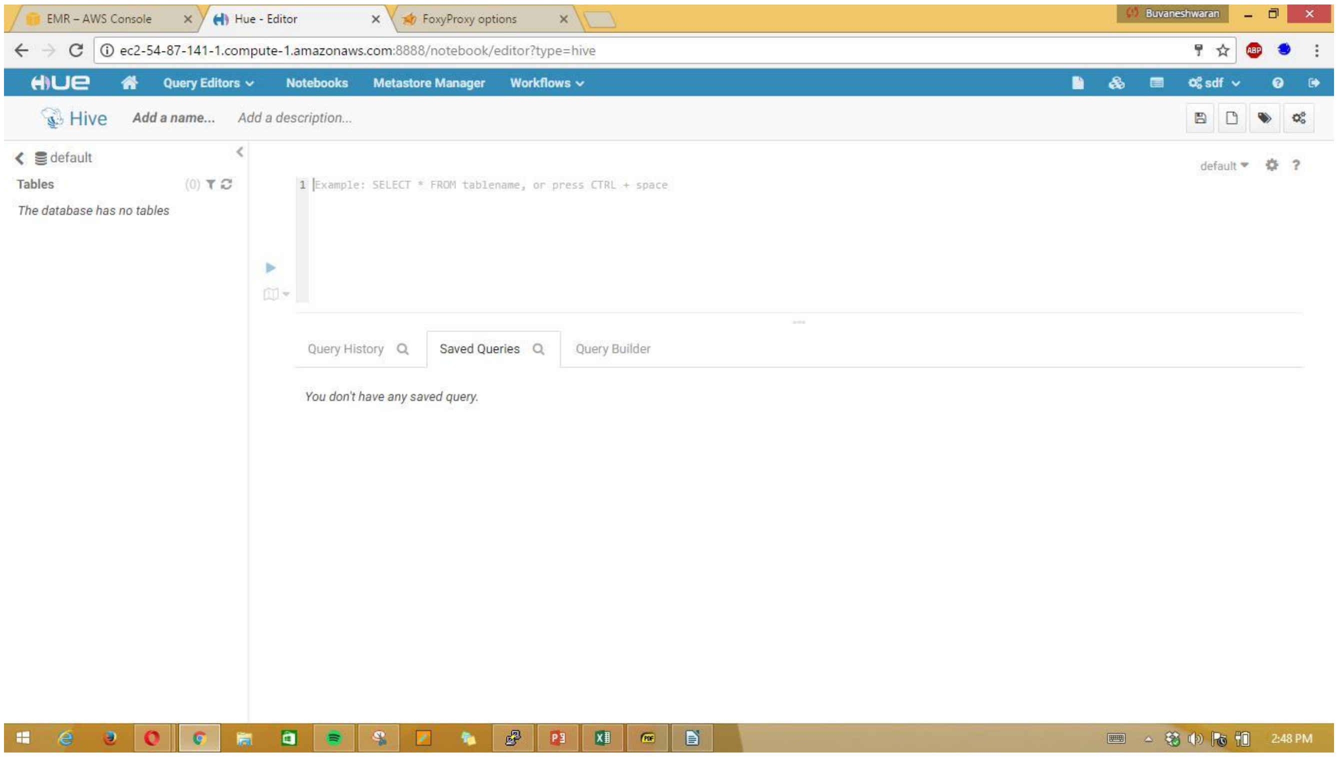


```
  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages("readr")
Installing package into ‘/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/src/contrib/readr_1.1.0.tar.gz'
Content type 'application/x-gzip' length 233702 bytes (228 KB)
==================================================
downloaded 228 KB

* installing *source* package ‘readr’ ...
** package ‘readr’ successfully unpacked and MD5 sums checked
** libs
g++ -m64 -I/usr/include/R -DNDEBUG  -I/usr/local/include -I"/home/manuel/R/x86_64-redhat-linu
x-gnu-library/3.3/Rcpp/include" -I"/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3/BH/incl
ude"   -fpic  -O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector --par
am=ssp-buffer-size=4 -m64 -mtune=generic  -c Collector.cpp -o Collector.o
virtual memory exhausted: Cannot allocate memory
make: *** [Collector.o] Error 1
ERROR: compilation failed for package ‘readr’
* removing ‘/home/manuel/R/x86_64-redhat-linux-gnu-library/3.3/readr’
Warning in install.packages :
  installation of package ‘readr’ had non-zero exit status

The downloaded source packages are in
	‘/tmp/RtmpJxMg50/downloaded_packages’
> 
```

