Project Report on

Big Data Technologies

Chicago Crime Analysis

Submitted for the award of the Master of Data Science Degree

By

Sujay Puvvadi

A20449705

Under the guidance of

Prof. Joseph Rosen

Illinois Institute of Technology

Master of Data Science

Illinois Institute of Technology

Chicago

December 2019

Contents

Project Proposal

Project Title:

Predictive analytics on crime in Chicago

Application Subject Area:

Crime

Dataset Source:

Chicago crimes in 2019. This dataset has got over 211,000 crime reports, with attributes such as case number, location, date, crime type, etc.

Project Motive:

To predict the future crimes in a particular area, with attributes such as date and location. Also to understand which days of the week the crime is highest.

Tools:

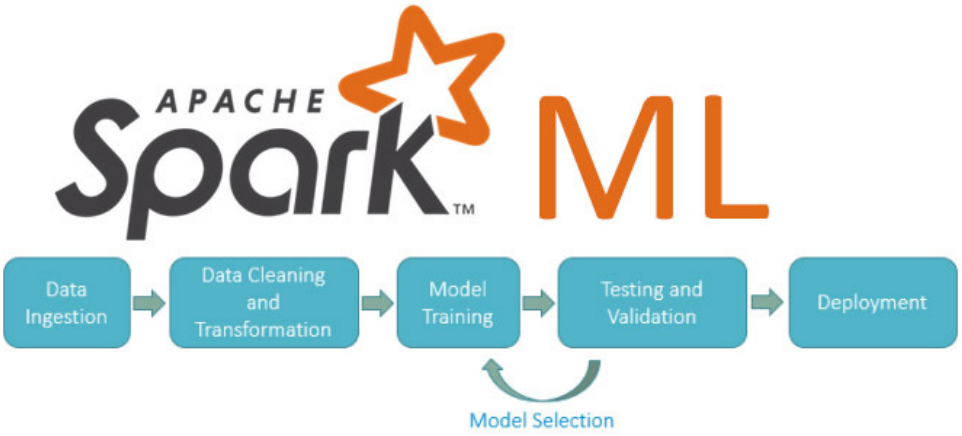
- Apache Spark (PySpark)
- Apache SparkSQL
- Amazon EMR
- Python

Introduction

This project is designed to predict the future crimes in a particular area, with attributes such as date and location. This project also to understand which days of the week the crime is highest. I will be using the dataset "Chicago crimes in 2019" from cityofchicago's website. This dataset has got over 211,000 crime reports which keep getting updated everyday, with attributes such as case number, location, date, crime type, etc.

I will be using S3 buckets to store the dataset, Apache Spark, Apache SparkSQL, Apache Spark Pyspark for data sourcing, data processing, data visualization and for the final results.

Search of Literature



MLlib

Apache Spark's Machine Learning Library (MLlib) is designed for simplicity, scalability, and easy integration with other tools. With the scalability, language compatibility, and speed of Spark, data scientists can focus on their data problems and models instead of solving the complexities surrounding distributed data (such as infrastructure, configurations, and so on).

MLlib allows for pre-processing, munging, training of models, and making predictions at scale on data. You can even use models trained in MLlib to make predictions in Structured Streaming. Spark provides a sophisticated machine learning API for performing a variety of machine learning tasks, from classification to regression, clustering to deep learning.

MLlib vs H2O

The fundamental difference between H2O and Spark MLLib is that H2O operates on a special data format (.hex), while MLLib takes in an RDD or a Dataset which are the basic data structures of Spark.In terms of algorithms available I'd say they are on par, as each of them has an algorithm for each task, albeit their choices of algorithms and their implementations vary.H2O is good for random forests and GBMs, but MLlib is better in every other way. In terms of sheer performance, MLlib is faster than H2O.

H2O is also solely concentrating on ML algorithms, while Spark is a whole framework.

MLlib vs Mahout

Since Mahout is built on top of MapReduce, it is constrained by disl accesses. It is slow and does not handle iterative jobs very well. Machine learning algorithms often need many iterations, therefore makes Mahout run really slow.
On the other hand, MLlib is built on top of Apache Spark, which is much faster than Mahout. Since MLlib is built on top of Apache Spark, it gets to take advantage of Spark's efficiency when running iterative Machine Learning algorithms. Its algorithms end up being much faster than Mahout equivalents.