Name	Description	Version
User Library		
BH	Boost C++ Header Files	1.62.0-1
hms	Pretty Time of Day	0.3
lazyeval	Lazy (Non-Standard) Evaluation	0.2.0
R6	Classes with Reference Semantics	2.2.0
Rcpp	Seamless R and C++ Integration	0.12.10
tibble	Simple Data Frames	1.3.0
System Library		
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-18
class	Functions for Classification	7.3-14
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.0.5
codetools	Code Analysis Tools for R	0.2-15
compiler	The R Compiler Package	3.3.3

(9.4) Running Hue on AWS

HIVE

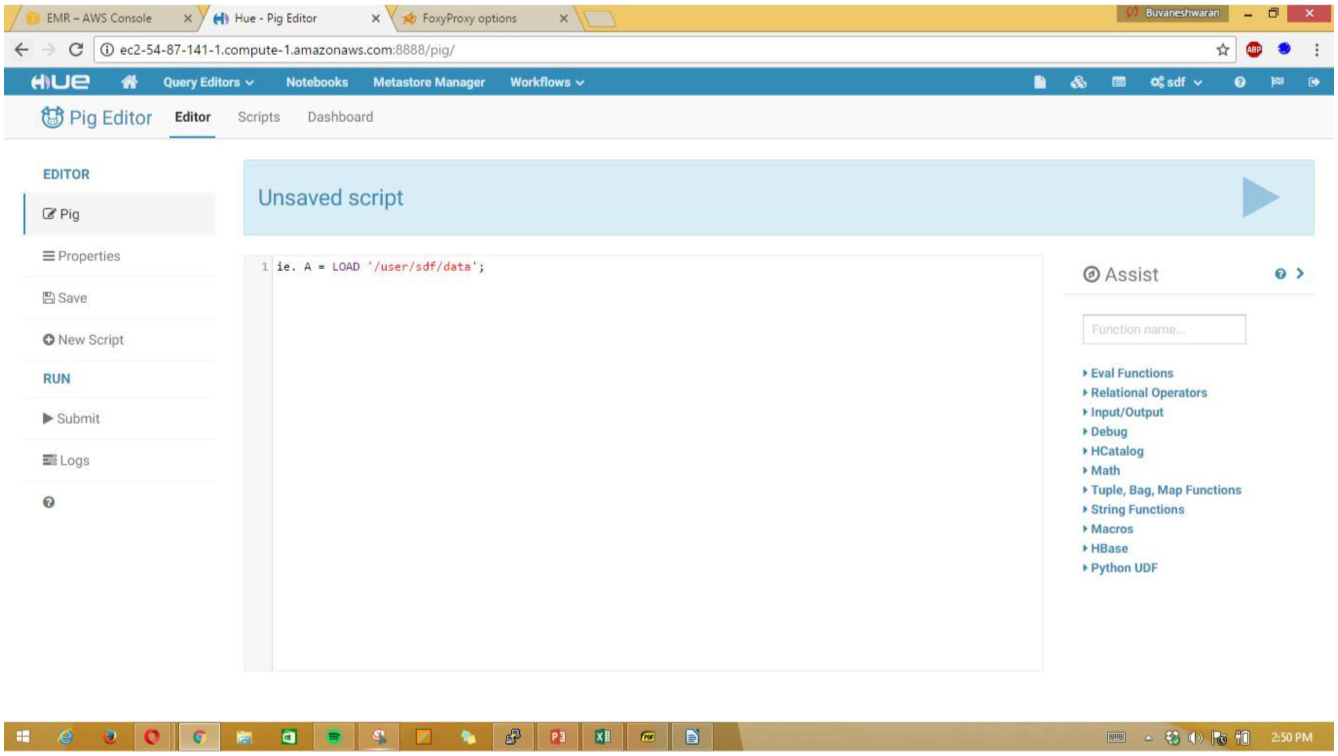


The database has no tables

```
Example: SELECT * FROM tablename, or press CTRL + space
```