MLlib vs ML

The Spark MlLib comes in 2 packages: Mllib and ML. While MlLib contains the original API built on top of RDDs, ML provides higher-level API built on top of DataFramesfor constructing ML pipelines which is much faster than MlLib.

Spark ML library, which is different from MLLib: Spark ML makes use of the DataFrame format which is fairly new in Spark, however it does not allow all types of algorithms, therefore although Spark ML may be faster than MLLib, you might find yourself needing MLLib anyways.

Implementation

Data Sourcing

I will be uploading the existing data to the Amazon Cloud. Amazon S3 buckets are going to be used. There are many ways to access the data from buckets. I will be accessing the data either through EMR or directly through PySpark. Here is a little bit about the S3 cloud storage:

Amazon S3:

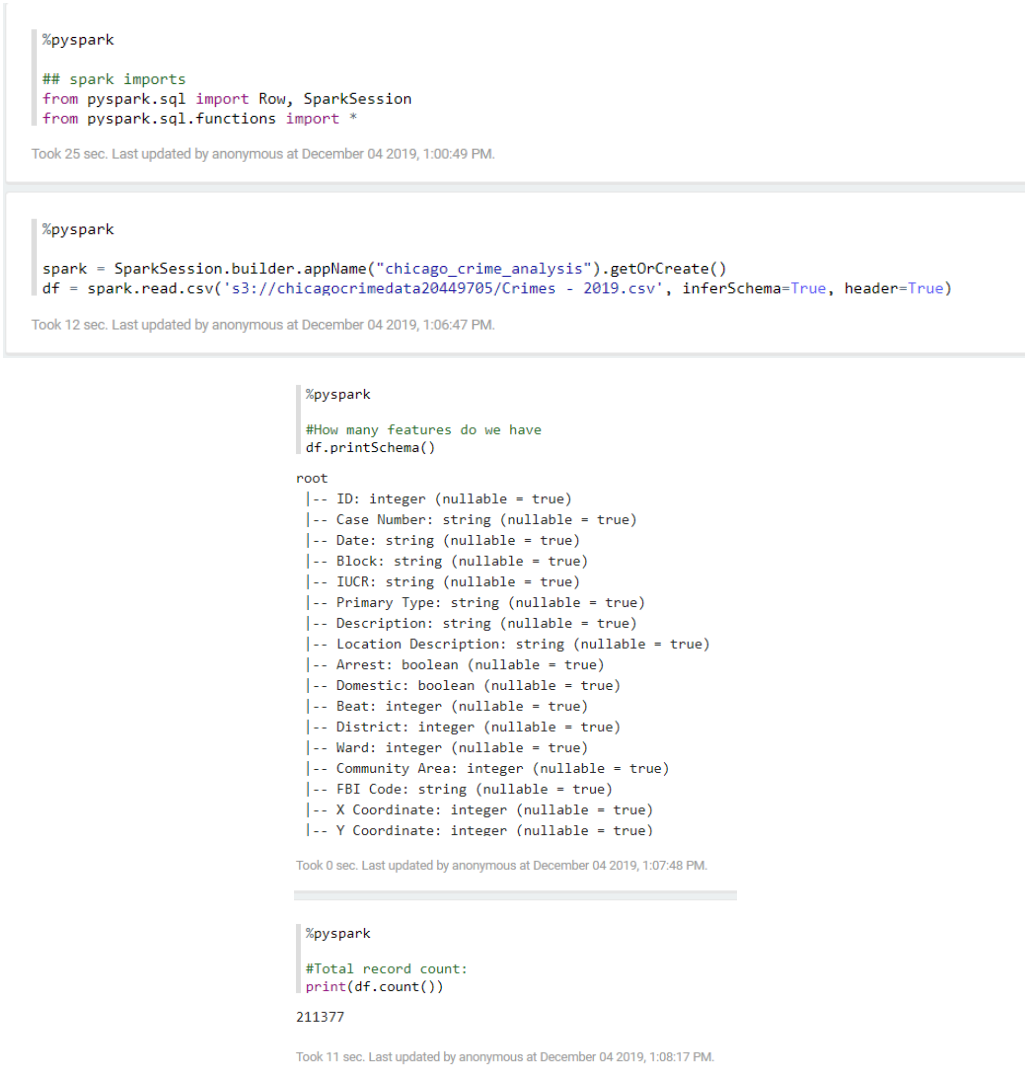
Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Amazon S3 is a bit bucket that acts as the storage end of a virtual Infrastructure-as-a-Service (IaaS) solution housed in the Amazon Web Services (AWS) cloud. Amazon S3 begins at $0.023 per gigabyte (GB) per month for your first 50 terabytes (TB) of data. However, this price scales downward the more total data you store in Amazon S3, in an effort to ensure that you pay only for exactly what you use. Amazon S3 is largely managed from its web console.

Amazon has established a sophisticated provider-managed key service known as the Key Management Service (KMS) to both generate and protect encryption keys for data stored in Amazon S3. This one-time operation ensures that nobody is getting access to your data

without the original key generated by the KMS. Amazon has implemented the strongest, industry-standard encryption methods.

Why Amazon S3 over any other cloud storage providers?

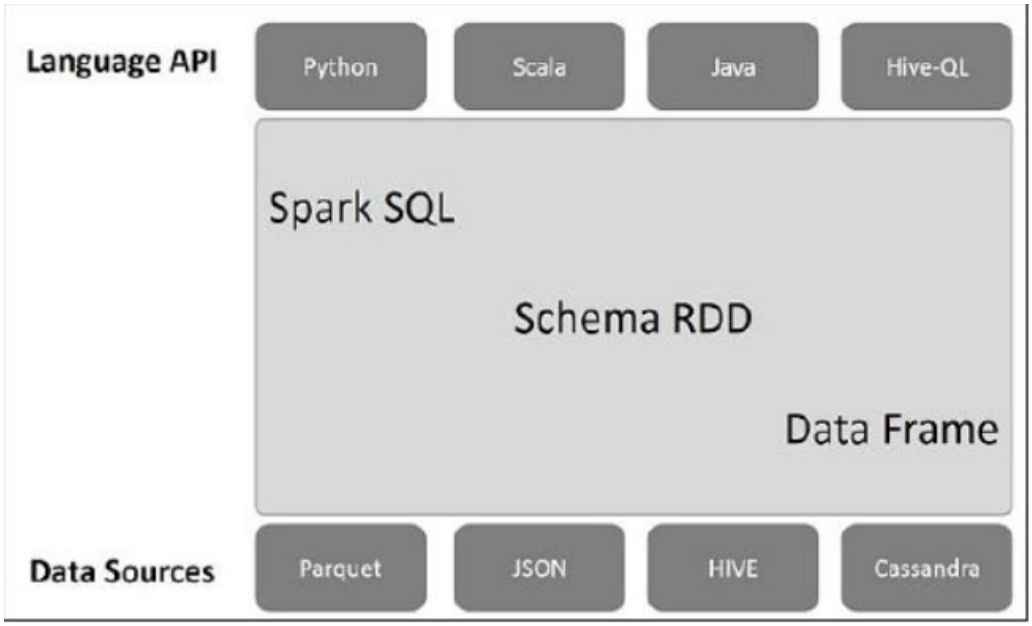
- **Amazon AWS S3 over Microsoft Azure Blob** as S3 is much cheaper than Blob, even though both storage services are equally robust in terms of every other thing. Also, Apache spark works more efficiently with Amazon S3.
- **Amazon AWS S3 over Google Cloud Service** as GCS allows a maximum of 1024 parts, while S3 allows 10,000 and both share the same 5TB maximum file size, for multipart uploads. Also, both GCS and S3 provide geo redundant storage but AWS implementation supports more locations, flexibility and API control. When looking at object storage compatible applications, S3 is clearly the most widely supported API by far.
- **Amazon AWS S3 over IBM Cloud IaaS** as S3 has a much simpler UI to deal with. Furthermore, Amazon S3 has more flexibility and it is more cost-effective.



As you can see from the above pictures, Amazon S3 Buckets have been used as the primary storage.

Data Transformation and Preprocessing

Apache Spark SQL:



Spark SQL Architecture

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data. It provides a programming abstraction called DataFrame and can act as distributed SQL query engine.