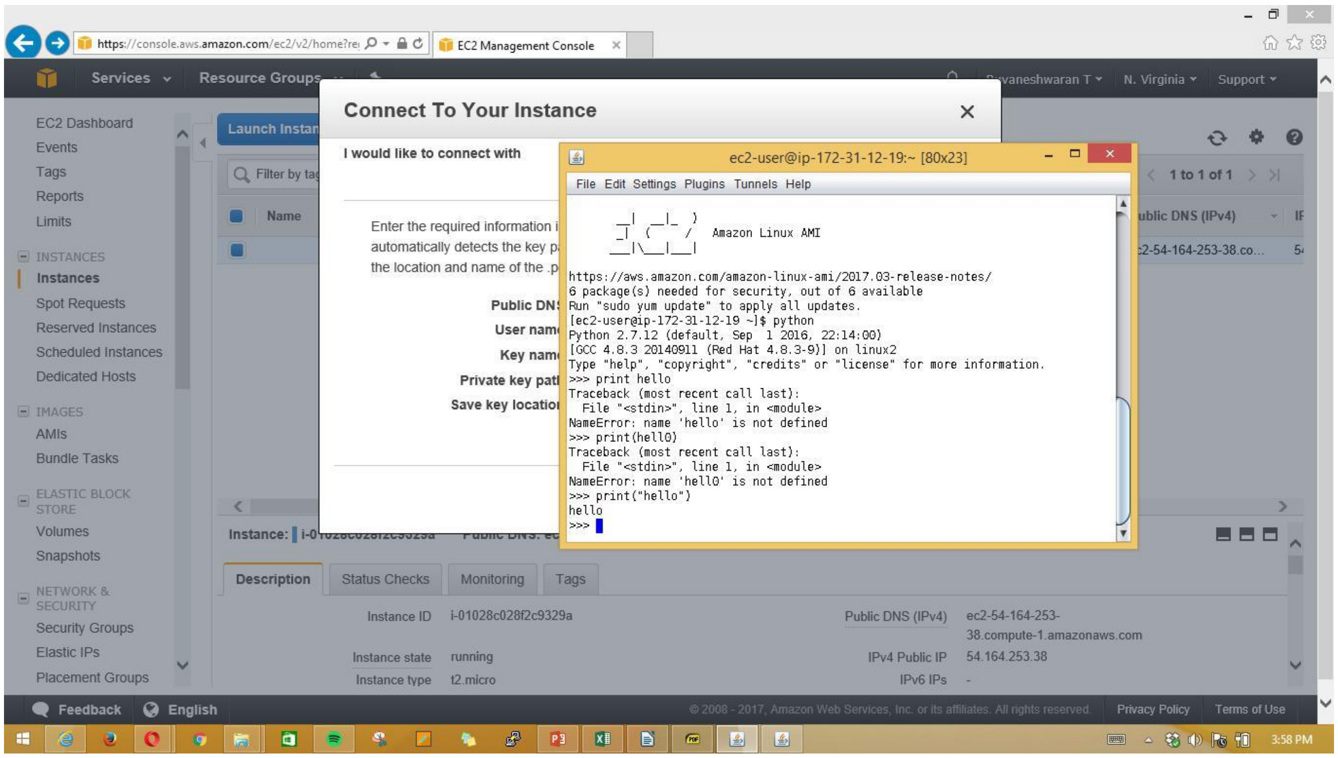
You don't have any saved query.

PIG



```
ie. A = LOAD '/user/sdf/data';
```

(9.5) Running Python on AWS



EC2 Management Console
Connect To Your Instance
ec2-user@ip-172-31-12-19:~ [80x23]
File Edit Settings Plugins Tunnels Help

```
       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
6 package(s) needed for security, out of 6 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-12-19 ~]$ python
Python 2.7.12 (default, Sep  1 2016, 22:14:00)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hello' is not defined
>>> print(hell0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hell0' is not defined
>>> print("hello")
hello
>>> 
```