Features of Spark SQL:

- Integrated − Seamlessly mix SQL queries with Spark programs. Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Python, Scala and Java. This tight integration makes it easy to run SQL queries alongside complex analytic algorithms.
- Unified Data Access − Load and query data from a variety of sources. Schema-RDDs provide a single interface for efficiently working with structured data, including Apache Hive tables, parquet files and JSON files.
- Hive Compatibility − Run unmodified Hive queries on existing warehouses. Spark SQL reuses the Hive frontend and MetaStore, giving you full compatibility with existing Hive data, queries, and UDFs. Simply install it alongside Hive.
- Standard Connectivity − Connect through JDBC or ODBC. Spark SQL includes a server mode with industry standard JDBC and ODBC connectivity.
- Scalability − Use the same engine for both interactive and long queries. Spark SQL takes advantage of the RDD model to support mid-query fault tolerance, letting it scale to large jobs too. Do not worry about using a different engine for historical data.

Dataframes:

A DataFrame is a distributed collection of data, which is organized into named columns. Conceptually, it is equivalent to relational tables with good optimization techniques. A DataFrame can be constructed from an array of different sources such as Hive tables, Structured Data files, external databases, or existing RDDs. This API was designed for modern Big Data and data science applications taking inspiration from DataFrame in R Programming and Pandas in Python.

Features of DataFrame:

- Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.
- Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc).
- State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).
- Can be easily integrated with all Big Data tools and frameworks via Spark-Core.
- Provides API for Python, Java, Scala, and R Programming.

```
%pyspark

#Crime Types
crime type groups = df.groupBy('Primary Type').count()
```

Took 0 sec. Last updated by anonymous at December 04 2019, 1:08:58 PM.

Here, we are showing the crimes by their overall frequency.

```
%pyspark

crime_type_counts = crime_type_groups.orderBy('count', ascending=False)
```

ook 0 sec. Last updated by anonymous at December 04 2019, 1:09:50 PM.

```
%pyspark

#These are the top 20 most frequent crime types:
crime_type_counts.show()
```

```
+--------------------+-----+
|        Primary Type|count|
+--------------------+-----+
|               THEFT|50332|
|             BATTERY|40949|
|     CRIMINAL DAMAGE|22105|
|             ASSAULT|17143|
|  DECEPTIVE PRACTICE|14041|
|       OTHER OFFENSE|13795|
|           NARCOTICS|11481|
|            BURGLARY| 7780|
| MOTOR VEHICLE THEFT| 7297|
|             ROBBERY| 6400|
|   CRIMINAL TRESPASS| 5614|
|   WEAPONS VIOLATION| 5252|
|OFFENSE INVOLVING...| 1911|
| CRIM SEXUAL ASSAULT| 1318|
|PUBLIC PEACE VIOL...| 1307|
```

Took 2 sec. Last updated by anonymous at December 04 2019, 1:10:41 PM.

Here, we are processing the “Date” column and checking when the crime began in the current year.
So it seems that the dataset we're dealing with comprises records from 2019-01-01 to 2019-10-23

```
%pyspark

#Recorded Date

import datetime
from pyspark.sql.functions import *
```

Took 0 sec. Last updated by anonymous at December 04 2019, 1:12:24 PM. (outdated)

```
%pyspark

df.select(min('Date').alias('first_record_date'), max('Date').alias('latest_record_date')).show()
```

```
+--------------------+--------------------+
|   first_record_date|  latest_record_date|
+--------------------+--------------------+
|01/01/2019 01:00:...|10/23/2019 12:58:...|
+--------------------+--------------------+
```

Took 0 sec. Last updated by anonymous at December 04 2019, 1:13:17 PM.

```
%pyspark

df = df.withColumn('date_time', to_timestamp('date', 'MM/dd/yyyy hh:mm:ss a'))\
       .withColumn('month', trunc('date_time', 'YYYY'))
```

Took 0 sec. Last updated by anonymous at December 04 2019, 1:16:46 PM.

```
%pyspark

df.select(['date','date_time', 'month']).show(n=2, truncate=False)
```

```
+----------------------+-------------------+----------+
|date                  |date_time          |month     |
+----------------------+-------------------+----------+
|10/22/2019 12:00:00 PM|2019-10-22 12:00:00|2019-01-01|
|10/15/2019 10:00:00 AM|2019-10-15 10:00:00|2019-01-01|
+----------------------+-------------------+----------+
only showing top 2 rows
```

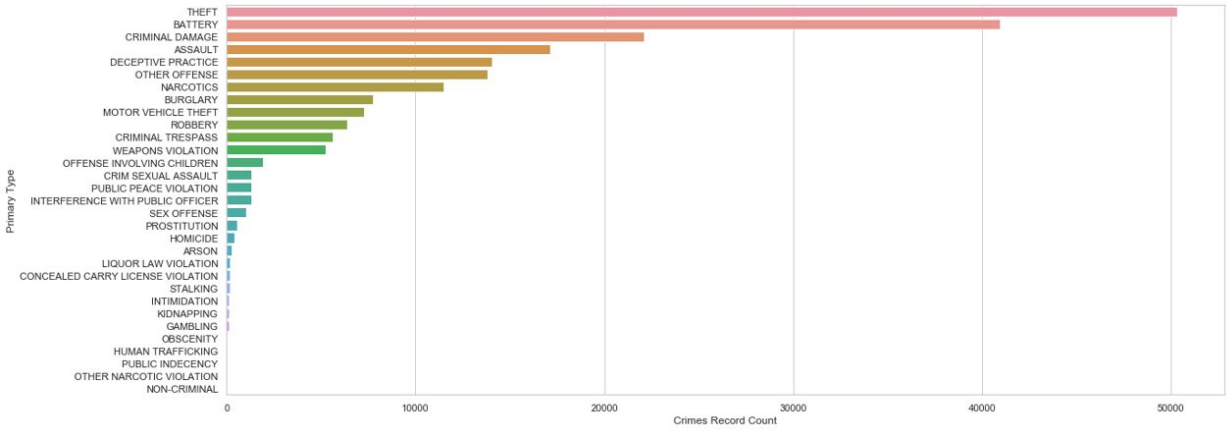
Took 0 sec. Last updated by anonymous at December 04 2019, 1:16:59 PM.

```
df.groupBy(['Location Description']).count().orderBy('count', ascending=False).show(10)
```

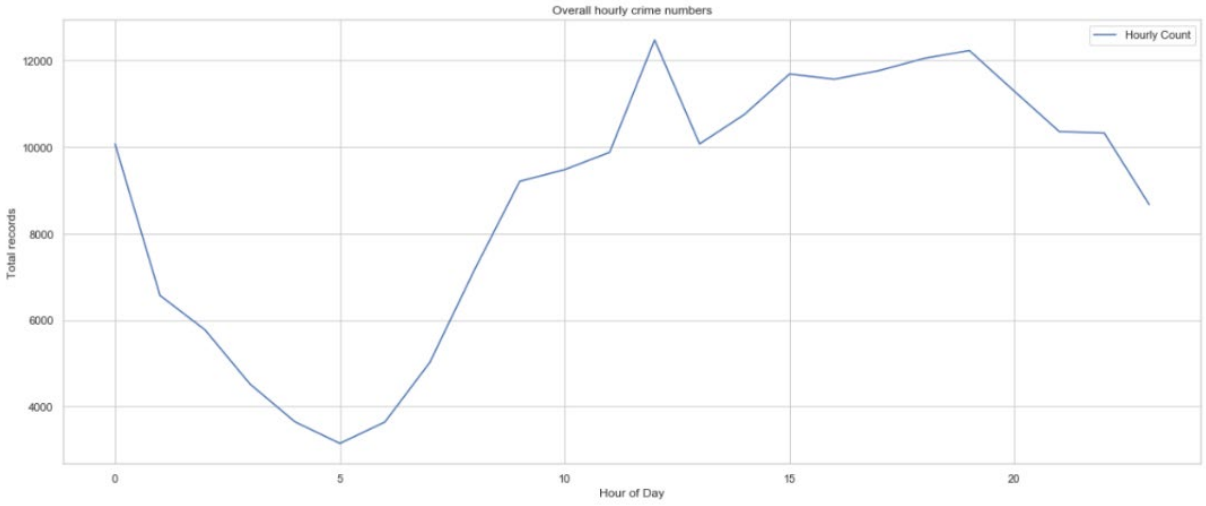
```
+--------------------+-----+
|Location Description|count|
+--------------------+-----+
|              STREET|46805|
|           RESIDENCE|34434|
|           APARTMENT|27890|
|            SIDEWALK|16942|
|               OTHER| 8577|
|PARKING LOT/GARAG...| 6218|
|          RESTAURANT| 5677|
|  SMALL RETAIL STORE| 5595|
|              ALLEY| 4257|
|RESIDENTIAL YARD ...| 4030|
+--------------------+-----+
only showing top 10 rows
```

Analysis

- The graph below shows the count of each crime that happened inn Chicago since the beginning of 2019.



- From the graph below, we can interpret that the greatest number of crimes occur between the times 10AM to 8PM.



- This table shows how many crimes occurred in each type of area.

```
df.groupBy(['Location Description']).count().orderBy('count', ascending=False).show(10)
```

```
+--------------------+-----+
|Location Description|count|
+--------------------+-----+
|              STREET|46805|
|           RESIDENCE|34434|
|           APARTMENT|27890|
|            SIDEWALK|16942|
|               OTHER| 8577|
|PARKING LOT/GARAG...| 6218|
|          RESTAURANT| 5677|
|  SMALL RETAIL STORE| 5595|
|              ALLEY| 4257|
|RESIDENTIAL YARD ...| 4030|
+--------------------+-----+
only showing top 10 rows
```

A closer look at crime date and time

- The information given here indicates when the crime is perpetrated. The date/time field may be able to draw a meaningful trend that can be used to predict crime. However, I believe that this leads much more to external factors, such as policy changes, law enforcement-related factors, and so on.
- It's much more likely that time-related features that are more closely relatable to crime occurrence be better predictors than the date and time. I mean, knowing the month of the year, the day of the week, and the hour of the day that the crime occurred can enable better chances of predicting accurately than simply knowing "when" AD crimes occurred.
- Adding predictors (so far fields) that read time information.
- Hour of the day (already added the 'hour' field)
- Day of the week
- Month of the year
- Day in a range. Instead of using the entire date-time, we'll use a "day sequence" that is counted from 2019-01-01.

```
%pyspark


df_dates = df_hour.withColumn('week_day', dayofweek(df_hour['date_time']))\
                  .withColumn('year_month', month(df_hour['date_time']))\
                  .withColumn('month_day', dayofmonth(df_hour['date_time']))\
                  .withColumn('date_number', datediff(df['date_time'], to_date(lit('2001-01-01'), format='yyyy-MM-dd')))\
                  .cache()
```

Took 0 sec. Last updated by anonymous at December 04 2019, 1:35:14 PM.

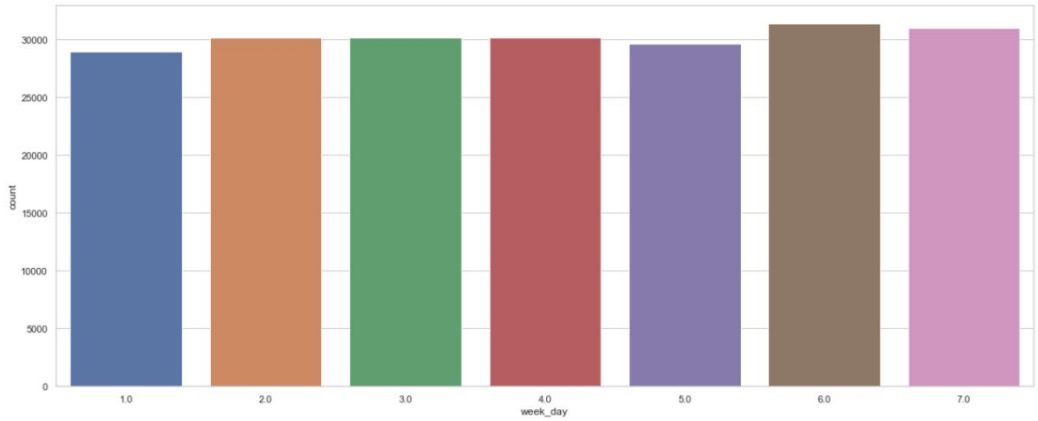
```
%pyspark

df_dates.select(['date', 'month', 'hour', 'week_day', 'year', 'year_month', 'month_day', 'date_number']).show(20, truncate=False)
```

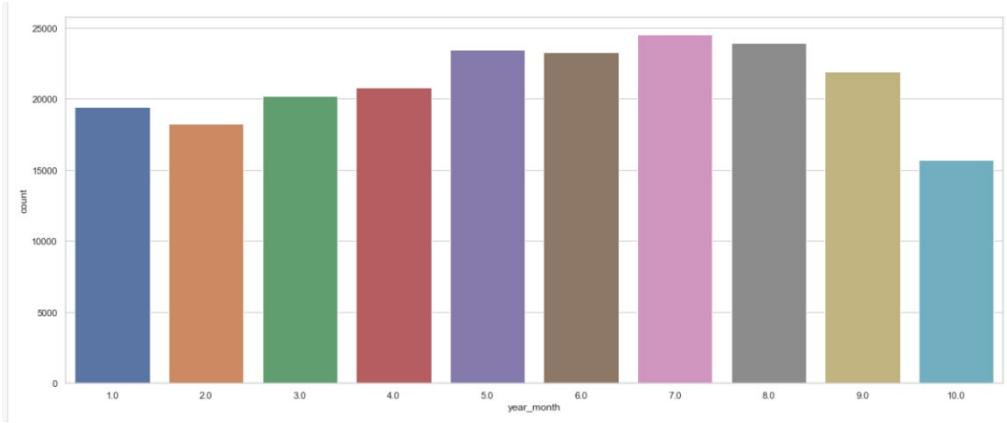
```
+----------------------+----------+----+--------+----+----------+---------+-----------+
|date                  |month     |hour|week_day|year|year_month|month_day|date_number|
+----------------------+----------+----+--------+----+----------+---------+-----------+
|10/22/2019 12:00:00 PM|2019-01-01|12  |3       |2019|10        |22       |6868       |
|10/15/2019 10:00:00 AM|2019-01-01|10  |3       |2019|10        |15       |6861       |
|10/23/2019 09:54:00 AM|2019-01-01|9   |4       |2019|10        |23       |6869       |
|10/20/2019 12:01:00 AM|2019-01-01|0   |1       |2019|10        |20       |6866       |
|10/23/2019 03:40:00 PM|2019-01-01|15  |4       |2019|10        |23       |6869       |
|10/23/2019 08:30:00 PM|2019-01-01|20  |4       |2019|10        |23       |6869       |
|10/23/2019 12:01:00 PM|2019-01-01|12  |4       |2019|10        |23       |6869       |
|10/23/2019 10:00:00 PM|2019-01-01|22  |4       |2019|10        |23       |6869       |
|10/23/2019 12:00:00 PM|2019-01-01|12  |4       |2019|10        |23       |6869       |
|10/23/2019 03:10:00 PM|2019-01-01|15  |4       |2019|10        |23       |6869       |
|10/23/2019 05:45:00 PM|2019-01-01|17  |4       |2019|10        |23       |6869       |
|10/21/2019 10:03:00 AM|2019-01-01|10  |2       |2019|10        |21       |6867       |
|10/23/2019 06:25:00 PM|2019-01-01|18  |4       |2019|10        |23       |6869       |
|10/23/2019 12:00:00 PM|2019-01-01|12  |4       |2019|10        |23       |6869       |
|05/31/2019 02:44:00 PM|2019-01-01|14  |6       |2019|5         |31       |6724       |
```

Took 5 sec. Last updated by anonymous at December 04 2019, 1:35:29 PM.

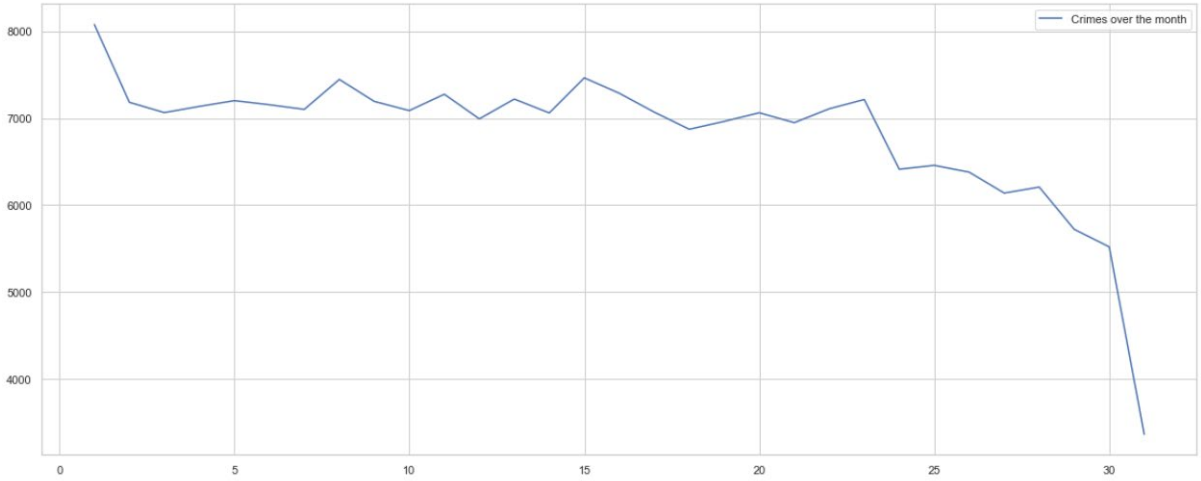
- The graph below shows the crimes per day, that is, it shows which days of the week is/are the most dangerous, starting from Monday.



- This graph shows the crimes occurred in each month for this year until date. From the graph below, we can infer that, July has not been a very good time for Chicago.



- This graph shows the number of crimes occurring per day, in a month. It seems like the crimes happen the most in the first week of every month.



```
 Community areas?
Chicago has 77 community areas. How do they appear next to one another in a count plot?
```

```
df_dates_community_areas = df_dates.na.drop(subset=['Community Area']).groupBy('Community Area').count()
```

What are the to 10 areas with recorded crime?

```
df_dates_community_areas.orderBy('count', ascending=False).show(10)
```

```
+--------------+-----+
|Community Area|count|
+--------------+-----+
|            25|12011|
|             8|10127|
|            32| 8662|
|            28| 7471|
|            29| 7212|
|            43| 6983|
|            23| 6504|
|            71| 6202|
|            24| 5939|
|            44| 5520|
+--------------+-----+
only showing top 10 rows
```

- Let's link names of these community areas. These float numbers are having a hard time making any sense.

```
area_names = """
01→Rogers Park
40→Washington Park
02→West Ridge
41→Hyde Park
03→Uptown
42→Woodlawn
04→Lincoln Square
43→South Shore
05→North Center
44→Chatham
06→Lakeview
45→Avalon Park
07→Lincoln Park
46→South Chicago
08→Near North Side
47→Burnside
09→Edison Park
48→Calumet Heights
10→Norwood Park
49→Roseland
11→Jefferson Park
50→Pullman
12→Forest Glen
51→South Deering
13→North Park
52→East Side
14→Albany Park
```

```
53→West Pullman
15→Portage Park
54→Riverdale
16→Irving Park
55→Hegewisch
17→Dunning
56→Garfield Ridge
18→Montclare
57→Archer Heights
19→Belmont Cragin
58→Brighton Park
20→Hermosa
59→McKinley Park
21→Avondale
60→Bridgeport
22→Logan Square
61→New City
23→Humboldt Park
62→West Elsdon
24→West Town
63→Gage Park
25→Austin
64→Clearing
26→West Garfield Park
65→West Lawn
27→East Garfield Park
66→Chicago Lawn
28→Near West Side
67→West Englewood
29→North Lawndale
68→Englewood
30→South Lawndale
69→Greater Grand Crossing
31→Lower West Side
70→Ashburn
32→Loop
```

```
71→Auburn Gresham
33→Near South Side
72→Beverly
34→Armour Square
73→Washington Heights
35→Douglas
74→Mount Greenwood
36→Oakland
75→Morgan Park
37→Fuller Park
76→O'Hare
38→Grand Boulevard
77→Edgewater
39→Kenwood
"""
```

References

- Dataset: https://data.cityofchicago.org/Public-Safety/Crimes-2019/w98m-zvie
- https://www.trustradius.com/compare-products/amazon-s3-simple-storage-service-vs-ibm-cloud-iaas
- https://www.zenko.io/blog/four-differences-google-amazon-s3-api/
- https://cloud.netapp.com/blog/aws-vs-azure-cloud-storage-comparison
- https://databricks.com/glossary/what-is-machine-learning-library
- http://h2o-release.s3.amazonaws.com/h2o/master/1283/docs-website/userguide/data.html
- https://www.quora.com/What-are-the-pros-and-cons-of-Spark-MLlib-vs-H2O
- https://www.linkedin.com/pulse/choosing-machine-learning-frameworks-apache-mahout-vs-debajani/
- https://www.tutorialspoint.com/spark_sql/spark_sql_quick_guide.htm